SpliDT: Partitioned Decision Trees for Scalable Stateful Inference at Line Rate

Paper #86

12 Pages Body, 14 Pages Total

ABSTRACT

Machine learning is increasingly being deployed in programmable network switches to enable real-time traffic analysis and security monitoring. Decision trees (DTs) offer a powerful approach for these tasks due to their interpretability and transparency. However, existing DT implementations in switches face a critical limitation: they require collecting all features before making a decision. This constraint forces models to use a small, fixed set of features per flow, limiting accuracy and scalability.

This paper introduces SpliDT, a scalable framework that removes this feature constraint through a partitioned inference architecture. Instead of requiring the same fixed features for all decisions, SpliDT assigns different sets of features to different parts of the tree and dynamically selects them as needed. To efficiently manage resources, SpliDT leverages recirculation to reuse registers and match keys at line rate. These capabilities are enabled by two core innovations: (1) a Sub-Tree ID (SID) tracking mechanism for managing inference across partitions and (2) a custom training framework using HyperMapper and Bayesian Optimization to optimize decision tree structure and feature allocation. Our evaluation shows that SpliDT achieves higher accuracy while accommodating up to 5× more stateful features and scaling to millions of flows, significantly outperforming state-of-the-art approaches like NetBeacon and Leo. Despite this increased capacity, SpliDT maintains low recirculation overhead (≤ 50 Mbps) and low time-to-detection (TTD), demonstrating that ML models can operate efficiently within the constraints of programmable switches.

1 INTRODUCTION

Machine Learning (ML) is rapidly becoming a cornerstone of modern networking, driving increasingly sophisticated applications including DDoS detection (LUCID [14], Flowlens [2]), congestion control [13, 21, 32, 50, 54], and VBR video streaming [34, 53]. These applications demand real-time, high-throughput inference [46] to handle the ever-growing scale and complexity of network traffic [3, 8–12, 42].

Programmable switches [25, 26], with their ability to process packets at line rate, have emerged as a powerful platform for deploying ML models directly in the data plane [28, 46, 47, 51, 52, 56, 58]. By offloading inference tasks to network hardware, these switches eliminate the need for control-plane intervention, enabling low-latency and high-performance decision-making [28, 46, 58].

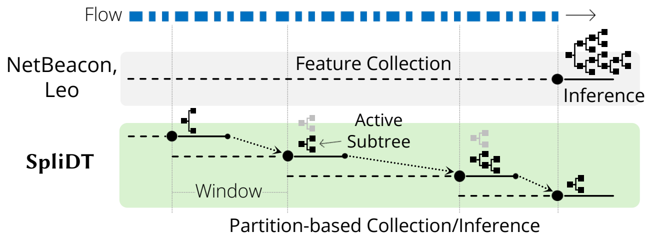

Figure 1: Comparison of in-network decision tree classification approaches. State-of-the-art methods (top) perform one-shot inference by collecting features over the entire flow duration. In contrast, SpliDT (bottom) collects features and performs inference incrementally across partitions using windows of packets—significantly scaling the number of stateful features with increasing model sizes, while achieving higher F1 scores at line rate.

A major focus of recent work has been on mapping decision tree (DT) models to programmable switches [7, 28, 51, 52, 58], mainly because of these models' interpretability and natural alignment with the reconfigurable match-action table (RMT) architecture [6, 25, 26]. While systems such as IIsy [52], NetBeacon [58], and Leo [28] have demonstrated the feasibility of deploying DTs in the data plane—to be able to operate within the stringent resource constraints of programmable switches—they have mainly addressed the challenge of rule explosion and concentrated on optimizing model representations (e.g., pruning DTs [28] and compressing match-action rules [52, 58]).

However, despite these advancements, the critical aspect of feature engineering (i.e., selecting and computing input features) has not featured prominently in this context, except for its limitations due to existing resource constraints. Indeed, while current approaches such as NetBeacon [58] and Leo [28] strictly limit the number of stateful features to a small, fixed set (e.g., the top-*k* most important features), other systems such as IIsy [52] and Mousika [51] prohibit using stateful features altogether. As we show in §2, these strategies result in poor model performance (e.g., reduced F1 scores) and prevent the deployed DT models from adequately capturing the complex patterns of realistic traffic [28, 52, 58].

More importantly, the general lack of focus on feature engineering stems from two common assumptions. First, all selected features are expected to be computed upfront before DT traversal begins Figure 1 (top), a constraint largely imposed by the pipelined nature of programmable switches. Second, DT execution is assumed to be a one-shot process, restricting opportunities for resource reuse across decision stages. As a result, existing approaches treat reducing stateful features as the only viable solution to meet resource constraints, forcing a trade-off between model accuracy and scalability (i.e., supporting more flows) [28, 58].

In this paper, we challenge these assumptions by introducing SpliDT, a system that enables efficient DT execution in the data plane while addressing feature engineering constraints. SpliDT is built on two key insights. First, *feature computation can be deferred*: Unlike traditional approaches that compute all features upfront, SpliDT introduces tree partitioning, which enables incremental and on-demand feature selection as DT traversal progresses Figure 1 (bottom). Tree partitioning systematically divides a DT into consecutive subtrees, ensuring that only features relevant to a specific subtree are selected and computed at any given time. Second, *resource reuse via resubmission*: By leveraging packet recirculation, SpliDT dynamically repurposes switch resources (i.e., registers and match keys) across tree partitions, enabling resource-efficient execution without sacrificing line rate. A key trade-off in this design is that deferred feature computation results in features being extracted at different time intervals, potentially affecting model accuracy and time-to-detection (TTD) for time-sensitive tasks. However, as we show in §5, this trade-off is insignificant and does not affect model performance or TTD while allowing for scalable and resource-efficient deployment of DTs in the data plane.

Building on these insights, SpliDT significantly *increases the total number of stateful features* that a DT can utilize over its entire execution. Unlike prior approaches that apply a fixed top-k feature set across the entire DT, SpliDT dynamically assigns each subtree—resulting from tree partitioning—its own relevant feature set, allowing features to vary between subtrees. The resulting subtrees are executed sequentially via packet recirculation, with SpliDT dynamically reusing stateful registers and match keys at each stage of tree traversal for the currently active subtree Figure 1 (bottom). We demonstrate in §5 that SpliDT supports an order of magnitude more stateful features (i.e.the total number of unique features across all subtrees) than state-of-the-art approaches [28, 58], while maintaining efficient execution in the data plane.

However, to achieve these benefits, SpliDT must overcome two key challenges: (1) determining how to efficiently compute and provide the necessary features for each subtree at runtime during inference, and (2) designing an effective partitioning strategy during training that optimally balances model accuracy and resource efficiency to support the maximum number of flows.

To address the first challenge, SpliDT processes a flow in windows of packets for each partition. Ideally, every partition would have access to the entire flow during inference; however, since switches monitor traffic at line-rate without buffering, this is not feasible [6, 25, 26]. Instead, SpliDT segments each flow into uniform windows, allowing each partition to observe a portion of the flow during inference, Figure 1 (bottom). Additionally, modern datacenter transport protocols (e.g., Homa [36] and NDP [22]) embed flow size information in packet headers, which switches can parse to identify window boundaries, use to stop feature collection or trigger active subtree selection, and leverage to transition to the next partition (via recirculation).

To tackle the second challenge, SpliDT employs an iterative design search methodology integrated with a custom training framework to fine-tune DT configurations, including partitioning strategies and resource allocation policies, for specific use cases (and datasets). The objective is to optimize the trade-off between the number of supported flows and model accuracy, identifying configurations that lie on the Pareto frontier. Using Bayesian Optimization (e.g., HyperMapper [37]), SpliDT systematically explores the design space to identify the most effective hyperparameters, including the maximum number of features (k) per subtree, the size and number of partitions, and the overall tree depth. Intuitively, smaller values of k allow for more flows to be supported,[1] while deeper trees improve model accuracy. Additionally, increasing the number of partitions expands the total set of unique features available across the tree but reduces the number of packets each partition can observe, limiting the temporal window for feature computation. SpliDT efficiently explores this trade-off space to derive a Pareto-optimal model that balances accuracy and scalability within the given resource budget.

Our results demonstrate that SpliDT establishes a new state-of-the-art in deploying DT models in the data plane. It consistently outperforms NetBeacon [58] and Leo [28], achieving higher accuracy to number of flows Pareto frontier irrespective of the number of supported flows (see Table 3), while supporting 5× more stateful (offloadable) features—enabling richer in-network inference—and maintaining a low recirculation overhead of just 50 Mbps in the worst case.

In summary, we make the following contributions:

- To enable window-based inference and register reuse, SpliDT introduces a Sub-Tree ID (SID) state in the data plane, which tracks the active subtree for a flow–functionally similar to a program counter in CPUs. The

[1]In Tofino1 [25], $k = 4$ supports up to 100,000 flows, which decreases to 65,000 with $k = 6$, and so on [28, 58].

