

EnsembleClassifier: Using Machine Learning to Select Exact Packet Classification Structures for Partitioned Rule Subsets

Ashesh Gaur, Sridhar Radhakrishnan and Mohammed Atiquzzaman
School of Computer Science, University of Oklahoma, Norman, OK, USA
Emails: {asheshgaur, sridhar, atiq}@ou.edu

Abstract—Packet classification is a core bottleneck in high-speed networks, where packets must be matched against large and heterogeneous rule sets under strict latency and memory constraints. Existing classifiers apply a single data structure across the entire ruleset, failing to adapt to local geometric diversity and resulting in suboptimal performance.

We present *EnsembleClassifier*, a new architecture that decouples rule-space partitioning from structure selection. The system decomposes the ruleset using connected-component analysis, producing disjoint subsets without rule replication. Each subset is assigned an exact classifier chosen from a portfolio using a learned model trained on Classbench rulesets with exhaustive performance labels. Correctness is guaranteed by construction: learning affects only structure selection, while packet matching remains exact and preserves first-match semantics.

Across ClassBench rulesets from 1k to 64k sizes, we generated approximately 70k feature vectors. These feature vectors were used to decide which algorithm was most suitable for the given feature vector. The composition tree then uses multiple algorithms for a given ruleset. Our composition tree reduced average per-packet classification latency by 46%, memory footprint by 99.14%, and build time by 98.65%, in each case measured against the strongest baseline in our portfolio for that metric.

These results show that treating packet classification as a composition problem enables adaptive, structure-aware designs that significantly improve performance while preserving strict correctness guarantees. More broadly, *EnsembleClassifier* demonstrates how learned decision-making can be embedded directly into network data-plane infrastructure without relaxing the exactness guarantees such infrastructure demands.

I. INTRODUCTION

ROUTERS and firewalls must classify packets at line rate. Each packet is matched against a ruleset, and the highest-priority matching rule determines the action—permit, deny, or forward [1], [2]. Modern rulesets contain thousands to tens of thousands of rules [3], [4], and software-defined networking (SDN) introduces frequent updates via systems such as OpenFlow [5] and Open vSwitch [6]. A practical classifier must therefore support both high-throughput lookups and dynamic updates.

Existing packet classifiers employ a range of techniques, including space partitioning [1], [7], boundary-based splitting [8], [9], hybrid approaches [4], [10], [11], and hashing-based methods [2], [12]. However, all apply a single structure globally, failing to adapt to local heterogeneity in rule distributions.

Real-world rulesets exhibit significant variation across the header space. Some regions contain dense overlapping ranges that benefit from hierarchical partitioning, while others contain sparse or exact-match rules where hashing or even linear scanning is more efficient. Applying a single structure globally forces a trade-off that is suboptimal for at least part of the ruleset.

A second limitation arises from rule representation. Many classifiers assume CIDR-aligned prefixes to simplify partitioning [1], [4], [7], [8]. In practice, operational rules frequently include arbitrary numerical ranges due to policy aggregation and application constraints [13], [14]. This mismatch, which we refer to as the *CIDR gap*, leads to inefficiencies in real deployments.

We present *EnsembleClassifier*, a packet classification framework that addresses both heterogeneity and the CIDR gap. The key idea is to decouple rule-space decomposition from structure selection. We recursively partition the ruleset using connected-component analysis across header fields, producing disjoint subsets without rule replication. For each subset, we select the most efficient exact classifier structure from a portfolio of candidates using a learned model trained on Classbench generated workloads.

The resulting classifier is a composition of exact structures. The learned model influences only structure selection and does not replace packet matching logic. Unlike neural classifiers that approximate decisions [15]–[17], *EnsembleClassifier* preserves exactness. We formally prove that correctness is guaranteed regardless of the model’s decisions.

This design enables localized specialization while maintaining global correctness. By separating partitioning from structure selection, *EnsembleClassifier* adapts to the geometric properties of rules in each region without introducing approximation.

Figure 1 illustrates the overall *EnsembleClassifier* workflow, from connected-component-based partitioning to learned structure selection and final packet classification. During construction, the input ruleset is recursively decomposed into terminal subsets using connected-component partitioning. For each subset, a feature vector is extracted and passed to the learned selector, which chooses the most efficient exact classifier structure. These structures are assembled into a composition

tree. During deployment, packets traverse routing nodes to the appropriate terminal node, where exact packet matching is performed.

