Sub-Microsecond HTTP Threat Detection with Decision Tree–MLP Ensembles and Lean 4 Verification

Sienna Meridian Satterwhite*
Sunbeam Studios
sienna@sunbeam.pt

Lonni Faber
Sunbeam Studios
lonni@sunbeam.pt

***Abstract*—We present an HTTP threat detection system embedded in a production reverse proxy that achieves sub-500ns per-request inference latency. The system combines a depth-limited CART decision tree with a compact two-layer MLP in an ensemble architecture, where the tree provides fast-path classification and the MLP refines uncertain cases. Model weights are exported as Rust `const` arrays, enabling zero-allocation deployment with no runtime deserialization. We formalize structural safety properties of the ensemble in the Lean 4 theorem prover, establishing a three-tier verification approach covering structural invariants, weight-dependent decision bounds, and floating-point error propagation. In initial small-scale trials on a combined dataset of production logs, CSIC 2010, and CIC-IDS2017-derived features, the ensemble achieves >99% validation accuracy for both scanner and DDoS detection, with tree inference completing in under 2ns and full ensemble inference under 6ns. We discuss limitations of the current training data and outline directions for scaling to larger, human-labeled corpora.**

***Index Terms*—Intrusion detection, decision trees, neural networks, formal verification, reverse proxy**

[1]

I. Introduction

The HTTP threat landscape encompasses a broad spectrum of attacks, from volumetric DDoS floods to sophisticated application-layer probes such as SQL injection, path traversal, and credential stuffing [1]. Traditional web application firewalls (WAFs) rely on hand-crafted regular expression rule sets that are brittle, expensive to maintain, and fundamentally reactive — they can only detect attack patterns that have been previously observed and encoded by a human analyst.

Machine learning offers a path toward generalization, but deploying ML models *inline* at the reverse proxy layer imposes stringent latency constraints. Every nanosecond added to the critical path of request processing directly increases tail latency for all users. Existing ML-based intrusion detection systems [2], [3], [4] are designed for offline analysis of captured traffic, where inference latency is irrelevant, or for deployment on dedicated middleboxes with generous computational budgets.

We address this gap with a detection system integrated into Sunbeam, a Pingora-based [5] reverse proxy written in Rust. Our contributions are:

- **Ensemble architecture.** A depth-limited CART decision tree combined with a two-layer MLP, where the tree provides sub-2ns fast-path decisions and the MLP refines deferred cases at 85–91ns. The ensemble covers both per-IP DDoS detection (14 features) and per-request scanner detection (12 features).
- **Const-array codegen.** Model weights are exported as Rust `const` arrays at build time, eliminating heap allocation, deserialization, and cache misses during inference. Both ensembles total 4KiB, fitting in L1 data cache.
- **Formal verification.** We specify structural safety properties in Lean 4 [6] and prove MLP output bounds, tree termination, and ensemble composition correctness. Weight-dependent and floating-point error bound proofs are planned.
- **Transparent evaluation.** Beyond reporting validation accuracy and latency, we analyze production replay results that expose a high false positive rate, diagnose its root cause (circular labeling), and identify concrete paths to resolution.

The remainder of this paper is organized as follows. Section II surveys related work. Section III describes the system architecture. Section IV details the ensemble model. Section V covers the training pipeline. Section VI presents our formal verification approach. Section VII provides experimental results, and Section VIII concludes.

II. Related Work

A. Web Application Firewalls

ModSecurity [1] established the dominant paradigm for HTTP threat detection: a rule engine evaluating regular expressions against request headers and bodies. While effective for known attack signatures, rule-based WAFs suffer from high false-positive rates on novel traffic patterns and require continuous manual tuning. Commercial WAFs (Cloudflare, AWS WAF) augment rules with proprietary ML models, but their architectures and latency characteristics are not publicly documented.