System	F1 Score	#Flows	Stateful Features
IIsy [52]	low	1M	none
Planter [56]	low	1M	none
NetBeacon [58]	mod	200K	top-$k \leq 6$
Leo [28]	low	500K	top-$k \leq 7$
SpliDT	high	1M	all

Table 1: Comparison of SpliDT with existing systems—sustaining higher F1 score while covering all stateful features at line rate.

SID dynamically guides both feature collection and TCAM rule selection for inference, allowing registers to be repurposed as flows traverse through subtrees.

- To train partitioned SpliDT trees, we develop a custom window-based training pipeline that leverages HyperMapper [37] for Bayesian optimization to explore the model configuration space. This pipeline systematically optimizes tree depth, features per sub-tree (k), and the number of partitions to balance model accuracy and switch resource constraints. It trains models, evaluates them against hardware limitations, and selects configurations that achieve a high accuracy while remaining deployable on programmable switches.
- Our evaluations show that SpliDT can accommodate up to 1M flows with high-precision 32-bit features and up to 4M flows with reduced 8-bit precision, demonstrating its scalability under varying resource constraints. SpliDT consistently achieves higher accuracy across all evaluated datasets, outperforming baseline systems on the Pareto frontier, while maintaining similarly low end-to-end time-to-detection (TTD) as existing approaches.

To facilitate reproducibility and further research in this area, we plan to publicly release the complete artifact, including training scripts, models, and evaluation datasets.

This work does not raise any ethical issues.

2 BACKGROUND & MOTIVATION

We first review prior work on in-network classification systems that use decision tree (DT) models and highlight the need for more stateful features for DT-based inference (§2.1). Next, we revisit the anatomy of DTs to derive domain-specific insights (§2.2) that guide us in addressing the challenges of scaling DTs on modern programmable switches (§2.3).

2.1 The Need for More Stateful Features

As network traffic scales to multi-Tbps rates, existing approaches, such as IIsy [52] and Planter [56], aim to support real-time inference by mapping DTs onto match-action tables (MATs) while relying exclusively on stateless, per-packet features. These methods optimize DT representation to fit within the constraints of available switch resources (i.e., MATs) but lack flow-level context, limiting their classification accuracy

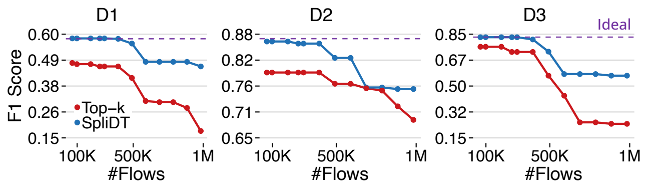

Figure 2: SpliDT and top-$k \leq 7$ model vs. the ideal scenario with unlimited resources. SpliDT, with access to all features, achieves higher F1 score than top-k for the datasets, D1–3 (details in §5). The per-packet models peak at 0.41, 0.56, and 0.59, respectively (not shown).

and adaptability [51, 52, 56, 58]. While they efficiently handle large flow volumes in the data plane (Table 1), their reliance on per-packet features significantly reduces accuracy, yielding F1 scores nearly 2× lower than models with full features access (Figure 2).

More recent work, such as NetBeacon [58] and Leo [28], improves classification accuracy by incorporating stateful features (i.e., top-k), allowing decision DT models to leverage flow-level context, which provides richer insights than per-packet features alone. While this approach enhances accuracy, it introduces significant memory constraints in programmable switches [6, 25, 26].

First, stateful features must be stored in registers, which share limited space with match-action tables (MATs) within each pipeline stage, creating a trade-off between feature storage and model complexity. For example, allocating four registers per flow consumes an entire switch stage for 65K flows, preventing that stage from being used for model execution. Increasing the number of registers or supporting more flows further reduces available MAT stages, limiting DT depth and restricting feature selection. Second, expanding stateful features increases match key sizes, inflating the number of table entries and exacerbating TCAM memory usage, which complicates mapping DT to match-action tables in the switch [25, 28, 58].

As a result, prior work has been constrained to a maximum of 200K flow rules and top-$k \leq 6$ stateful features—values dictated by available switch memory and the need to balance register allocation with model depth—offering only moderate accuracy (Table 1). Beyond this threshold, performance deteriorates as DT depth is constrained, reducing classification accuracy, while increasing match key sizes strain TCAM capacity, limiting scalability. The combination of these trade-offs underscores the fundamental challenge of integrating stateful features into DT-based inference while maintaining scalability within the strict resource constraints of programmable switches.

Observation: The constraints of prior DT-based systems are often perceived as intrinsic to programmable switch architectures, but are they truly fundamental? We argue that these

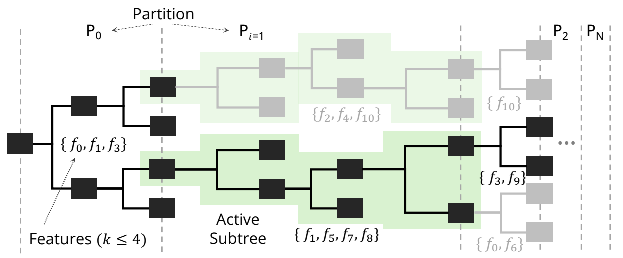

Figure 3: Domain-specific properties of DTs: Partitions (P_i) comprising multiple subtrees, each with its own subset of features (k). During traversal, the active subtree is selected within each partition.

Data	Feature Density (%) / Partition	Feature Density (%) / Subtree	Recirc. Bandwidth (Mbps) E1: Webserver	Recirc. Bandwidth (Mbps) E2: Hadoop
D1	47.15 ± 38.44	6.15 ± 2.95	2.93 ± 2.44	5.99 ± 3.51
D2	53.49 ± 44.19	7.28 ± 2.72	6.01 ± 4.01	12.32 ± 5.76
D3	53.95 ± 43.42	6.08 ± 3.37	3.58 ± 3.21	7.33 ± 4.62

Table 2: Feature density (%) across partitions and subtrees in a DT, and maximum recirculation bandwidth (Mbps) when processing the datasets (D1–3) for two datacenter environments (E1–2), details in §5

limitations do not stem from hardware constraints but from ingrained assumptions about how DTs should be processed. Conventional approaches assume that all stateful features must be collected before inference begins, necessitating upfront register allocation. Furthermore, these approaches treat DT execution as a single-pass, feed-forward process, confining computation to the spatially available pipeline resources.

This paper challenges the prevailing belief that feature richness and scalability must always be in conflict. We demonstrate that by decoupling stateful feature selection from DT execution, both can scale independently. Unlike prior work (e.g., NetBeacon [58] and Leo [28]), which sacrifices feature expressiveness for scalability, SpliDT dynamically selects and reuses stateful features across inference steps, efficiently managing available hardware resources without restricting model complexity. Compared to traditional top-k systems, SpliDT achieves a significantly improved Pareto frontier, simultaneously enhancing F1 scores and flow rule capacity (Figure 2). These results challenge the notion that switch-imposed constraints inherently limit DT scalability, showing instead that the primary bottleneck arises from rigid execution models that preallocate features and enforce single-pass processing.

2.2 Domain-Specific Properties of DTs

DT inference begins at the root node, where a decision is made based on selected features to determine the next node to visit. This process continues at each level, using different features at each step until a leaf node is reached. Instead of processing the tree level by level, we can group consecutive levels into *partitions* (Figure 3) and focus on the subtrees in each partition. With this approach, inference progresses one partition at a time, where the decisions resulting from traversing the *active* subtree in one partition determine which subtree to traverse in the next partition. This allows for more efficient traversal, selecting only the next subtree based on relevant features and their conditions rather than evaluating entire levels sequentially.

This subtree-by-subtree execution enables features to be collected incrementally and on-demand for the *active* subtree in each partition (Figure 1, bottom). Unlike traditional approaches that require gathering all features upfront, this method loads only the features needed for the current subtree. With just k available feature slots, we can dynamically allocate and process only the relevant features at each step, avoiding the restrictive top-k selection enforced by existing systems [28, 58]. This approach maximizes feature utilization without discarding valuable contextual information across the entire tree.

However, two key conditions must hold for this approach to be effective: (a) *feature density across subtrees*: each subtree within a partition must use at most k features out of the total N available features. Unlike traditional top-k approaches that select a fixed set of k features for the entire tree, here k represents the number of feature slots available at any given step, which can be dynamically reassigned to different features as the inference progresses. If even a single subtree requires more than k features, the benefits of incremental feature computation are lost, as more than k features would need to be allocated upfront; (b) *incremental computation architecture*: the system must support subtree-by-subtree traversal, dynamically collecting and computing features while reusing the same k feature slots at each step. This ensures efficient execution without requiring all features to be preloaded at once, overcoming the limitations of prior top-k selection methods that discard all but the most globally important features across the entire tree.

To examine the feasibility of the first condition, we studied feature usage across subtrees in multiple datasets (Table 2). In the considered datasets (D1–3),[2] we found that at most only 10% of features were required in any given subtree. For instance, in dataset D1, where $N = 41$, subtrees required on average only 3.73 features. Note that this 11× reduction in storage requirement makes it possible to execute the entire tree using only k registers (e.g., 4 for D1) and avoids the need to impose a strict limit of k on the overall features.