This work sits at the intersection of machine learning and network systems. Routers and firewalls are increasingly expected to adapt their internal organization to the workloads they carry, a goal shared with the broader move toward self-optimizing network infrastructure. Software-defined networking has made the control plane programmable, yet the data-plane structures that actually match packets are still selected statically at design time. EnsembleClassifier treats that selection as a learned decision: a supervised model observes the geometric signature of each rule subset and chooses the structure best suited to it. What separates this from typical learning-based networking systems is that the learned component is advisory rather than authoritative—it governs how packets are searched, never what the answer is. Exactness is a property of the composition tree, not of the model, so the system gains adaptivity without inheriting the reliability concerns that normally accompany learned components in critical infrastructure.

The contributions of this paper are as follows:

- **A new formulation of packet classification as a composition problem.** We show that classifier performance is governed by the local geometric properties of rule subsets rather than global ruleset characteristics, motivating a shift from single-structure designs to adaptive composition.
- **EnsembleClassifier: a compositional architecture with no rule replication.** We introduce a hierarchical classifier that decomposes the ruleset using connected-component partitioning, producing disjoint subsets and eliminating replication.
- **Decoupling correctness from learning.** We design a system in which learning is used only for structure selection, while correctness follows from the composition tree and routing invariants. This guarantees exact first-match semantics independent of the learned model.
- A supervised learning formulation for structure selection. We cast structure selection as a local classification problem over terminal rule subsets and train a model using Classbench generated connected rulesets with exhaustive performance labeling.
- Significant improvements in latency and memory. Across ClassBench and arbitrary-range workloads, EnsembleClassifier achieves up to **1.85×** faster classification speed than the fastest classifier, PartitionSort, **99.14%** lower memory footprint compared to PriorityTuple and **98.65%** lower build time compared to PriorityTuple.

The remainder of this paper is organized as follows. Section II formalizes the packet classification problem. Section III presents the construction and correctness of EnsembleClassifier. Section IV describes the structure selection model. Section V evaluates performance. Section VI concludes.

II. PROBLEM DEFINITION

In this section, we define the problem of packet classification. We then define the problem of decomposition of multidimensional search space, and our connected components approach to tackle it (Fig. 1).

A. Packet Classification

A ruleset $\mathcal{R} = \{R_1, \dots, R_n\}$ consists of n rules defined over d packet header fields. Each rule R_i specifies an interval in each field:

$$R_i = \{[f_{i_{\min}}^1, f_{i_{\max}}^1], \dots, [f_{i_{\min}}^d, f_{i_{\max}}^d]\},$$

along with a priority p_i and an action a_i .

A packet P matches R_i if:

$$\forall k \in \{1, \dots, d\}, \quad f_{i_{\min}}^k \leq P[k] \leq f_{i_{\max}}^k.$$

The classification problem is to return the highest-priority matching rule:

$$R^* = \arg \max_{R_i \in \mathcal{R}, R_i \text{ matches } P} p_i.$$

Geometrically, each rule corresponds to an axis-aligned hyper-rectangle, and packet classification reduces to the Orthogonal Point Enclosure Query (OPEQ) problem [18], [19].

B. Decomposition Challenges

Packet classification is inherently multidimensional, requiring simultaneous constraints across all d fields. Most classifiers decompose this problem by partitioning along selected dimensions. However, poor partitioning choices can lead to significant rule replication, increasing both memory usage and lookup latency [13], [14]. Thus, classifier performance is highly sensitive to how the ruleset is decomposed.

C. Heterogeneity of Rule Subsets

Let $\mathcal{R}_v \subseteq \mathcal{R}$ denote a subset obtained through recursive partitioning. Different subsets exhibit distinct geometric properties:

- dense overlapping ranges favoring partitioning structures,
- exact-match dominated regions favoring hashing,
- mixed intervals favoring hybrid methods, and
- small subsets where linear scanning is sufficient.

No single classifier structure performs well across all such cases. This suggests that performance is governed by local subset properties rather than global ruleset characteristics.

D. Structure Selection Problem

We formalize structure selection as follows.

Input: ruleset $\mathcal{R}$, training trace $\mathcal{T}$, and a portfolio of exact classifier structures $\mathcal{S} = \{S_1, \dots, S_m\}$.

Goal: Construct a composition tree $\mathcal{C}$ such that:

- 1) correctness is preserved (exact highest-priority match),
- 2) latency, memory, and build time are minimized, and
- 3) performance generalizes to unseen traces.