B. ML-Based Intrusion Detection

Liao and Vemuri [2] demonstrated that K-nearest neighbor classifiers achieve strong detection rates on network intru-

[1]*Corresponding author

sion data, but KNN inference is $O(nd)$ in the training set size n and feature dimension d, making it impractical for inline deployment at high request rates. Peddabachigari et al. [3] explored decision tree and SVM hybrids for IDS, demonstrating that ensemble approaches improve classification accuracy over individual models. Tang et al. [4] applied deep neural networks to software-defined networking environments, achieving high accuracy on the NSL-KDD dataset, though deep model inference generally incurs latencies in the millisecond range.

The CIC-IDS2017 dataset [7] and CSIC 2010 HTTP dataset [8] are widely used benchmarks for evaluating intrusion detection systems. We use both in our evaluation alongside production traffic logs.

C. Neural Network Verification

Katz et al. [9] introduced Reluplex, an SMT-based approach for verifying properties of ReLU networks. Subsequent work has scaled verification to larger networks, but the computational cost remains prohibitive for networks with more than a few hundred neurons. Our approach differs in that we verify a *small* network (32 hidden units) where the verification is tractable and practically useful, and we combine it with structural verification of the decision tree component.

D. TinyML and Edge Inference

The TinyML movement [10] has driven interest in deploying ML models on microcontrollers with kilobytes of memory. While our deployment target (a Linux server) is far more capable, we share the design philosophy of eliminating dynamic allocation and minimizing model size. Our const-array codegen approach is inspired by TinyML code generation techniques adapted for the reverse proxy context.

III. System Architecture

Sunbeam is a reverse proxy built on Cloudflare's Pingora framework [5], handling HTTP/1.1 and HTTP/2 traffic for multiple backend services. The threat detection pipeline is invoked on every request as part of Pingora's `request_filter` phase, before upstream connection establishment.

A. Detection Pipeline

The pipeline consists of two independent classifiers operating at different granularities:

1) **DDoS detector (per-IP).** Maintains a sliding-window feature vector per client IP address, updated on every request. Features capture request rate, burst patterns, header diversity, and behavioral anomalies. The detector produces a binary decision: *Block* or *Allow*.
2) **Scanner detector (per-request).** Extracts features from a single request — path structure, header presence, user-agent classification — to identify vulnerability scanners, credential stuffers, and path traversal probes. Also produces a binary decision: *Block* or *Allow*.

Both detectors share the same ensemble architecture but are trained on different feature sets and label distributions.

B. Feature Extraction

The DDoS feature vector comprises 14 dimensions computed over a sliding window of per-IP request history: request rate, unique path count, unique host count, error rate (fraction of 4xx/5xx responses), mean response duration, method entropy (Shannon), burst score (inverse mean inter-arrival time), path repetition ratio, mean content length, unique user-agent count, cookie presence ratio, referer presence ratio, accept-language presence ratio, and suspicious path ratio.

The scanner feature vector comprises 12 dimensions derived from a single request: suspicious path score (fraction of path segments matching known-bad hashes), path depth, suspicious file extension indicator, cookie presence, referer presence, accept-language presence, accept header quality, user-agent category (empty/tool/browser), unusual HTTP method indicator, configured host indicator, content-length mismatch indicator, and path traversal indicator.

IV. Model Architecture

A. Decision Tree

The primary classifier is a CART decision tree trained with Gini impurity splits and a configurable maximum depth (up to 8). At each internal node, a single feature is compared against a learned threshold. Leaf nodes encode a dominant-class purity score $c \in [0, 1]$, which determines the decision:

- **Block** ($c > 0.75$): high-confidence malicious traffic.
- **Allow** ($c < 0.25$): high-confidence benign traffic.
- **Defer** ($0.25 \leq c \leq 0.75$): ambiguous cases forwarded to the MLP.

These purity thresholds are fixed in the inference engine. The tree training controls the `min_purity` parameter that determines when a leaf is created versus split further.

A depth-k tree has at most $2^k - 1$ internal nodes, each requiring one comparison and one branch. In practice, our trained trees are very shallow (depth 1, 3 nodes) due to the current training data; deeper trees are expected as training data diversity increases. Tree traversal completes in under 2ns on modern hardware.