For the second condition, we demonstrate in the next section (§2.3) how modern programmable switches, with support for packet recirculation [25, 26], can be reimagined as

[2]The remaining datasets, D4–7, show similar trends, see §5.

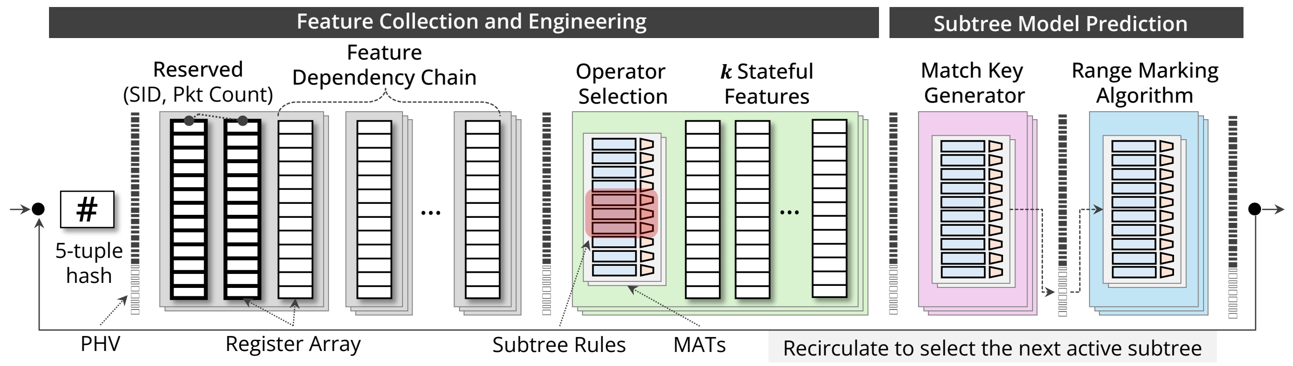

Figure 4: SpliDT's partitioned inference architecture, processing flow windows in two phases: (1) Feature Collection and Engineering (left) and (2) Subtree Model Prediction (right)—leveraging resource reuse (i.e., registers and match keys) for efficient execution within each DT partition.

time-shared machines, enabling efficient reuse of resources (e.g., registers) across partitions within the data plane.

2.3 Switch as a Time-Shared Machine

Programmable switches are traditionally viewed as spatial architectures with fixed resource constraints, where exceeding available resources leads to failures during program compilation [5, 6, 25]. However, we argue that because this perspective focuses mainly on the static aspects of both programmable switches and spatial architectures, it is overly restrictive and in need of being revisited.

For example, modern switches support packet recirculation, with bandwidths reaching 100 Gbps (Tofino1) [25], without impacting line rate. This capability adds an important dynamic element to the traditional view. In effect, it enables temporal execution, allowing different stages of a program (e.g., in P4 [5]) to activate across multiple recirculations in a time-shared manner. By leveraging this mechanism, the state can be distributed over time, facilitating the reuse of limited resources (e.g., registers and match keys in MATs).

By carefully restructuring DTs (expressed in P4), we can exploit this dynamic capability to scale DT-based inference beyond the physical limits of available resources—akin to how CPUs abstract resource constraints by reusing registers over time. Table 2 shows that for the evaluated datasets (D1–3),[2] the maximum recirculation path usage is within 20 Mbps for the two datacenter environments E1–2, significantly lower than the available bandwidth (i.e., 100 Gbps).

Next, we show how SpliDT leverages the domain-specific properties of DTs and reuse of switch resources via recirculation to optimize the Pareto frontier (F1 score vs. number of flow rules) while supporting all stateful features at line rate.

3 DESIGN OF SPLIDT

We now present the SpliDT design; its *partitioned inference architecture* for efficient DT execution in the data plane (§3.1) and the *custom training framework*, for effective model configuration through design search (§3.2).

3.1 Partitioned Inference Architecture

As illustrated in Figure 4, SpliDT's partitioned inference architecture operates in two phases: (1) Feature Collection and Engineering and (2) Subtree Model Prediction—iteratively processing portions (windows) of flows within each DT partition via resource reuse (i.e., registers and match keys).

3.1.1 Feature Collection and Engineering. SpliDT maintains three distinct register-array sets to manage stateful feature collection and subtree selection for prediction, as shown in Figure 4. These registers include: (1) reserved state registers for tracking metadata such as the subtree ID (SID), (2) registers for computing intermediate and dependent states (e.g., timestamps for inter-arrival time (IAT) calculations), and (3) k registers for storing stateful features specific to the active subtree within the current DT partition.

Upon packet arrival, SpliDT hashes its 5-tuple using CRC32 [23] to determine the register index corresponding to the flow. First, it retrieves the subtree ID (SID) from the reserved register array and updates the packet count in the second register array. Next, depending on the use case (and dataset), some DT models require intermediate values to compute stateful features before prediction. For instance, IAT computation requires storing the previous packet's timestamp to calculate the inter-packet gap.

INFO: *Reserved and dependency chain registers can significantly limit the number of features per subtree (k) as they must scale alongside the k features to support the same number of flows. This contention for register space creates a tradeoff that must be carefully managed to balance feature capacity and flow scalability in SpliDT, §3.2.*

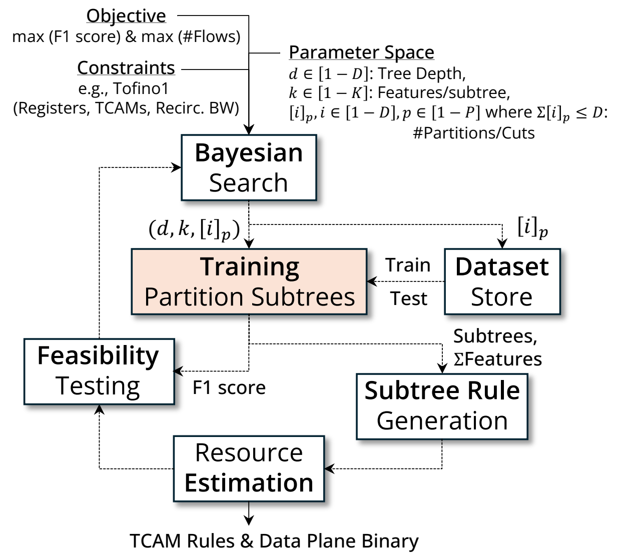
Objective
max (F1 score) & max (#Flows)
Constraints
e.g., Tofino1
(Registers, TCAMs, Recirc. BW)
Parameter Space
$d \in [1-D]$: Tree Depth,
$k \in [1-K]$: Features/subtree,
$[i]_p, i \in [1-D], p \in [1-P]$ where $\Sigma[i]_p \leq D$:
#Partitions/Cuts
Bayesian Search
$(d, k, [i]_p)$
$[i]_p$
Training Partition Subtrees
Train
Test
Dataset Store
Feasibility Testing
F1 score
Subtrees, ΣFeatures
Subtree Rule Generation
Resource Estimation
TCAM Rules & Data Plane Binary

Figure 5: Workflow of SpliDT's Model Design Search.

To support such computations, SpliDT implements a dependency chain, a sequence of register arrays distributed across multiple pipeline stages to enable hierarchical computation. Since programmable switches [6, 25, 26] cannot process dependent data within a single stage, computation must be spread across multiple stages. In our evaluations, the deepest observed dependency chain was four stages—a depth well within the capabilities of modern programmable switches (like Tofino1 [25]).

Operator Selection. Since each subtree may require a different operation to compute its stateful features, SpliDT dynamically updates the operation applied to each feature in every flow window. To achieve this, SpliDT utilizes match-action tables (MATs) for each stateful feature, acting as selectors to apply the appropriate operation on demand.

At compile time (§3.2.1), SpliDT populates these tables with rules that define which operator to apply for each subtree. The MATs match on the subtree ID (SID) and select the corresponding action to perform the necessary computation. Modern programmable switches support the parallel execution of multiple MATs within a stage. For instance, Tofino1 supports up to 16 MATs with 750 entries each, which is well within the requirements of our design—SpliDT utilizes only six MATs to support $k = 6$ stateful features in our evaluations (§5), with each table containing at most 200 entries.

Lastly, to prevent continuous updates of features on every arriving packet and to identify window boundaries, SpliDT incorporates additional match fields in the MATs. For instance, to update a stateful feature only on SYN packets, the MATs can include TCP flags as a match condition, ensuring the feature update is triggered only when a SYN is received.

3.1.2 Subtree Model Prediction. The prediction phase (Figure 4) is primarily composed of match-action tables (MATs). The first set of MATs constructs the match key for the DT model. Similar to the Operator Selection MATs, SpliDT maintains separate MATs for each feature, using precomputed rules to determine which stateful feature (register) should be used as a key. This key consists of dedicated metadata headers, which the DT model uses for classification.