Approach. The construction consists of two phases:

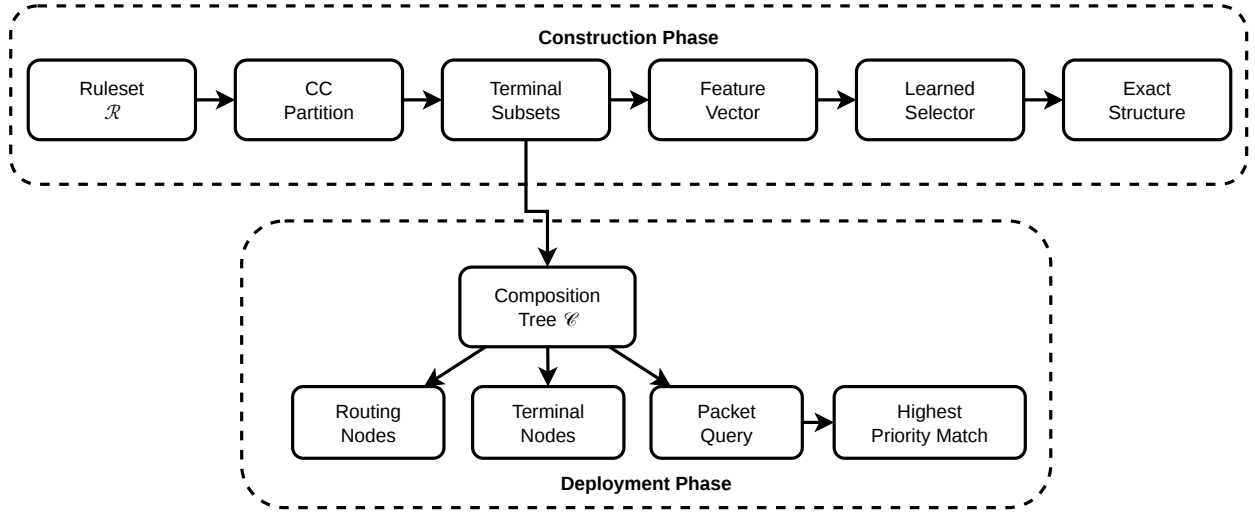


Fig. 1: EnsembleClassifier overview

- 1) **Partitioning:** Recursively decompose $\mathcal{R}$ by selecting dimensions that yield multiple connected components, stopping when no further separation is possible or subset size falls below a threshold.
- 2) **Structure Selection:** For each terminal subset $\mathcal{R}_v$, select $S_j \in \mathcal{S}$ using a learned model based on local geometric and traffic features.

Structure selection is independent across subsets, reducing the problem to supervised learning over local regions. The central challenge is to learn a mapping from subset properties to the classifier structure that minimizes classification latency.

III. ENSEMBLECLASSIFIER SYSTEM

EnsembleClassifier constructs a composite packet classifier by recursively partitioning the ruleset into connected components and assigning a classifier structure to each resulting subset.

The design separates two concerns: (1) decomposition of the ruleset via connected-component partitioning, and (2) structure selection using the learned model (Section IV).

This separation ensures that learning influences only structure choice, while packet matching remains exact.

A. Overview

Given ruleset $\mathcal{R}$ and packet trace $\mathcal{T}$, the system builds a composition tree $\mathcal{C}$.

Each node v contains a rule subset $\mathcal{R}_v \subseteq \mathcal{R}$ and a packet subset $\mathcal{P}_v \subseteq \mathcal{T}$.

Nodes are either:

- **Routing nodes:** partition packets based on a selected field, or
- **Terminal nodes:** contain an exact classifier over $\mathcal{R}_v$.

Each rule appears in exactly one terminal node, eliminating replication.

B. Construction Procedure

Tree construction is recursive (Algorithm 1) and consists of four steps:

- 1) remove bypass fields,
- 2) check terminal conditions,
- 3) select a split field,
- 4) recurse or build a terminal node.