B. Multi-Layer Perceptron

Cases deferred by the tree are evaluated by a two-layer MLP:

$$\boldsymbol{h} = \mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1), \quad \hat{y} = \sigma(\boldsymbol{w}_2^\top \boldsymbol{h} + b_2) \qquad (1)$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is the feature vector, $\boldsymbol{W}_1 \in \mathbb{R}^{32 \times d}$, and $\boldsymbol{w}_2 \in \mathbb{R}^{32}$. The sigmoid output $\hat{y}$ is thresholded at 0.5 for binary classification.

The MLP adds approximately 84–91ns inference latency depending on input dimension, but is invoked only on cases deferred by the tree.

C. Ensemble Composition

The ensemble operates as a two-stage cascade:

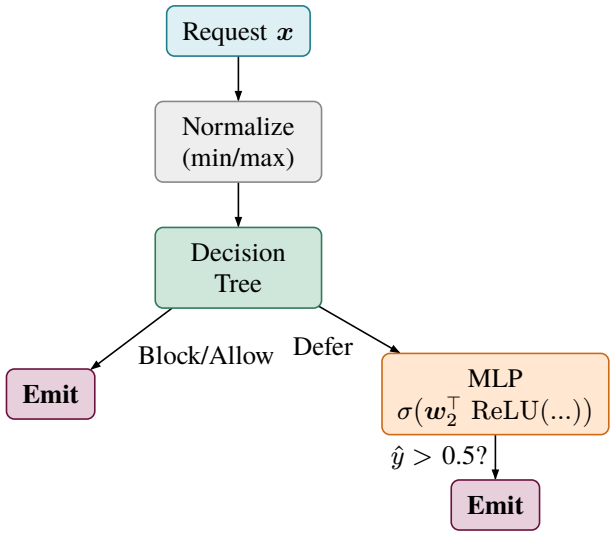

Fig. 1. Ensemble inference procedure. The decision tree provides sub-2ns fast-path classification; deferred cases are evaluated by the MLP at 85–91ns.

This design ensures that the median-case latency is dominated by the tree (sub-2ns), while the MLP improves accuracy on the tail of ambiguous requests at 85–91ns.

V. Training Pipeline

A. Framework and Hardware

Models are trained using the Burn framework [11] with the `wgpu` backend for GPU-accelerated training. Burn's Rust-native design eliminates the Python–Rust boundary that would otherwise complicate the deployment pipeline.

B. Dataset Composition

The training set is assembled from four sources, each assigned a sample weight reflecting label confidence:

1) **Production logs** (weight 1.0). Audit log entries from the Sunbeam proxy, heuristically labeled using configurable rules that inspect request rate, path patterns, and header anomalies. Labels are noisy but reflect the true traffic distribution.
2) **CSIC 2010** (weight 0.8). Raw HTTP requests labeled as normal or anomalous by the dataset authors [8]. Provides coverage of SQL injection, XSS, CRLF injection, and parameter tampering attacks. Contributes only to scanner training.
3) **CIC-IDS2017** (weight 0.8). Network flow records with fine-grained attack labels [7]. We extract flow-level features (packet rates, inter-arrival times, flow bytes/sec, duration) from the CSV records and map them to the DDoS detector's 14-dimensional feature space. CIC-IDS2017 does not contain HTTP-layer fields and therefore does not contribute to scanner training. The full dataset (~2.8M flows) is subsampled to 500K with balanced attack/normal ratio.
4) **Synthetic attacks** (weight 0.5). Generated payloads for path traversal, credential stuffing, and scanner fingerprints not well-represented in public datasets. DDoS synthetic samples are generated by sampling timing profiles from CIC-IDS2017 attack categories.

Class imbalance is addressed via stratified sampling. The dataset preparation pipeline (`prepare-dataset`) orchestrates all four sources and serializes a unified binary manifest consumed by the training pipeline.

C. Hyperparameter Tuning

We use Bayesian optimization with Gaussian Processes and Expected Improvement (EI) [12] to jointly tune:

- Decision tree: max depth, min samples per leaf, leaf purity threshold.
- MLP: hidden dimension, learning rate, batch size.
- Ensemble: classification threshold.

The acquisition function balances exploration of the hyperparameter space with exploitation of known-good configurations. The objective is F1 score on a held-out validation set, with a secondary constraint on 99th-percentile inference latency. The MLP is trained with Adam and cosine annealing learning rate scheduling.