Classification is performed using the Range Marking Algorithm [58], which translates a DT into two MAT-based rule tables. The first set of feature-specific MATs matches incoming metadata headers representing feature values and encodes them into unique bit strings. The second MAT—a single table containing DT model rules—uses these encoded feature values, along with the subtree ID (SID), as the lookup key to perform classification.

If a flow reaches its final partition (subtree) or an early exit leaf node, the classification label is sent as a digest to the controller as the final decision. Otherwise, the next SID is recirculated via the resubmission channel [25], updating the SID register with the next subtree ID and resetting the registers for the dependency chain and k stateful features in preparation for the next flow window.

3.2 Custom Search/Training Framework

In the following section, we describe the design search framework to generate optimal SpliDT DTs (§3.2.1) followed by the partitioned tree training algorithm (§3.2.2).

3.2.1 SpliDT Design Search. Figure 5 shows the overall workflow of SpliDT framework. The goal is to deliver a Pareto frontier for the SpliDT partitioned trees. Given the set of optimization objectives, input parameter space, and hardware/performance constraints, a Bayesian Optimization (BO) search phase begins that iteratively suggests model parameters to evaluate. We query the preprocessed window-based training/test dataset corresponding to the suggested number of partitions from a DB and our custom partitioned DT training algorithm trains the model with these parameters and tests it for its F1 score. We then generate its TCAM entries, calculate the hardware resource utilization, the number of flows supported, and feasibility of running this model on the switch at line rate. The F1 score, number of flows, and feasibility metric are fed back into the BO search to trigger the next iteration and this process repeats for a preset number of BO search iterations.

In the next sections, we elaborate on each individual aspect of this workflow, starting with the inputs to the BO search.

Parameter Space, Objectives, and Constraints. The search space consists of three parameters: one variable each for maximum tree depth (integer D: range 1–50) and features per subtree (integer k: range 1–4), and a list of partition sizes (integer i_p: range 1–50) whose length is equal to number of partitions and sum is equal to tree depth D. The objectives include the model's test F1 score on the and number of flows supported in the switch. The constraints include performance (resubmission bandwidth) and hardware resources (switch stages, TCAM blocks, and register space). Together, this parameter set, along with the dataset description, is fed as the input to the BO search.

Bayesian Search. Provided the input search space and objectives, the framework searches for a Pareto frontier for SpliDT DTs for the given dataset. Bayesian Optimization (BO) is a black-box search technique [49] designed for optimizing expensive-to-evaluate functions. It builds a probabilistic surrogate model, typically a Gaussian Process (GP) or Random Forest, to approximate the objective function and uses an acquisition function to determine the most promising points to sample next. By balancing exploration (searching new areas) and exploitation (refining known good areas), BO efficiently navigates complex, high-dimensional search spaces with minimal evaluations. This makes it ideal for applications like hyperparameter tuning, where each function evaluation (e.g., training a partitioned decision tree) is costly.

The BO search iteratively suggests parameters for multiple partitioned DTs to evaluate in parallel. Each set of parameters is used to train a partitioned DT using our custom training algorithm (§3.2) and its performance is evaluated on a test dataset. The model resource usage, feasibility and accommodated number of flows are evaluated in the following stages, which feedback to the BO search to continue to the next search iteration for a preset number of iterations.

Subtree Rule Generation. Given the trained partitioned DT (§3.2), we generate the TCAM rules required to represent this model in the data plane. We use the Range Marking algorithm [58] to generate TCAM entries for the partitioned decision trees. This algorithm encodes decision tree rules into TCAM efficiently by mapping feature value ranges to compact ternary bit strings. It divides each feature's range into non-overlapping ranges and generates unique *range marks* (bit strings), and ensures merged *associative ranges* retain distinct representations. For each feature, its thresholds from the trained partitioned DT are converted into ternary matches and the range marks are output as actions. These TCAM entries are installed into the feature tables to output range marks as the match key for the model table. Another set of TCAM entries are generated for the model itself and installed into the model table that matches on these feature range marks to determine the next subtree ID or the predicted class (at the last partition). This representation efficiently encodes each leaf into one TCAM rule for the model table, avoiding the rule explosion problem.

These feature and model entries are generated for each subtree in the partitioned DT and installed into the tables in

Algorithm 1 SpliDT partitioned DT training algorithm.

```
 1: procedure TrainPartitionedDT($dataset, depths, partition, k$)
 2:     /* k is the features per subtree */
 3:     if $level \geq$ len($depths$) then
 4:         return
 5:     $depth \leftarrow depths[partition]$
 6:     /* train one subtree at this partition */
 7:     $tree \leftarrow$ TrainSubTree($dataset[partition], depth, k$)
 8:     /* get subset of data samples for each leaf node */
 9:     $leaf\_subsets \leftarrow$ PartitionSamplesByLeaves($tree, dataset$)
10:     /* train subtrees for the next partition */
11:     for all ($leaf, subset$) $\in leaf\_subsets$ do
12:         if len($subset$) $> 0$ then
13:             TrainRecursiveTree($subset, depths, partition + 1$)
```

Figure 6: SpliDT partitioned DT training algorithm.

the data plane inference pipeline and include an exact match on the subtree ID (SID) to select the correct subtree for each partition and flow.

Feasibility Testing. For the generated TCAM entries, we calculate the number of TCAM blocks required for feature and model tables. The remaining stages (if any) are used for the bookkeeping and feature registers which determines the number of flows that can be supported with this model. Recirculated traffic volume is estimated using real-world datacenter environments (§5), the number of partitions (number of recirculations), and the number of flows which can consecutively recirculate packets. If the switch can accommodate flows and recirculation is within available limits, the design is considered feasible, otherwise infeasible.

The test F1 score, number of flows, and feasiblity are fed back into the BO search with search moving to the next iteration, at each step refining the suggested model parameters.

3.2.2 SpliDT Custom Training.

Figure 6 shows the algorithm to train our partitioned DTs. Given an overall tree depth, partition sizes, and features per subtree, we start by training the single tree at the first partition using all the datasamples corresponding to the first window. Then, for each of its leaf nodes, we determine the dataset samples that traversed to that leaf node and use only that subset of samples (and corresponding window to each partition) to selectively train the corresponding subtree. Leaf nodes that don't reach the depth of the tree do not spawn any more subtrees in their following partitions. This algorithm recursively trains the partitioned DT for window-based inference, specializing each subtree for that specific data distribution that reaches it during training.

4 IMPLEMENTATION

We implement the SpliDT framework in Python (v3.10.13), allowing seamless integration with widely used libraries such

Dataset	Description	Classes
D1: CIC-IoMT2024	A cybersecurity dataset [11] with Internet of Medical Things (IoMT) traffic for intrusion detection in healthcare.	19
D2: CIC-IoT2023-a	A simplified version of the CIC-IoT-2023 dataset [10], categorized into four primary classes of IoT traffic.	4
D3: ISCX-VPN2016	A dataset containing VPN and non-VPN traffic [12] for evaluating VPN detection and privacy-related analyses.	13
D4: CampusTraffic	UCSB campus dataset [20] containing various application types, including web, cloud, social, and, streaming traffic.	11
D5: CIC-IoT2023-b	A comprehensive IoT dataset [10] containing multi-class network traffic data for evaluating IoT security threats.	32
D6: CIC-IDS2017	A network intrusion detection dataset [8] for various attack scenarios, including DoS, DDoS, and brute force.	10
D7: CIC-IDS2018	An anomaly detection dataset [9] capturing network traffic for diverse attacks and benign activities.	10

Table 3: Real-world network security datasets used for evaluation.

as Pandas [39], Scikit-Learn [40], and TensorFlow [48]. At the core of SpliDT's optimization process is HyperMapper [37] (commit:3dfa8a7), which uses Bayesian Optimization (BO) to efficiently search for the best model configurations. Unlike other BO frameworks such as OpenTuner [1], HyperOpt [4], and GPflow Opt [30], HyperMapper supports multi-objective optimization, diverse parameter types (real, integer, categorical), and feasibility testing. These features allow us to optimize SpliDT models for multiple goals simultaneously, such as maximizing F1 score and flow capacity. The feasibility testing feature also helps eliminate tree configurations that exceed available switch resources (e.g., TCAM limits) or introduce excessive resubmission traffic. The BO experiments are configured via YAML, specifying datasets, parameter space, objectives, and switch constraints.

To train and test SpliDT's partitioned trees, we use the DecisionTreeClassifier class from Scikit-Learn [40] and recursively generate sub-trees for all partitions. We employ the Range-Marking algorithm, as described in NetBeacon [58], to generate TCAM rules. We parallelize evaluations using Python's ProcessPoolExecutor from the concurrent.futures module to speed up the BO search. For data plane inference, we implement SpliDT in P4 [5] and compile it using BF-SDE (v9.11.0) for the Tofino1 [25] switch. Finally, to install TCAM rules into the switch, we use bfrt_grpc client API in Python.