Algorithm 1 BuildTree: Composition Tree Construction

Input: Rule subset $\mathcal{R}_v$, packet subset $\mathcal{P}_v$, fields D , depth h

Input: Max depth H , threshold B , model $\mathcal{M}$

Output: Node v

- 1: $D \leftarrow \text{REMOVEBYPASSFIELDS}(\mathcal{R}_v, D)$
 - 2: **if** SHOULDMAKETERMINAL($\mathcal{R}_v, D, h, H, B$) **then**
 - 3: **return** MAKETERMINAL($\mathcal{R}_v, \mathcal{P}_v, D, \mathcal{M}$)
 - 4: $(f^*, \mathcal{K}) \leftarrow \text{CHOOSEPLITFIELD}(\mathcal{R}_v, D)$
 - 5: **if** $f^* = \text{null}$ **then**
 - 6: **return** MAKETERMINAL($\mathcal{R}_v, \mathcal{P}_v, D, \mathcal{M}$)
 - 7: Create routing node v
 - 8: $v.\text{TYPE} \leftarrow \text{ROUTING}$
 - 9: $v.\text{SPLITFIELD} \leftarrow f^*$
 - 10: $D' \leftarrow D \setminus \{f^*\}$
 - 11: **for** each component $\kappa \in \mathcal{K}$ **do**
 - 12: $\mathcal{R}_\kappa \leftarrow \kappa.\text{RULES}$
 - 13: $I_\kappa \leftarrow \kappa.\text{RANGE}$
 - 14: $\mathcal{P}_\kappa \leftarrow \text{PARTITIONPACKETS}(\mathcal{P}_v, f^*, I_\kappa)$
 - 15: $u_\kappa \leftarrow \text{BUILDTREE}(\mathcal{R}_\kappa, \mathcal{P}_\kappa, D', h+1, H, B, \mathcal{M})$
 - 16: Append (I_κ, u_κ) to $v.\text{CHILDREN}$
 - 17: **return** v
-

Fields that are identical across all rules in $\mathcal{R}_v$ provide no discriminatory power and are removed. CHOOSEPLITFIELD returns a remaining field whose interval overlap graph yields

more than one connected component, together with those components; if no such field exists it returns null and the node becomes terminal. A node becomes terminal if the number of rules in the node get less than or equal to a predefined bin threshold B , no fields remain or maximum depth H is reached.

Algorithm 2 RemoveBypassFields

Input: Rule subset $\mathcal{R}_v$, field set D

Output: Reduced field set D'

```

1:  $D' \leftarrow D$ 
2: for each field  $f \in D$  do
3:   if ALLRULESIDENTICALONFIELD( $\mathcal{R}_v, f$ ) then
4:     Remove  $f$  from  $D'$ 
5: return  $D'$ 

```

Algorithm 3 ShouldMakeTerminal

Input: $\mathcal{R}_v, D, h, H, B$

```

1: if  $|\mathcal{R}_v| \leq B$  or  $D = \emptyset$  or  $h \geq H$  then
2:   return true
3: else
4:   return false

```

Algorithm 4 MakeTerminal

Input: $\mathcal{R}_v, \mathcal{P}_v, D$, model $\mathcal{M}$

```

1: Create node  $v$ 
2:  $v.TYPE \leftarrow \text{TERMINAL}$ 
3:  $s_v \leftarrow \text{EXTRACTFEATURES}(\mathcal{R}_v, \mathcal{P}_v, D)$ 
4:  $(S_j, \theta) \leftarrow \mathcal{M.SELECTSTRUCTURE}(s_v)$ 
5:  $v.STRUCTURENAME \leftarrow S_j$ 
6:  $v.CONFIG \leftarrow \theta$ 
7:  $v.STRUCTURE \leftarrow \text{BUILD}(S_j, \theta, \mathcal{R}_v)$ 
8: return  $v$ 

```

C. Packet Classification

Classification follows a single root-to-leaf path. The process is defined in Algorithm 5. The packet traverses from root of the composition tree on the basis of field and the split value defined in the intermediate, routing nodes to arrive at the relevant leaf node. The leaf node has information on which classifier to use. Thus, the packet is fed to the relevant classifier (NPtree, PartitionSort, CutSplit, etc.). Thus, classification will always have 100% accuracy since the classification techniques mentioned are exact classifiers.

D. Correctness

Correctness is independent of the learned model and follows from the composition structure.

Definition 1 (Routing Invariant). *A routing node ensures that any matching rule is contained in exactly one child and that packets are forwarded to that child.*

Theorem 1 (Compositional Exactness). *If all terminal nodes use exact classifiers and routing nodes satisfy the routing invariant, then classification returns the exact highest-priority matching rule.*

Proof. By induction on tree depth. Terminal nodes are exact by construction. At routing nodes, the invariant guarantees that all matching rules are preserved in the unique child traversed. The result follows recursively. $\square$

E. Summary

EnsembleClassifier decomposes packet classification into independent subproblems and assigns each an appropriate exact structure. Learning affects only structure selection, while correctness is guaranteed by construction.