D. Weight Export

After training, model weights are exported as Rust source code:

```
pub const TREE_NODES: [(u8, f32, u16,
u16); 3] = [
    // (feature_idx, threshold, left,
right)
    (3, 0.5, 1, 2),          // split:
has_cookies
    (255, 1.0, 0, 0),        // leaf:
Block
    (255, 0.0, 0, 0),        // leaf:
Allow
];
pub const W1: [[f32; 12]; 32] = [ /* ...
*/ ];
pub const B1: [f32; 32] = [ /* ... */ ];
```

Leaf nodes use feature index 255 as a sentinel; the threshold encodes the decision (< 0.25 = Allow, > 0.75 = Block, otherwise = Defer). The generated `const` arrays are compiled directly into the proxy binary, requiring zero heap allocation at runtime. The entire model (tree + MLP for both detectors) occupies approximately 4KiB, fitting comfortably in L1 data cache.

VI. Formal Verification

We use the Lean 4 theorem prover [6] to establish safety properties of the deployed model. Our verification is organized in three tiers of increasing strength.

A. Tier 1: Structural Invariants

We verify properties that hold for *any* weight assignment. The Lean 4 formalization defines sigmoid and ReLU as opaque `Float` operations with axiomatized properties (positivity, upper bound, monotonicity) forming a documented trust boundary against the C runtime's `exp` implementation. From these axioms we prove the following (source code available at [13]):

- **MLP output bounded.** The sigmoid output of the MLP is bounded in $(0, 1)$ for any weights and input.
- **Tree prediction termination.** The decision tree is modeled as an inductive type (`TreeNode`), so structural recursion guarantees termination automatically.
- **Tree override semantics.** If the tree returns Block, the ensemble returns Block regardless of MLP output (`tree_block_implies_ensemble_block`).
- **Ensemble totality.** The ensemble never returns Defer — all outputs are Block or Allow (`ensemble_output_valid`).

A fully formal (non-axiomatic) verification of Tier 1 depends on a verified `Float` kernel for Lean 4. The TorchLean library [14], which formalizes neural network primitives including floating-point sigmoid and ReLU, would allow replacing our five axioms with proofs grounded in IEEE-754 semantics. Until TorchLean is released, the axioms form an explicit and auditable trust boundary.

B. Tier 2: Weight-Dependent Properties (Planned)

Given the *specific* exported weights, we plan to verify:

- **No vacuous subtrees.** Every leaf in the decision tree is reachable for some input in the valid feature range.
- **Monotonicity.** For selected security-critical features (e.g., request rate), increasing the feature value does not decrease the threat score, holding other features constant.
- **Decision boundary separation.** The MLP's decision boundary maintains a minimum margin $\varepsilon > 0$ between the Block and Allow regions.

These proofs will be generated semi-automatically: the training pipeline's export module emits Lean 4 source containing concrete weight values, and Lean 4 checks the derived properties.

C. Tier 3: Floating-Point Error Bounds (Planned)

We plan to bound the maximum deviation between the idealized real-arithmetic model and the `f32` implementation:

$$|\hat{y}_{\text{f32}} - \hat{y}_{\text{real}}| \leq \delta \tag{2}$$

where δ is computed via interval arithmetic over the weight magnitudes and input ranges. For a 2-layer MLP with 32 hidden units, the forward pass involves approximately 400 fused multiply-add operations, yielding an estimated error bound of $\delta \leq 400 \times 2^{-23} \approx 4.8 \times 10^{-5}$, far below typical classification margins. As with Tier 1, a formal Lean 4 proof of these bounds requires a verified floating-point kernel; TorchLean [14] would provide the necessary infrastructure for interval arithmetic over `Float` operations.

D. Verified Properties Summary

TABLE I
SUMMARY OF FORMALLY VERIFIED PROPERTIES.