In total, the SpliDT framework consists of 4,700 lines of Python code. The P4 implementation and controller required 1,600 and 240 lines, respectively. Each experiment, corresponding to a dataset, was defined using 400 lines of YAML, totaling 2,800 lines for all seven datasets.

5 EVALUATION

We evaluate SpliDT DTs for their end-to-end classification performance and resource utilization (§5.2) and perform microbenchmark experiments (§5.3).

5.1 Experiment Setup

Testbed Environment. Our testbed consists of two servers, serving as a traffic generator and receiver, respectively, connected via an Edgecore Wedge 100-32X Tofino1 [25] switch running the Stratum OS [16]. These servers are equipped with a 64-core Intel Xeon Platinum 8358P CPU @2.60 GHz with 512 GB RAM, running Proxmox VE (v8.0.4) [41]. Both servers also host a dual-port Intel XL710 [24] 10/40G NIC as well as an Nvidia ConnectX-6 [38] 100G NIC and run MoonGen [15] to generate and receive traffic. We use the same servers for our BO experiments and run 500 iterations with 16 parallel evaluations per iteration for each dataset.

Baselines, and Real-World Applications and Environments. Our baselines include Leo [28] and NetBeacon [58], two state-of-the-art data plane DT implementations. Both baselines implement stateful (top-k) DTs in the data plane and improve scalability with respect to tree depth via efficient TCAM rule generation. Table 3 lists the real-world datasets D1–7 used in our evaluations and describes their use for attack scenarios and security applications that range from intrusion detection to traffic classification and identification of modern attack patterns like DoS, botnets, and infiltration. Together, these datasets allow for a robust evaluation of the performance of SpliDT DTs against baselines across a diverse set of network security problems. Additionally, we measure the amount of resubmitted traffic for each dataset under two representative data center environments: E1: Webserver (WS) [42], with a flow duration distribution suggesting the existence of many concurrent long-lived flows; and E2: Hadoop (HD) [42], an environment with a majority of highly bursty mice flows.

Dataset Generation. To generate the data required to train SpliDT DTs (i.e., per-flow windows of packets), we modified the widely-used CICFlowMeter tool [17, 18, 31]. Identifying a flow by its 5-tuple, this tool collects flow-level statistics, computes 78 flow-level features, and dumps the resulting statistics at the FIN packet without retaining intermediate data. Considering an equal number of windows of packets for all flows, regardless of their size, we added support in CICFlowMeter to dump statistics at each window boundary (e.g., every quarter of a flow in the case there are 4 partitions) and reset the flow statistics after each window. NetBeacon's *phases* are similar to SpliDT's windows but differ in two ways: the size of NetBeacon's phase intervals increases exponentially (i.e., 2, 4, 8, 16, ...), and the flow statistics are retained between phases (i.e., same top-k features across all phases). For each dataset shown in Table 3, we generate up

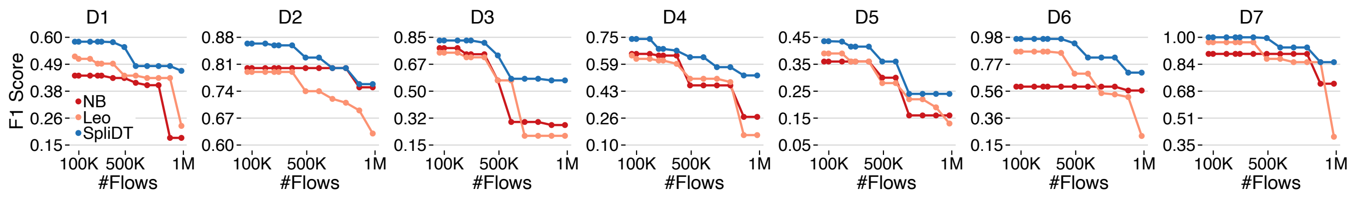

Figure 7: Pareto frontier of SpliDT vs. baselines, indicating the best F1 score for a given #flows in the data plane.

Data	#Flows	F1 Score			Depth / #Partitions			#Features			#TCAM Entries			Register Size (bits)		
		NB	Leo	SpliDT	NB	Leo	SpliDT	NB	Leo	SpliDT	NB	Leo	SpliDT	NB	Leo	SpliDT
D1	100K	0.44	0.51	0.58	13	11	27 / 3	6	5	41	3,041	16,384	3,802	192	160	128
	500K	0.43	0.44	0.56	15	5	28 / 3	3	4	39	4,071	2,048	3,867	96	128	64
	1M	0.18	0.23	0.46	3	3	13 / 1	2	2	2	86	2,048	4,874	64	64	64
D2	100K	0.80	0.79	0.86	13	10	28 / 2	6	6	43	3,740	8,192	10,633	192	192	128
	500K	0.80	0.74	0.83	15	6	24 / 3	2	4	40	5,028	2,048	11,965	64	128	64
	1M	0.75	0.63	0.76	15	3	19 / 2	1	2	2	2,097	2,048	1,161	32	64	32
D3	100K	0.78	0.75	0.83	15	11	30 / 2	6	6	39	3,776	16,384	2,397	192	192	128
	500K	0.57	0.57	0.73	13	6	13 / 4	4	4	41	1,713	2,048	3,224	128	128	64
	1M	0.28	0.21	0.57	15	3	15 / 1	2	2	2	505	2,048	291	64	64	64
D4	100K	0.65	0.62	0.74	15	10	15 / 1	6	7	7	2,462	8,192	3,458	192	224	224
	500K	0.46	0.50	0.63	15	10	15 / 1	3	2	3	1,508	8,192	6,089	96	64	96
	1M	0.27	0.16	0.52	13	3	12 / 1	2	2	2	1,289	2,048	446	64	64	64
D5	100K	0.36	0.39	0.44	9	10	42 / 5	6	7	41	4,470	8,192	19,337	192	224	128
	500K	0.30	0.28	0.36	9	6	12 / 1	3	4	3	6,102	2,048	3,798	96	128	96
	1M	0.16	0.13	0.24	9	3	13 / 1	2	2	2	907	2,048	1,244	64	64	64
D6	100K	0.60	0.87	0.97	5	10	14 / 5	3	6	32	251	8,192	420	96	192	96
	500K	0.60	0.70	0.93	5	10	22 / 3	3	3	34	251	8,192	583	96	96	64
	1M	0.57	0.22	0.71	7	3	20 / 2	2	2	3	407	2,048	740	64	64	32
D7	100K	0.90	0.97	1.00	7	8	17 / 5	5	7	19	340	2,048	209	160	224	128
	500K	0.90	0.87	1.00	7	7	17 / 6	4	3	13	285	2,048	146	128	96	64
	1M	0.72	0.40	0.85	7	3	5 / 1	2	2	2	284	2,048	20	64	64	64

Table 4: Model performance vs. resource usage tested against Tofino1 switch (6.4 Mbits TCAM budget, 12 stages) [25].

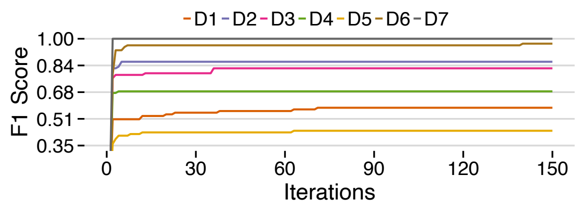

Figure 8: BO search iterations required to reach best F1 score. All datasets converge within 150 iterations.

Stages	D1	D2	D3	D4	D5	D6	D7	Mean
Data Fetch	0.9s	0.32s	0.01s	0.07s	0.91s	0.24s	0.18s	0.38s
Training	556s	228s	10s	84s	725s	163s	111s	268.14s
Optimizer	33s	45s	43s	43s	30s	32s	37s	37.57s
Rulegen	0.8s	0.99s	0.91s	1.08s	0.71s	0.71s	0.91s	0.87s
Backend	42μs	45μs	43μs	42μs	46μs	44μs	47μs	44.6μs
Total Time	589s	273s	54s	128s	756s	196s	148s	306.29s

Table 5: Average per-iteration breakdown of the various stages of SpliDT framework.

to 7 partitions[3] for SpliDT; for NetBeacon [58], we consider the same phases as in the public artifact.[4]

5.2 End-to-End Analysis

Pareto Frontier. Across all datasets, SpliDT DTs consistently outperform the baselines in the sense of yielding a better tradeoff curve; that is, achieving a higher accuracy for the same number of flows. More precisely, by virtue of consistently achieving a better tradeoff between model performance (i.e., accuracy) and model scalability (i.e., number of flows), the SpliDT DTs define the Pareto frontier for all considered datasets (Figure 7). Moreover, the obtained tradeoff curves are monotonically decreasing, with all models delivering better accuracy when supporting fewer flows and

[3]We explored using more partitions for SpliDT but classification performance significantly drops with > 7 partitions.

[4]NetBeacon phases: 2, 4, 8, 32, 256, 512, 2048.

then having to compromise model size and feature coverage to support more flows but at a cost of lower accuracy.