IV. STRUCTURE SELECTION MODEL

The learned model $\mathcal{M}$ selects, for each terminal node v , the classifier structure $S \in \mathcal{S}$ that minimizes classification latency for the local rule subset $\mathcal{R}_v$.

Since partitioning is independent of learning, structure selection is a local decision problem: each terminal node is optimized independently based on its rule geometry and traffic characteristics. This decoupling simplifies learning and preserves correctness.

We formulate structure selection as supervised learning: given features describing $\mathcal{R}_v$, predict the structure that minimizes mean classification latency.

Training is performed by generating Classbench rulesets and applying the same recursive partitioning procedure used during classifier construction. The resulting terminal nodes are used as training instances. This aligns the training distribution with the deployment distribution, since the learned model operates only on terminal subsets at inference time.

A. Synthetic Training via Partition-Derived Terminal Subsets

Training data is generated by sampling Classbench synthetic rulesets and applying the recursive partitioning procedure used by EnsembleClassifier. Terminal nodes extracted from the resulting composition trees define the training instances.

Each sampled ruleset produces one or more terminal subsets through recursive partitioning. Because these subsets are generated by the same decomposition procedure used at deployment, they match the distribution of inputs seen by the learned selector during classifier construction.

Algorithm 5 Packet Classification

Input: Tree $\mathcal{C}$, packet P

```

1:  $v \leftarrow \text{root}$ 
2: while  $v$  is not terminal do
3:    $k \leftarrow v.SPLITFIELD$ 
4:    $v \leftarrow v.ROUTE(P[k])$ 
5: return  $v.STRUCTURE.QUERY(P)$ 

```

Algorithm 6 Synthetic Training via Partition-Derived Terminal Subsets

Input: Number of samples N , field count d , size range $[n_{\min}, n_{\max}]$, generation distribution $\mathcal{G}$, structure set $\mathcal{S}$, field set D , max depth H , threshold B

Output: Trained model $\mathcal{M}$

```

1:  $\mathcal{L} \leftarrow \emptyset$ 
2: for  $i = 1$  to  $N$  do
3:   Sample  $n \sim [n_{\min}, n_{\max}]$ 
4:    $\mathcal{R}_i \leftarrow \text{GENERATERANDOMRULESET}(n, d, \mathcal{G})$ 
5:    $\mathcal{C}_i \leftarrow \text{BUILDTREE}(\mathcal{R}_i, \emptyset, D, 0, H, B, \text{null})$ 
6:    $\mathcal{V}_i \leftarrow \text{EXTRACTTERMINALNODES}(\mathcal{C}_i)$ 
7:   for each terminal node  $v \in \mathcal{V}_i$  do
8:      $\mathcal{R}_v \leftarrow v.\text{RULES}$ 
9:      $\mathcal{P}_v \leftarrow \text{GENERATEPACKETS}(\mathcal{R}_v)$ 
10:     $s_v \leftarrow \text{EXTRACTFEATURES}(\mathcal{R}_v, \mathcal{P}_v)$ 
11:    for each  $S_j \in \mathcal{S}$  do
12:      Measure mean latency  $\bar{L}_{vj}$  via benchmarking
13:     $S_v^* \leftarrow \arg \min_{S_j} \bar{L}_{vj}$ 
14:    Add  $(s_v, S_v^*)$  to  $\mathcal{L}$ 
15: Train  $\mathcal{M}$  on  $\mathcal{L}$  using cross-entropy loss

```

B. Why Synthetic Training

Synthetic training provides three key advantages.

First, it enables full supervision: for each terminal subset, every candidate structure is evaluated, allowing direct comparison across the entire portfolio.

Second, it ensures coverage of diverse rule geometries by generating synthetic rulesets and extracting terminal subsets through partitioning, including cases that may be rare in real workloads.

Third, it aligns training with deployment: terminal subsets are produced using the same recursive partitioning procedure used during classifier construction, ensuring that the learned model is trained on the same distribution it encounters in practice.

Each terminal node is represented by feature vector s_v . It contains unique endpoints, mean overlap ratio, range entropy, prefix-alignable fraction and point-rule fraction. All features are normalized.

C. Learning Objective

For each instance (s_i, S_i^*) :

$$S^*(s_i) = \arg \min_{S_j \in \mathcal{S}} \bar{L}_{ij}$$

The model is trained with cross-entropy:

$$\mathcal{L} = \sum_i \ell(\hat{S}(s_i), S^*(s_i))$$

The objective directly optimizes classification latency, ensuring consistency with evaluation.