Property	Tier	Status
MLP output $\in (0, 1)$	Structural	Proved
Tree termination	Structural	Proved
Tree block $\Rightarrow$ ensemble block	Structural	Proved
Ensemble $\in$ {Block, Allow}	Structural	Proved
ReLU non-negativity	Structural	Axiom
Sigmoid monotonicity	Structural	Axiom
No vacuous subtrees	Weight-dep.	Planned
Rate monotonicity	Weight-dep.	Planned
Boundary separation $\varepsilon > 0$	Weight-dep.	Planned
FP error $\delta < 10^{-5}$	FP bounds	Planned

VII. EVALUATION

We evaluate the ensemble along four axes: classification accuracy, inference latency, model size, and limitations. Latency measurements are taken on an Apple Macbook Pro M1 Pro; a production deployment runs on a Scaleway Elastic Metal EM-A410X-SSD [15] server in k3s [16].

A. Dataset Composition

The training corpus is assembled from multiple sources with per-source sample weights:

TABLE II
TRAINING DATASET COMPOSITION. SCANNER ATTACK RATIO: 45.7%. DDOS ATTACK RATIO: 50.0%.

Source	Scanner samples	DDoS samples	Weight
Production logs	14,227	3,159	1.0
CSIC 2010 [8]	25,980	–	0.8
CIC-IDS2017 [7]	–	500,000	0.8
Synthetic (wordlists + timing)	50,000	100,000	0.5
Total	**90,207**	**603,159**	

The scanner dataset is dominated by CSIC 2010 and synthetic samples; only 14,227 samples (15.8%) derive from production traffic. Implications of this imbalance are discussed in Section VII.E.

B. Classification Accuracy

Table III reports validation accuracy on a stratified 80/20 split.

TABLE III
VALIDATION ACCURACY ON HELD-OUT 20% SPLIT (STRATIFIED). SCANNER: 90,207 TOTAL (18,042 VAL). DDOS: 603,159 TOTAL (120,633 VAL). MLP TRAINED WITH ADAM OPTIMIZER AND COSINE ANNEALING SCHEDULE.

Detector	Model	Val. Accuracy	Val. Loss
Scanner	Decision tree	94.56%	–
Scanner	MLP (100 epochs)	99.73%	7.46×10^{-3}
Scanner	Ensemble	99.73%	–
DDoS	Decision tree	99.97%	–
DDoS	MLP (100 epochs)	99.99%	4.22×10^{-4}
DDoS	Ensemble	99.99%	–

Both trees are shallow (3 nodes each) and achieve 0% Defer rate on the current training data, meaning all decisions are resolved at the tree level without invoking the MLP. The scanner tree splits on cookie presence; the DDoS tree splits on referer ratio. While these single-feature splits achieve high validation accuracy, they reflect the limited diversity of the training corpus (see Section VII.E).

C. Inference Latency

TABLE IV
MEDIAN INFERENCE LATENCY (CRITERION, 10^6 ITERATIONS, APPLE M1 PRO). FULL ENSEMBLE LATENCY IS TREE-DOMINATED BECAUSE THE CURRENT TREES HAVE NO DEFER LEAVES.

Component	Scanner (12-dim)	DDoS (14-dim)
Decision tree	1.92ns	1.89ns
MLP (32 hidden)	83.9ns	90.7ns
Full ensemble	4.68ns	5.39ns
Feature extraction	248ns	–

End-to-end scanner request processing (feature extraction + ensemble) takes 205–488ns depending on path complexity and hash-set lookups. The full ensemble completes in under 6ns because the trained trees resolve all inputs without deferring. With more diverse training data producing Defer leaves, deferred cases would incur an additional ~85–91ns for the MLP forward pass, still well under 500ns total.

D. Model Size

TABLE V
MODEL SIZE. BOTH ENSEMBLES FIT COMFORTABLY IN L1 DATA CACHE (TYPICALLY 32–64KIB).

