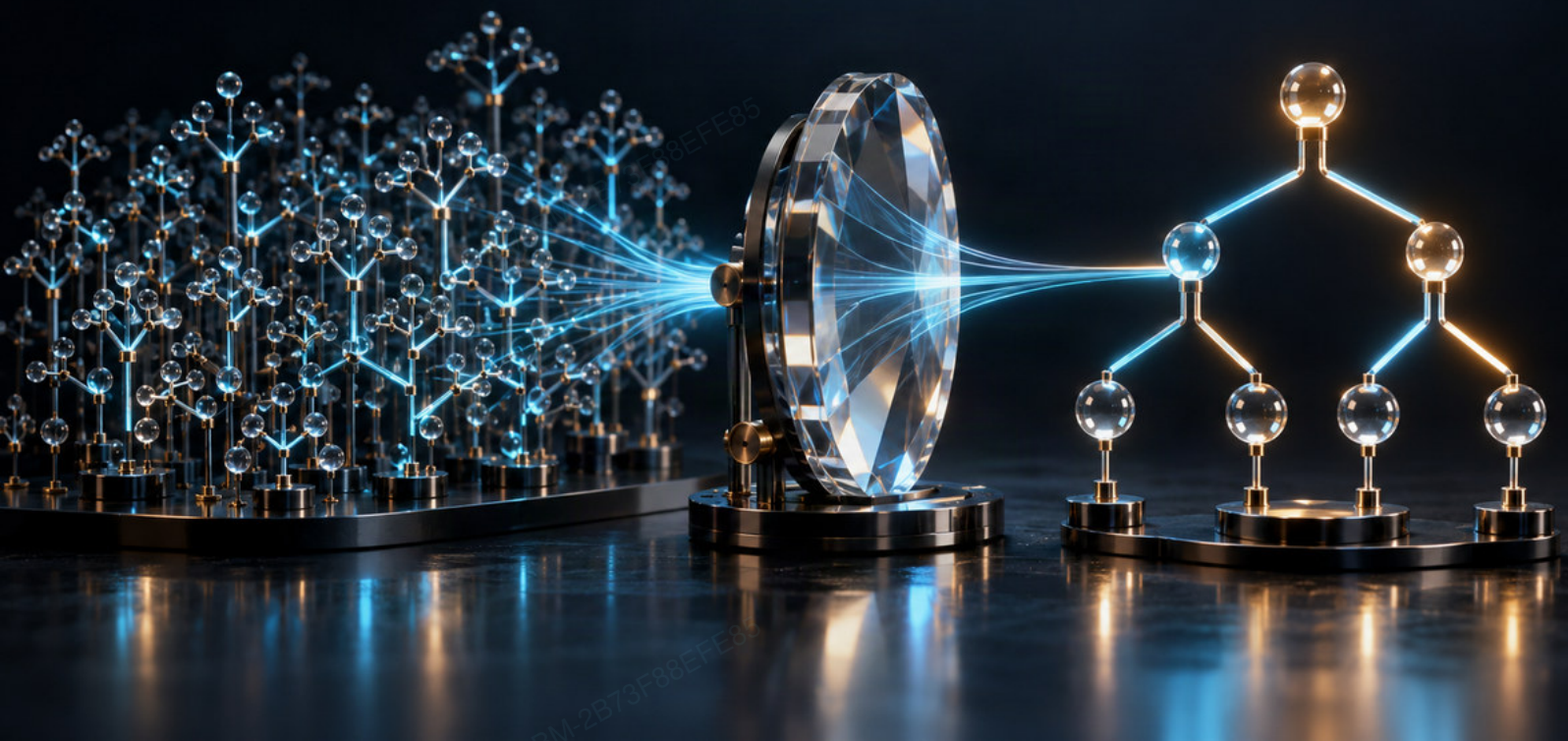




RESEARCH MANUSCRIPT | OCTOBER 4, 2026

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.

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 Mingus¹

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 check-pointing 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

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

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

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

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. $\square$

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 $O(m^2 n^2 (m + n))$ and memory $O(m^2 n^2)$ for an $m \times n$ grid. Optimality is relative

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

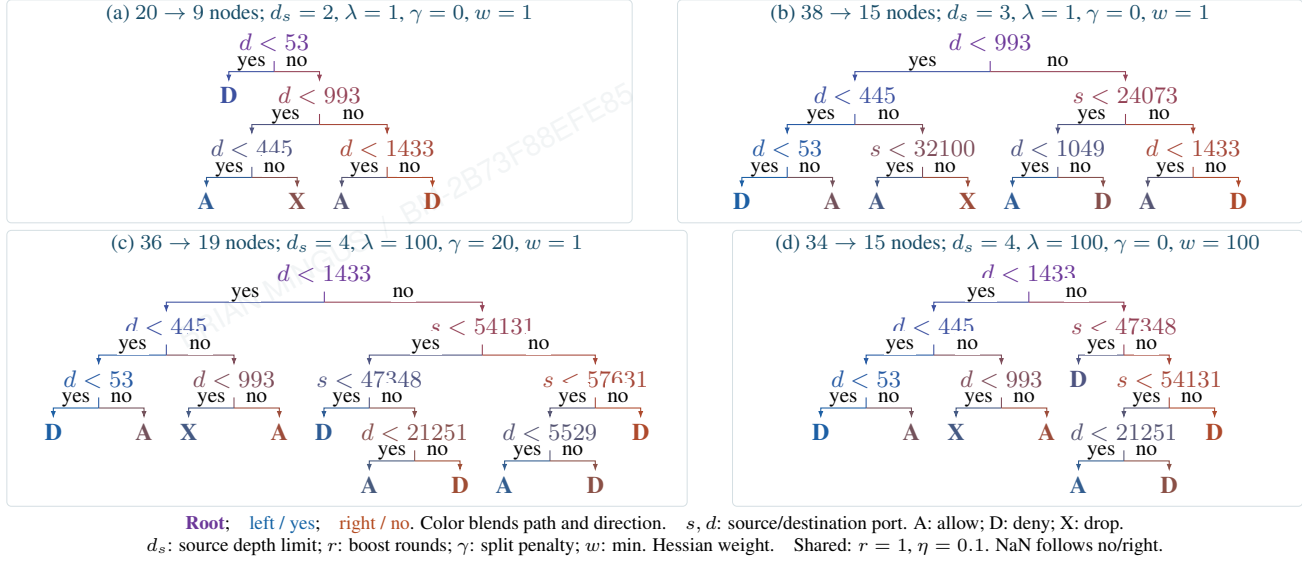


Figure 1. Complete firewall decision functions after exact minimization. The four trees have distinct branching structures and were selected from a 96-setting source-model sweep. Counts compare source-ensemble nodes with final physical-tree nodes. All predicates and leaves are shown; colors encode branch 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

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

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 $445 \leq d < 993$ 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).

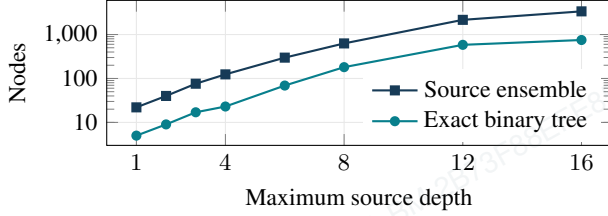


Figure 2. Completed conversions at fixed two boosting rounds (eight source trees) and fixed histogram budget. Connected observations show measured growth, not a fitted asymptotic law.

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

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

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 Coverttype 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 Coverttype 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.

References

- Blackard, J. Covtype [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](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.