D. Model Architecture and Inference

$\mathcal{M}$ is a three-layer feedforward network with 128 hidden units in the first two layers and 64 units in the third layer. ReLU activations are used.

The output is a softmax over $\mathcal{S}$:

$$\hat{S}(s_v) = \arg \max_{S \in \mathcal{S}} P(S | s_v)$$

Inference takes microseconds and introduces negligible overhead. Structure parameters are obtained via a lookup table indexed by predicted structure and discretized feature bins. The trained model is fixed and used during tree construction.

The model is used only for structure selection at terminal nodes. It does not affect packet matching semantics, which remain exact. It is trained and evaluated on ClassBench-derived rulesets, using disjoint ruleset files for training and evaluation so that no terminal subset is seen during both phases.

V. EXPERIMENTAL EVALUATION

In this section we define the experimental framework used. This includes the rulesets used, packet traces, training of the classifier and the metrics used.

A. Implementation and Setup

EnsembleClassifier is implemented in C++ on top of prior implementations of NPtree-Split, NPtree-Cut [13], HyperCuts, HyperSplit, SmartSplit, CutSplit, CutTSS, and PartitionSort [12]. The selector is a three-layer neural network with 128 hidden units per layer except the last one which has 64 units.

Experiments are conducted on a 128 GB RAM, 32-core AMD Threadripper system.

Rulesets. We evaluate on ClassBench ACL, FW, and IPC rulesets (1K–64K rules). Each sized ruleset has 60 files with 5 configuration files each for ACL and FW, with IPC having 2 configuration files. Each configuration file is used with 5 different seeds in order to best simulate actual traffic conditions through the ruleset. The combined number of ruleset files used is 420.

Packet Traces. Each ruleset uses a construction trace $\mathcal{T}$ and a disjoint evaluation trace $\mathcal{V}$.

Training. Ruleset files of all sizes (1k - 64k) were considered for training of the MLP. We also ran an experiment with only using the 1k sized rulesets for training. This is because statistical analysis showed that the percentage of average of overlapping rules to the total number of rules is the same for all ruleset sizes, given the different seeds and configuration files being the same for all sizes.

For 1k rules unique feature vectors were generated as a result of creating the composition tree leaf nodes for each ruleset file. When training was done for all ruleset files, the total number of unique feature vectors the MLP was trained on were 70681.

Baselines. We compare against HyperSplit, NPtree, PartitionSort, HyperCuts, SmartSplit, and PTSS. The source code for these is from their respective GitHub repositories. The source code for these techniques was used as is, apart from

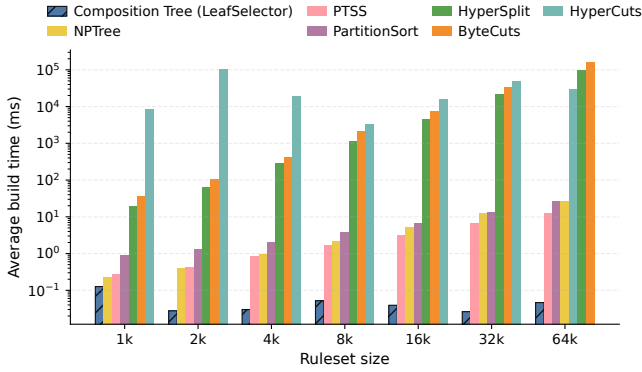


Fig. 2: Build time across ruleset sizes.

minor changes for reporting in order to comply with the common interface of EnsembleClassifier¹.

Configuration Fairness. Each baseline is evaluated under its best global configuration (sweeping standard parameter ranges). EnsembleClassifier selects configurations per terminal node using learned lookup tables.

Metrics. We report build time, classification time, and memory utilization, and compare it against others. These times are averaged over all ruleset size files, i.e. 1k, 2k, 4k upto 64k. Thus, each averaging is done over data gathered for runs on 60 ruleset files. The packet trace used is common for all classifiers in view of fairness.

B. Build Time

Figure 2 shows build time across different ruleset sizes for EnsembleClassifier, along with other baseline techniques. The build times are in log-scale.

Partitioning in EnsembleClassifier runs in $O(n \log n)$, and structure selection adds negligible overhead. On average, Ensemble Classifier results in 44.42% reduced build times for 1k rules, going on to 99.64% reduction in build times for 64k rules.