Feature Scalability and Resource Utilization. As shown in Table 4, SpliDT DTs generally achieve the best accuracy across all datasets, with a balanced trade-off between the depth of the tree and the number of partitions. They require less register space while maintaining competitive accuracy. TCAM entries and register sizes vary significantly, with SpliDT DTs optimizing TCAM usage while keeping register size manageable. Like shown in D6, SpliDT supports more features compared to the baselines for all flow sizes (in 96, 64, 32-bit budget for 100k, 500k, and 1M flows), while also having deeper trees. The results highlight the efficiency of SpliDT DTs in achieving high accuracy while using the same resources as the baselines.

Offline Software Overhead. SpliDT's offline model search framework reaches maximum accuracy for all datasets within 150 search iterations (Figure 8). The per-iteration timing breakdown provided in Table 5 shows that training is the most time-consuming of all stages (88% on average), followed by the BO optimization stage (about 12% on average) that uses HyperMapper [37] to produce an optimal, high-performing model. Training SpliDT DTs is expensive because of their recursive nature, which in turn requires considering a larger number of subtrees with larger overall tree depths as well as additional steps to partition the training data for each recursion. Despite these overheads, the average experiment delivers a high-performing model in 3.8 hours.

5.3 Microbenchmarks

SpliDT framework finds optimal model parameters for partitioned decision trees. The SpliDT framework navigates the parameter spaces available for tree partitioning to deliver best-performing models for any given number of flows. Figure 9a shows the Pareto frontier when we fix the overall tree depth (to be 10, 20, and 30, respectively) and vary the other two parameters (features per sub-tree and number of partitions) on the Pareto frontier. For example, for D2, the Pareto frontier for depth 30 is better compared to that of depth 10 for all but the case of 1M flows or so. In general, each dataset benefits from a different tree depth depending on feature complexity and class distribution, and the SpliDT framework effectively explores the design space to deliver an optimal model for a given use case. Similarly, as shown in Figure 9b where we fix the number of partitions to be 1, 3, and 5, respectively, fewer partitions yield overall better Pareto frontiers, mainly because a small number of partitions translates into more observed packets in each window. Finally, limiting the number of features per sub-tree to 1, 2, and 3, respectively, we observe in Figure 9c that using more features typically results in the best accuracy but comes at a cost of supporting fewer flows. On the other hand, reducing the number of features per sub-tree allows supporting a large number of flows but at the cost of low accuracy.

SpliDT DTs outperform baselines for any given TCAM budget. Figure 10 shows that SpliDT DTs have higher accuracy compared to the baselines across varying TCAM budgets. This result is a direct consequence of reducing the match key size, achieved by decreasing the number of features (k) matched in the model table for the active subtree.

SpliDT DTs recirculate limited traffic and do not impact end-to-end flow-level time-to-detection. Table 6 shows that for the two environments (E1: Webserver, E2: Hadoop), packet recirculation for SpliDT DTs remains well within the resubmission buffer capacity (100 Gbps). Moreover, in Figure 12, we show that in the case of D3, the per-flow time-to-detection—the time it takes from the start of tree traversal until a final inference decision has been made—closely matches that of NetBeacon, implying that recirculation for SpliDT DTs has no impact on model performance.

SpliDT DTs require constant register space to accommodate any number of features. SpliDT DTs maintain constant register space (for k features per subtree), independent of the total number of different features utilized across the entire tree (Figure 11). As also shown in (Table 4, SpliDT can store up to 43, 32-bit features in the budget of 128 bits. This scalability result implies that because of the diligent use of stateful memory, more complex DT models requiring more TCAM entries can be offloaded to the data plane. In contrast, baseline models require a proportional register space to support a fixed number of features, highlighting the feature scalability advantage of the SpliDT design.

SpliDT DTs can accommodate a larger number of flows with reduced bit-precision. We reduce bit-precision for all features from 32-bit integers to 16 and 8-bits, respectively, and measure the impact on model accuracy and the number of supported flows. Figure 13 shows that the reduction in bit-precision equally affects all models, mainly because all are decision trees. On average, we observe a 7% (14%) drop in accuracy with 16-bit (8-bit) precision, while the maximum number of supported flows increases to 2M (4M). Note that in both scenarios, SpliDT results in a better Pareto frontier compared to baselines, again owing to the enhanced feature coverage provided by SpliDT DTs.

6 LIMITATIONS & FUTURE WORK

Adaptive Window Sizing. Currently, SpliDT uses fixed per-flow window sizes, which may not always align with real-world traffic dynamics. This limitation could impact model accuracy and resource efficiency, particularly for bursty or highly variable traffic patterns. Future work will explore

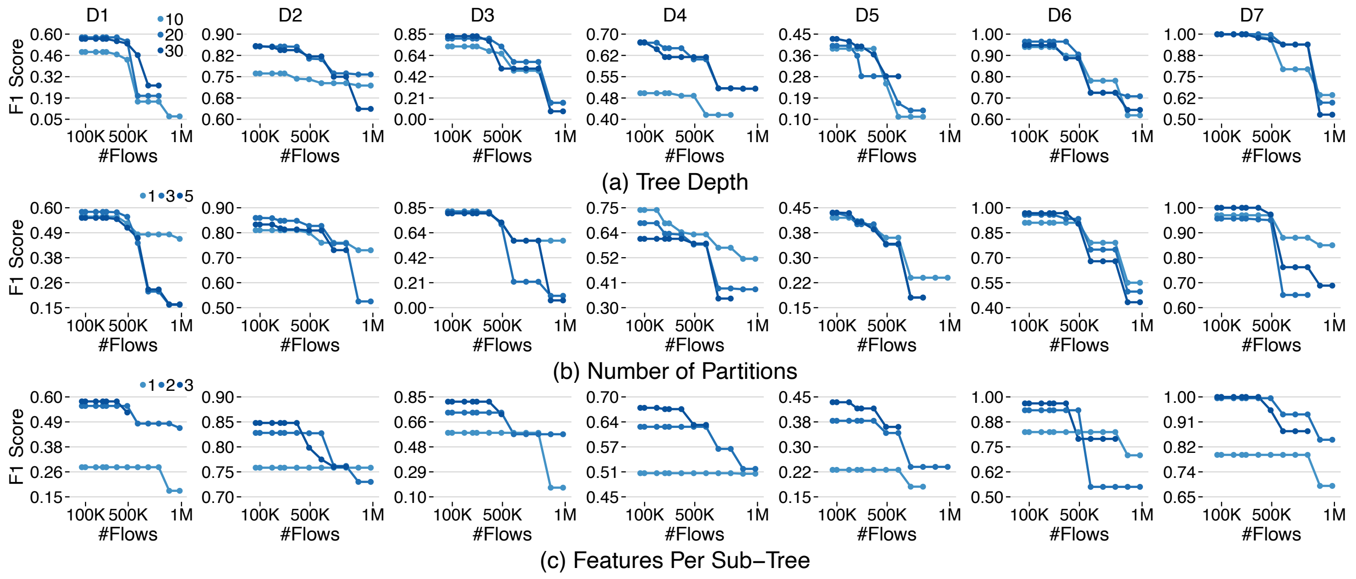
Figure 9: Pareto frontiers for SpliDT partitioned trees (top to bottom): (a) fixed depth, (b) fixed number of partitions, and (c) fixed features per subtree.

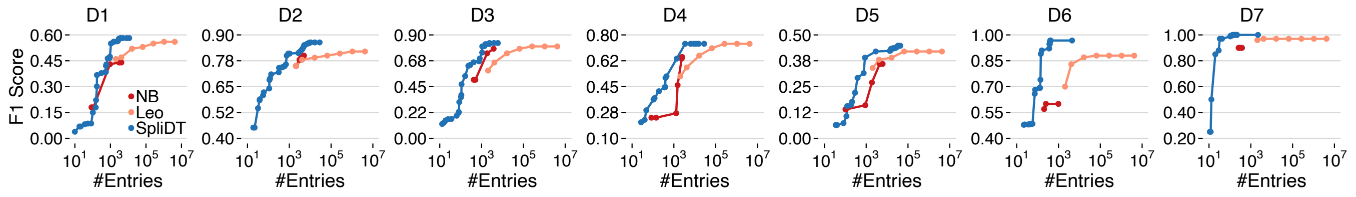
Figure 10: Comparison of #TCAM entries against F1 score for SpliDT vs. baselines.