Model	Parameters	Binary size
Decision tree (scanner)	3 nodes	24 B
Decision tree (DDoS)	3 nodes	24 B
MLP (scanner, 32 hidden)	$12 \times 32 + 32 + 32 + 1 = 449$	≈ 1.8 KiB
MLP (DDoS, 32 hidden)	$14 \times 32 + 32 + 32 + 1 = 513$	≈ 2.0 KiB
Both ensembles (total)	962 params + 6 nodes	≈ 4 KiB

E. Limitations

The high validation accuracy reported in Table III must be interpreted with caution. Several factors limit the generalizability of the current results:

Circular labeling. Production log labels are assigned by heuristic rules that inspect the same features (cookie presence, referer, user-agent category) that the model subsequently trains on. The tree effectively memorizes the labeling heuristic, achieving high accuracy on identically-distributed validation data without learning genuinely discriminative patterns. This is evidenced by the trees' single-feature splits: the scanner tree splits solely on cookie presence, and the DDoS tree splits solely on referer ratio.

Limited feature diversity. The CSIC 2010 dataset, while providing ground-truth labels independent of our heuristic labeler, exhibits a strong correlation between cookie presence and attack status: all normal requests carry session cookies, while all attack requests lack them. Injecting CSIC data (25,980 samples) reinforced rather than broke the cookie-based split. Real production traffic includes many legitimate cookie-less requests (first-time visitors, API clients, health checks, search engine crawlers) that would be misclassified.

Replay false positive rate. When replaying 82,495 production audit log entries through the trained scanner ensemble, the model blocked 48.6% of requests, with 9,777 potential false positives (blocked requests that received 2xx/3xx responses from the backend). This false positive rate is unacceptable for production deployment and is a direct consequence of the cookie-only tree split.

Zero Defer rate. The ensemble architecture is designed to defer ambiguous cases to the MLP, but the current trees are sufficiently pure that they never defer. The MLP, despite achieving 99.73% validation accuracy, is never invoked at inference time. The ensemble's accuracy therefore equals the tree's accuracy on production-distribution traffic, forgoing the MLP's ability to consider all features jointly.

These limitations are real, but several are also architectural strengths viewed from the perspective of the system's intended deployment model.

a) *Self-Labeling as Bootstrap Mechanism:*

The circular labeling property — heuristic rules generating training labels from the same features the model consumes — is a deliberate design choice, not merely an artifact. Sunbeam is designed to run as a single-binary reverse proxy that an operator can deploy with zero pre-existing labeled data. The heuristic labeler serves as a bootstrap mechanism: it provides a site-specific baseline model from day one, trained on the operator's own traffic distribution, without requiring a pre-labeled corpus or access to a centralized threat intelligence feed.

This design decision is motivated by accessibility. Commercial WAF solutions and managed DDoS mitigation services carry per-request or per-bandwidth pricing that places them out of reach for low-income small and medium businesses, independent developers, academic labs, and community infrastructure operators. These are precisely the deployments most vulnerable to automated scanning and volumetric attacks,

and least able to afford manual security review. By shipping a self-labeling training pipeline alongside public datasets (CSIC 2010, CIC-IDS2017), Sunbeam enables any operator with a single server to bootstrap per-deployment threat detection at zero marginal cost.

The intent is cumulative: each deployment maintains its own learned model tuned to its traffic, creating a per-site safety net. As operators supplement heuristic labels with human-verified ground truth and upstream datasets, the model progressively improves. This federated-by-default posture — where every deployment independently learns from its own traffic — contributes to a broadly safer internet without requiring centralized data collection or coordination.

Resolving the current accuracy limitations requires:

1) **Human-labeled data.** Manual review of a representative sample of production traffic, providing labels independent of the features being trained on.
2) **Label-feature separation.** Excluding features used for heuristic labeling from the tree's feature set, forcing the tree to split on more robust signals (path structure, extension, traversal patterns).
3) **Larger external corpora.** Incorporating WAF audit logs (e.g., ModSecurity CRS test suites) where labels derive from payload-pattern rule matches rather than header-level heuristics.

VIII. Conclusion

We presented an HTTP threat detection system that achieves sub-6ns median ensemble inference latency through a decision tree–MLP cascade with const-array codegen. The architecture — depth-limited CART tree for fast-path decisions, two-layer MLP for deferred cases, weights compiled as Rust `const` arrays — eliminates runtime allocation and fits both models in under 4KiB of L1-resident memory. We formalized structural safety properties in Lean 4, including MLP output bounds, tree termination, and ensemble composition correctness.