On average, Ensemble Classifier takes 98.65% less time to build than PriorityTuple when taking all rulesets into account. The performance difference grows as the ruleset size increases.

C. Classification Latency

Figure 3 shows mean classification latency. For 1k ruleset sizes, EnsembleClassifier reduces the mean classification time by 39.02%. For 64k rulesets it reduces the time by 63.35%. This increase in reduction is monotonic for ruleset sizes.

On average, EnsembleClassifier has 45.97% lower latency than the next best classifier, PartitionSort. Improvements increase with ruleset size due to greater heterogeneity.

D. Memory Footprint

Figure 4 shows memory usage across all ruleset sizes. Considering all ruleset sizes, memory used is reduced by 99.14% on average, compared to the next best technique in memory usage: PriorityTuple.

¹<https://github.com/asheshgaur/EnsembleClassifier>

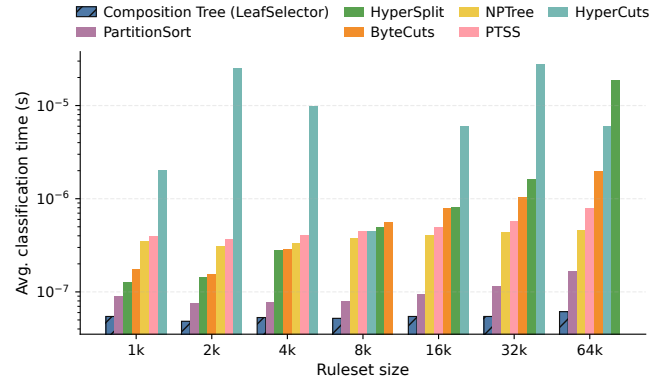


Fig. 3: Mean classification latency across ruleset sizes.

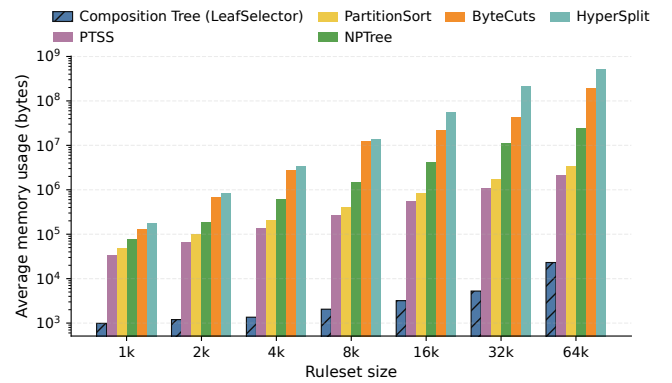


Fig. 4: Memory footprint across ruleset sizes.

Hash based techniques are the best in minimizing rule duplication, so PriorityTuple being the next best memory utilizer out of the portfolio of techniques in no surprise. EnsembleClassifier is better than PriorityTuple due to selecting lightweight structures for small subsets of rules. Also, the Composition Tree divides the ruleset into subsets of rules such that no rule replication occurs.

Table II shows the summary of the performance gains with EnsembleClassifier with respect to average classification latency (per packet), memory usage and build times. The first row for each metric shows performance data with training from all sized rulesets. The second row shows data with training from only 1k sized rulesets.

VI. CONCLUSION

Existing packet classifiers apply a single data structure to an entire ruleset, despite significant variation in rule geometry. We presented EnsembleClassifier, which addresses this by recursively partitioning rules into connected components using separable field dimensions, and selecting the best exact classifier for each component based on its geometry.

Partitioning is purely mechanical, while structure selection is guided by a learned model trained on synthetically generated rulesets with ground-truth performance labels. The resulting

TABLE I: Overall Performance Summary

Metric	Best Baseline	EnsembleClassifier	Improvement
Latency (s)	$9.98 \cdot 10^{-8}$	$5.39 \cdot 10^{-8}$	$1.85\times$
Memory (bytes)	616,991	5,320	99.14%
Build Time (ms)	3.654	0.0494	98.65%

TABLE II: Overall Performance Summary.

Metric	Best Baseline	EnsembleClassifier	Improvement
Query (s)	$9.98 \cdot 10^{-8}$ (same)	$5.39 \cdot 10^{-8}$ $6.4 \cdot 10^{-8}$	$1.85\times$ $1.55\times$
Memory (bytes)	616,991 (same)	5,320 6,784	99.14% 98.9%
Build Time (ms)	3.654 (same)	0.0494 0.236	98.65% 93.5%

classifier is provably exact, unlike neural approaches that approximate matches and may produce errors.