#Flows	D1		D2		D3		D4		D5		D6		D7	
	E1: WS	E2: HD	E1: WS	E2: HD	E1: WS	E2: HD	E1: WS	E2: HD	E1: WS	E2: HD	E1: WS	E2: HD	E1: WS	E2: HD
100K	1.46±0.83	3.0±1.19	0.98±0.55	2.0±0.8	0.98±0.55	2.0±0.8	0±0	0±0	2.44±1.38	5.0±1.99	2.44±1.38	5.0±1.99	2.44±1.38	5.0±1.99
500K	7.32±4.15	14.99±5.97	7.32±4.15	14.99±5.97	9.75±5.53	19.98±7.96	0±0	0±0	0±0	0±0	7.32±4.15	14.99±5.97	14.63±8.3	29.97±11.93
1M	0±0	0±0	9.75±5.53	19.98±7.96	0±0	0±0	0±0	0±0	0±0	0±0	9.75±5.53	19.98±7.96	0±0	0±0

Table 6: Maximum recirculation bandwidth (Mbps) of SpliDT partitioned trees when processing datasets (D1–7) for the two datacenter environments, E1: Webserver (WS) and E2: Hadoop (HD), with varying flow sizes.

adaptive window sizing across partitions, allowing SpliDT to optimize feature extraction using real flow characteristics.

Security & Robustness. SpliDT relies on flow size metadata stored in packet headers, making it susceptible to spoofing attacks that manipulate window boundaries. If an attacker alters flow size fields, this could lead to incorrect partitioning, misclassifications, resource exhaustion, or even denial-of-service (DoS) attacks. Future work will focus on enhancing security mechanisms, such as redundancy mechanisms or anomaly detection, to protect against header manipulation.

Towards explainable AI with SpliDT. DTs play a central role in explainable AI due to their interpretability and transparency, making them well-suited for high-stakes decision-making. Whether obtained directly by training interpretable models [43] or indirectly through explainable approximations of trained black box models [27], DTs allow domain experts to audit and verify decisions. Future work will extend SpliDT to navigate a broader trade-off space, optimizing not just for scalability and efficiency, but also for fidelity with respect to a domain expert-approved and trusted DT model.

7 RELATED WORK

Hybrid Telemetry Systems. SpliDT builds on recent efforts that design hybrid telemetry systems and integrate data plane and control plane processing to enable scalable, adaptive network monitoring. These systems leverage techniques such as iterative query refinement [19, 44], runtime query modification [57, 59], and dynamic query planning [35, 44] to optimize data collection while preserving normal packet

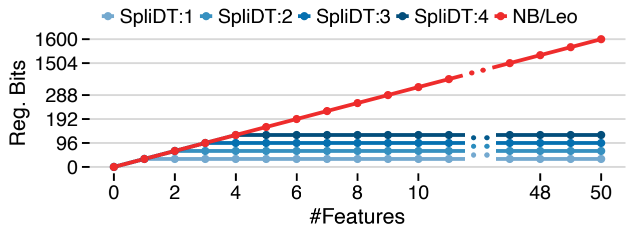

Figure 11: Register sizes (in bits) vs. number of features supported by each model. SpliDT:k is a partitioned tree with k features per subtree.

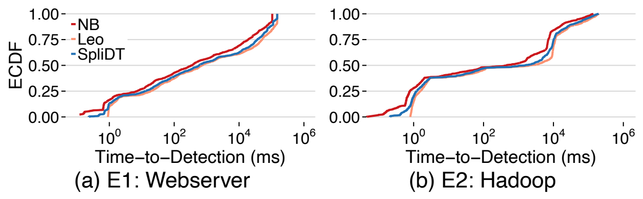

Figure 12: Time-to-detection (TTD) of D3 for environments (E1–E2). Other datasets show a similar trend.

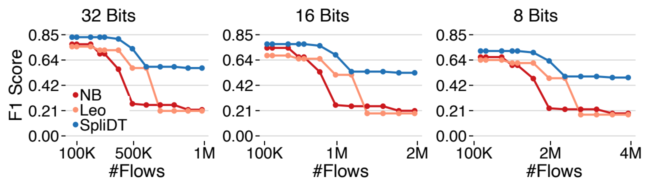

Figure 13: Pareto frontier of D3 vs. bit precisions.

forwarding. Using similar principles, SpliDT applies incremental processing and runtime adaptability to DT inference rather than telemetry. Like Sonata [19] and DynaMap [44], SpliDT supports progressive execution, and similar to Newton [59], DynaMap [44], and Flymon [57], it leverages packet recirculation. However, instead of refining queries or dynamically updating the query execution plan, SpliDT uses tree partitioning to perform DT inference via subtree transitions.

Decision Tree-Based Inference in the Data Plane. Several efforts have mapped DTs to programmable switches for real-time classification. Leo [28] deploys scalable DTs by optimizing match-action table (MAT) representations, while NetBeacon [58] introduces ternary table encoding for large-scale inference. Mousika [51] optimize binary DTs via knowledge distillation, reducing resource usage while maintaining hardware compatibility. Planter [56] integrates packet-level summaries into DT-based classification, while IISY [52] maps ML models to match-action pipelines, improving inference efficiency. pForest [7] extends DTs to random forests, dynamically adapting feature selection based on real-time traffic. Unlike prior DT-based solutions, SpliDT removes the static top-k feature constraint, enabling dynamic feature allocation across tree partitions. In contrast to existing methods that apply single-pass DT execution, SpliDT introduces incremental inference via subtree transitions, optimizing stateful feature storage and *scaling to millions of flows* while minimizing TCAM overhead.

Neural Network-Based Inference in the Data Plane. NN-based approaches in switches prioritize high expressiveness but face resource constraints. Taurus [46] introduces a MapReduce-inspired ML inference framework, while Homunculus [47] automates ML model deployment on SmartNICs. Brain-on-Switch (BoS) [55] integrates RNNs and transformers for sequential data processing, improving accuracy but requiring significant compute resources. N3IC [45] explores binary neural networks for low-latency inference, and ServeFlow [33] introduces neural accelerators for high-accuracy classification with minimal resource overhead. AC-DC [29] balances performance and efficiency for ML-based classification in dynamic traffic conditions. Unlike existing NN-based solutions, SpliDT is optimized for strict hardware constraints in programmable switches, achieving line-rate inference without specialized accelerators. While BoS achieves higher accuracy for sequential tasks, SpliDT enables scalable, resource-efficient DT inference by leveraging partitioned tree processing and recirculation-based register reuse, making it a practical choice for performing inference at line rate.

8 CONCLUSION

We demonstrate in this paper that SpliDT significantly outperforms state-of-the-art approaches such as NetBeacon [58] and Leo [28] by achieving better model accuracy while accommodating up to 5× more stateful features and scaling to millions of flows. Despite this improved scalability, SpliDT maintains a low recirculation overhead ($\leq$ 50 Mbps) and a low time-to-detection (TTD). These results are enabled by two core innovations: (1) *a tree partitioning framework with subtree ID (SID) tracking*, which enables window-based inference and register reuse by dynamically managing feature selection and TCAM rules one subtree at a time; (2) *a window-based training pipeline using Bayesian Optimization,* which optimizes tree depth, feature allocation, and tree partitions to balance accuracy and scalability; and By jointly optimizing the tree structure offline and executing lightweight inference online, SpliDT efficiently supports real-time flow classification in resource-constrained environments.

More broadly, SpliDT advances the feasibility of deploying ML models directly in the data plane, demonstrating that programmable switches can support complex, adaptive inference at scale without sacrificing performance. This work paves the way for future advancements in integrating more expressive ML models into high-speed networks, optimizing data plane resource management, and ensuring robust and safe real-time decision-making for modern network infrastructures.

REFERENCES

[1] Ansel, J., Kamil, S., Veeramachaneni, K., Ragan-Kelley, J., Bosboom, J., O'Reilly, U.-M., and Amarasinghe, S. Opentuner: An Extensible Framework for Program Autotuning. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation* (2014).

[2] Barradas, D., Santos, N., Rodrigues, L., Signorello, S., Ramos, F. M. V., and Madeira, A. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. In *NDSS* (2021).

[3] Benson, T., Anand, A., Akella, A., and Zhang, M. Understanding data center traffic characteristics. *SIGCOMM CCR* (2010).

[4] Bergstra, J., Yamins, D., and Cox, D. D. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *ICML* (2013).

[5] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. P4: Programming Protocol-Independent Packet Processors. In *ACM SIGCOMM CCR* (2014).

[6] Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *ACM SIGCOMM* (2013).

[7] Busse-Grawitz, C., Meier, R., Dietmüller, A., Bühler, T., and Vanbever, L. pForest: In-Network Inference with Random Forests. *arXiv preprint arXiv:1909.05680* (2022).

[8] Canadian Institute for Cybersecurity. CIC IDS 2017 Dataset. https://www.unb.ca/cic/datasets/ids-2017.html, 2017.

[9] Canadian Institute for Cybersecurity. CIC IDS 2018 Dataset. https://www.unb.ca/cic/datasets/ids-2018.html, 2018.

[10] Canadian Institute for Cybersecurity. CIC IoT 2023 Dataset. https://www.unb.ca/cic/datasets/iotdataset-2023.html, 2023.

[11] Canadian Institute for Cybersecurity. CIC IoMT 2024 Dataset. https://www.unb.ca/cic/datasets/iomt-dataset-2024.html, 2024.

[12] Canadian Institute for Cybersecurity. CIC VPN Dataset. https://www.unb.ca/cic/datasets/vpn.html, Accessed: 2024-01-30.