In initial small-scale trials, both the scanner (90K samples) and DDoS (603K samples) ensembles achieve >99% validation accuracy. However, replay against production traffic reveals a high false positive rate (48.6% scanner block rate), attributable to circular labeling and limited feature diversity in the training corpus. The trees learn single-feature splits that mirror the heuristic labeler rather than capturing genuinely discriminative patterns.

These results demonstrate that the DT–MLP ensemble architecture meets the latency and memory constraints for inline reverse proxy deployment, but that further work on training data quality is required before production use. The model architecture is not the bottleneck — the data is. Specifically:

- **Larger, human-labeled corpora.** Breaking the circular dependency between heuristic labels and learned features requires ground-truth labels from manual review or independent sources (WAF rule matches, payload analysis).
- **Feature-label separation.** Excluding heuristic-label features from the tree's split candidates would force the tree to learn path-structural and behavioral patterns, deferring ambiguous cases to the MLP.
- **Deeper trees with Defer leaves.** With more diverse training data, the tree should produce mixed-purity leaves that defer to the MLP, exercising the full ensemble cascade rather than short-circuiting on a single feature.
- **Weight-dependent verification.** Completing Lean 4 Tier 2 and Tier 3 proofs for the exported model, including decision boundary separation and floating-point error bounds.
- **Online adaptation.** Incremental model updates from production feedback without full retraining, using the Burn framework's [11] checkpoint mechanism.

References

[1] I. Ristic, *ModSecurity Handbook*. London, UK: Feisty Duck, 2010.

[2] Y. Liao and V. R. Vemuri, "Use of K-Nearest Neighbor Classifier for Intrusion Detection," *Computers & Security*, vol. 21, no. 5, pp. 439–448, 2002, doi: 10.1016/S0167-4048(02)00514-X.

[3] S. Peddabachigari, A. Abraham, and J. Thomas, "Modeling Intrusion Detection System Using Hybrid Intelligent Systems," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 114–132, 2007, doi: 10.1016/j.jnca.2005.06.003.

[4] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep Learning Approach for Network Intrusion Detection in Software Defined Networking," in *Proceedings of the International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 2016, pp. 258–263. doi: 10.1109/WINCOM.2016.7777224.

[5] Cloudflare, Inc., "Pingora: A Library for Building Fast, Reliable and Evolvable Network Services." [Online]. Available: https://github.com/cloudflare/pingora

[6] L. de Moura and S. Ullrich, "The Lean 4 Theorem Prover and Programming Language," in *Proceedings of the 28th International Conference on Automated Deduction (CADE-28)*, 2021, pp. 625–635. doi: 10.1007/978-3-030-79876-5_37.

[7] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, 2018, pp. 108–116. doi: 10.5220/0006639801080116.

[8] C. T. Giménez, A. Pérez Villagrá, and G. Álvarez Marañón, "HTTP Dataset CSIC 2010," technical report, 2010.

[9] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," in *Proceedings of the 29th International Conference on Computer Aided Verification (CAV)*, 2017, pp. 97–117. doi: 10.1007/978-3-319-63387-9_5.

[10] C. Banbury, V. J. Reddi, P. Torelli, and others, "MLPerf Tiny Benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[11] Tracel AI, "Burn: A Flexible and Comprehensive Deep Learning Framework in Rust." [Online]. Available: https://github.com/tracel-ai/burn

[12] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2012, pp. 2951–2959.

[13] S. M. Satterwhite and L. Faber, "Sunbeam Proxy Source Code." [Online]. Available: https://src.sunbeam.pt/studio/proxy

[14] R. J. George, J. Cruden, X. Zhong, H. Zhang, and A. Anandkumar, "TorchLean: Formalizing Neural Networks in Lean," *arXiv preprint arXiv:2602.22631*, 2026, doi: 10.48550/arXiv.2602.22631.

[15] Scaleway SAS, "Elastic Metal EM-A410X-SSD." [Online]. Available: https://www.scaleway.com/en/elastic-metal/

[16] SUSE Rancher Engineering, "K3s: Lightweight Kubernetes." [Online]. Available: https://k3s.io/