The partition tree introduces no rule replication—each rule appears in exactly one terminal component. Memory overhead arises only from the chosen classifier structures and is reflected in experimental measurements.

Results on ClassBench-derived rulesets (1K–64K) show that EnsembleClassifier outperforms any single fixed classifier in build time, latency, and memory. Overall, this confirms that per-component classifier selection yields a more efficient and scalable latency–memory–build tradeoff than any uniform approach.

We believe this pattern—learning that reorganizes a system while a structural invariant preserves its correctness—generalizes beyond packet classification to other domains where intelligent systems must operate under hard correctness constraints.

REFERENCES

- [1] P. Gupta and N. McKeown, “Classifying packets with hierarchical intelligent cuttings,” *IEEE Micro*, vol. 20, no. 1, pp. 34–41, 2000.
- [2] V. Srinivasan, S. Suri, and G. Varghese, “Packet classification using tuple space search,” *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 4, p. 135–146, aug 1999. [Online]. Available: <https://doi.org/10.1145/316194.316216>
- [3] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, “Effcuts: Optimizing packet classification for memory and throughput,” in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM ’10. New York, NY, USA: Association for Computing Machinery, 2010, p. 207–218. [Online]. Available: <https://doi.org/10.1145/1851182.1851208>
- [4] W. Li, X. Li, H. Li, and G. Xie, “Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification,” in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 2645–2653.
- [5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, mar 2008. [Online]. Available: <https://doi.org/10.1145/1355734.1355746>
- [6] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, “The design and implementation of open vswitch,” in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’15. USA: USENIX Association, 2015, p. 117–130.
- [7] S. Singh, F. Baboescu, G. Varghese, and J. Wang, “Packet classification using multidimensional cutting,” in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 213–224. [Online]. Available: <https://doi.org/10.1145/863955.863980>
- [8] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, “Packet classification algorithms: From theory to practice,” in *IEEE INFOCOM 2009*, 2009, pp. 648–656.
- [9] P. He, G. Xie, K. Salamatian, and L. Mathy, “Meta-algorithms for software-based packet classification,” in *2014 IEEE 22nd International Conference on Network Protocols*, 2014, pp. 308–319.
- [10] W. Li and X. Li, “Hybridcuts: A scheme combining decomposition and cutting for packet classification,” in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. IEEE, 2013, pp. 41–48.
- [11] W. Li, T. Yang, O. Rottenstreich, X. Li, G. Xie, H. Li, B. Vamanan, D. Li, and H. Lin, “Tuple space assisted packet classification with high performance on both search and update,” *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1555–1569, 2020.
- [12] S. Yingchareonthawornchai, J. Daly, A. X. Liu, and E. Torng, “A sorted-partitioning approach to fast and scalable dynamic packet classification,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1907–1920, 2018.
- [13] A. Gaur, S. Radhakrishnan, A. Narasimhan, and M. Atiquzzaman, “On packet classification with multidimensional range trees,” in *ICC 2024 - IEEE International Conference on Communications*, 2024, pp. 1084–1089.
- [14] A. Narasimhan, S. Radhakrishnan, M. Atiquzzaman, and C. R. Subramanian, “High-speed packet classification: A case for approximate sorting,” in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 5765–5770.
- [15] A. Rashelbach, O. Rottenstreich, and M. Silberstein, “A computational approach to packet classification,” *IEEE/ACM Transactions on Networking*, vol. 30, no. 3, pp. 1073–1087, 2022.
- [16] E. Bakopoulou, B. Tillman, and A. Markopoulou, “Fedpacket: A federated learning approach to mobile packet classification,” *IEEE Transactions on Mobile Computing*, vol. 21, no. 10, pp. 3609–3628, 2022.
- [17] Z. Li, T. Wang, J. Liu, Y. Zhang, T. Ma, and K. Liu, “Tupelick: A high stability packet classification based on neural network,” in *2024 IEEE 25th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2024, pp. 377–382.
- [18] P. He, G. Xie, K. Salamatian, and L. Mathy, “Meta-algorithms for software-based packet classification,” in *2014 IEEE 22nd International Conference on Network Protocols*, 2014, pp. 308–319.
- [19] S. Rahul, “Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in r_3 ,” in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA ’15. USA: Society for Industrial and Applied Mathematics, 2015, p. 200–211.