[13] Dong, M., Li, Q., Zarchy, D., Godfrey, P. B., and Schapira, M. PCC: Re-Architecting Congestion Control for Consistent High Performance. In *USENIX NSDI* (2015).

[14] Doriguzzi-Corin, R., Millar, S., Scott-Hayward, S., Martínez-del Rincón, J., and Siracusa, D. Lucid: A practical, lightweight deep learning solution for ddos attack detection. *IEEE Transactions on Network and Service Management* (2020).

[15] Emmerich, P., Gallenmüller, S., Raumer, D., Wohlfart, F., and Carle, G. Moongen: A Scriptable High-Speed Packet Generator. In *ACM IMC* (2015).

[16] Foundation, O. N. Stratum OS. https://www.opennetworking.org/stratum/. Accessed on 01/30/2025.

[17] Gil, G. D., Lashkari, A. H., Mamun, M., and Ghorbani, A. A. Characterization of encrypted and VPN traffic using time-related features. In *Proceedings of the 2nd international conference on information systems security and privacy* (2016).

[18] (GitHub), A. H. L. CICFlowMeter. https://github.com/ahlashkari/CICFlowMeter/tree/master, last accessed: 12/30/2024.

[19] Gupta, A., Harrison, R., Canini, M., Feamster, N., Rexford, J., and Willinger, W. Sonata: Query-driven network telemetry. In *Proc. ACM SIGCOMM* (2018).

[20] Guthula, S., Beltiukov, R., Battula, N., Guo, W., Gupta, A., and Monga, I. netFound: Foundation Model for Network Security. *arXiv preprint arXiv:2310.17025* (2025).

[21] Ha, S., Rhee, I., and Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating Systems Review* (2008).

[22] Handley, M., Raiciu, C., Agache, A., Voinescu, A., Moore, A. W., Antichi, G., and Wójcik, M. Re-Architecting Datacenter Networks and Stacks for Low Latency and High Performance. In *ACM SIGCOMM* (2017).

[23] He3 Team. Understanding the CRC32 Hash: A Comprehensive Guide. https://he3.app/blogs/understanding-the-crc32-hash-a-comprehensive-guide/, 2024.

[24] Intel. Intel Ethernet Controller 700 Series - Open vSwitch Hardware Acceleration Application Note. https://builders.intel.com/docs/networkbuilders/intel-ethernet-controller-700-series-open-vswitch-hardware-acceleration-application-note.pdf, last accessed: 01/30/2024.

[25] Intel. Tofino: P4-programmable Ethernet switch ASIC that delivers better performance at lower power. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, last accessed: 12/31/2024.

[26] Intel. Tofino2: Second-generation P4-programmable Ethernet Switch ASIC that Continues to Deliver Programmability without Compromise. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html, last accessed: 12/31/2024.

[27] Jacobs, A. S., Beltiukov, R., Willinger, W., Ferreira, R. A., Gupta, A., and Granville, L. Z. AI/ML for Network Security: The Emperor has no Clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (2022).

[28] Jafri, S. U., Rao, S., Shrivastav, V., and Tawarmalani, M. Leo: Online ML-based Traffic Classification at Multi-Terabit Line Rate. In *USENIX NSDI* (2024).

[29] Jiang, X., Liu, S., Naama, S., Bronzino, F., Schmitt, P., and Feamster, N. AC-DC: Adaptive Ensemble Classification for Network Traffic Identification. *arXiv preprint arXiv:2302.11718* (2023).

[30] Knudde, N., van der Herten, J., Dhaene, T., and Couckuyt, I. GPflowOpt: A Bayesian Optimization Library Using TensorFlow. *arXiv preprint arXiv:1711.03845* (2017).

[31] Lashkari, A. H., Gil, G. D., Mamun, M. S. I., and Ghorbani, A. A. Characterization of tor traffic using time based features. In *International Conference on Information Systems Security and Privacy* (2017).

[32] Li, Y., Miao, R., Liu, H. H., Zhuang, Y., Feng, F., Tang, L., Cao, Z., Zhang, M., Kelly, F., Alizadeh, M., and Yu, M. HPCC: High Precision Congestion Control. In *ACM SIGCOMM* (2019).

[33] Liu, S., Shaowang, T., Wan, G., Chae, J., Marques, J., Krishnan, S., and Feamster, N. ServeFlow: A Fast-Slow Model Architecture for Network Traffic Analysis. *arXiv preprint arXiv:2402.03694* (2024).

[34] Mao, H., Netravali, R., and Alizadeh, M. Neural Adaptive Video Streaming with Pensieve. In *ACM SIGCOMM* (2017).

[35] Misa, C., O'Connor, W., Durairajan, R., Rejaie, R., and Willinger, W. Dynamic Scheduling of Approximate Telemetry Queries. In *USENIX NSDI* (2022).

[36] Montazeri, B., Li, Y., Alizadeh, M., and Ousterhout, J. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *ACM SIGCOMM* (2018).

[37] Nardi, L., Bodin, B., Saeedi, S., Vespa, E., Davison, A. J., and Kelly, P. H. Algorithmic Performance-accuracy Trade-off in 3D Vision Applications using Hypermapper. In *IEEE IPDPSW* (2017).

[38] Nvidia. ConnectX-6 Network Adapters. https://www.nvidia.com/en-us/networking/ethernet/connectx-6-dx/, last accessed: 01/30/2025.

[39] pandas. pandas. https://pandas.pydata.org/, last accessed: 01/29/2025.

[40] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. Scikit-learn: Machine learning in Python. *Journal of machine learning research* (2011).

[41] Proxmox. Proxmox. https://www.proxmox.com/en/, last accessed: 01/30/2025.

[42] Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. Inside the Social Network's (Datacenter) Network. In *ACM SIGCOMM* (2015).

[43] Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*.

[44] Shou, C., Bhatia, R., Gupta, A., Harrison, R., Lokshtanov, D., and Willinger, W. Query planning for robust and scalable hybrid network telemetry systems. *Proceedings of the ACM on Networking 2*, CoNEXT1 (2024), 1–27.

[45] Siracusano, G., Galea, S., Sanvito, D., Malekzadeh, M., Antichi, G., Costa, P., Haddadi, H., and Bifulco, R. Re-architecting Traffic Analysis with Neural Network Interface Cards. In *USENIX NSDI* (2022).

[46] Swamy, T., Rucker, A., Shahbaz, M., Gaur, I., and Olukotun, K. Taurus: A Data Plane Architecture for Per-Packet ML. In *ASPLOS* (2022).

[47] Swamy, T., Zulfiqar, A., Nardi, L., Shahbaz, M., and Olukotun, K. Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks. In *ASPLOS* (2023).

[48] Tensorflow. Tensorflow. https://www.tensorflow.org/, last accessed: 01/29/2025.

[49] Wikipedia. Bayesian Optimization. https://en.wikipedia.org/wiki/Bayesian_optimization, last accessed: 01/30/2025.

[50] Winstein, K., and Balakrishnan, H. TCP ex machina: Computer-generated Congestion Control. In *ACM SIGCOMM CCR* (2013).

[51] Xie, G., Li, Q., Dong, Y., Duan, G., Jiang, Y., and Duan, J. Mousika: Enable General In-Network Intelligence in Programmable Switches by Knowledge Distillation. In *IEEE INFOCOM* (2022).

[52] Xiong, Z., and Zilberman, N. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *ACM HotNets* (2019).

[53] Yan, F. Y., Ayers, H., Zhu, C., Fouladi, S., Hong, J., Zhang, K., Levis, P., and Winstein, K. Learning in situ: A Randomized Experiment in Video Streaming. In *USENIX NSDI* (2020).

[54] Yan, F. Y., Ma, J., Hill, G. D., Raghavan, D., Wahby, R. S., Levis, P., and Winstein, K. Pantheon: The Training Ground for Internet Congestion-Control Research. In *USENIX ATC* (2018).

[55] Yan, J., Xu, H., Liu, Z., Li, Q., Xu, K., Xu, M., and Wu, J. Brain-on-switch: towards advanced intelligent network data plane via NN-driven traffic analysis at line-speed. In *USENIX NSDI* (2024).

[56] Zheng, C., Zang, M., Hong, X., Bensoussane, R., Vargaftik, S., Ben-Itzhak, Y., and Zilberman, N. Automating In-Network Machine Learning. *arXiv preprint arXiv:2205.08824* (2022).

[57] Zheng, H., Tian, C., Yang, T., Lin, H., Liu, C., Zhang, Z., Dou, W., and Chen, G. Flymon: enabling on-the-fly task reconfiguration for network measurement. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 486–502.

[58] Zhou, G., Liu, Z., Fu, C., Li, Q., and Xu, K. An Efficient Design of Intelligent Network Data Plane. In *USENIX Security Symposium* (2023).

[59] Zhou, Y., Zhang, D., Gao, K., Sun, C., Cao, J., Wang, Y., Xu, M., and Wu, J. Newton: intent-driven network traffic monitoring. In *ACM CoNEXT* (2020), pp. 295–308.