# FLOWGUARD: A MACHINE LEARNING SYSTEM FOR DDOS MITIGATION VIA LLM-AUGMENTED IN-SWITCH CLASSIFICATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Distributed denial-of-service (DDoS) attacks remain a severe threat to network availability. Programmable switches enable line-rate DDoS mitigation, but their limited computational and memory resources restrict in-switch classifiers to lightweight models, which are both less accurate on attack traffic that resembles legitimate flows and quickly outdated as attack patterns evolve. To address these challenges, we present FlowGuard, a closed-loop hybrid DDoS mitigation framework that combines line-rate in-switch classification with LLM-assisted out-of-switch analysis and classifier adaptation. In the switch data plane, a lightweight decision tree performs line-rate flow classification and enables immediate mitigation. Outside the switch, an LLM analyzes compact window-level traffic statistics to produce risk assessments, victim subnets, and rationales for flagged windows, from which FlowGuard derives flow-level weak labels. When statistical drift is detected, a second LLM stage determines whether retraining is warranted. If retraining is approved, FlowGuard trains a new decision tree from these weak labels together with the original labeled data, and deploys it only after it passes a safety gate. Through this closed-loop design, FlowGuard combines line-rate DDoS mitigation with the ability to adapt to evolving traffic patterns. Experiments on CIC-DDoS2019 and IoT-23 show that FlowGuard achieves an $F_1$ score of $0.968$, representing relative improvements of $24.6\%$ and $43.8\%$ over our static in-switch decision-tree and the in-network baseline SISTAR, respectively. Code and data are available at `https://anonymous.4open.science/r/FlowGuard`.

## 1 INTRODUCTION

Distributed denial-of-service (DDoS) attacks remain a persistent and disruptive threat to network services. By generating massive volumes of malicious traffic, these attacks can overwhelm network resources and severely degrade service availability. The proliferation of vulnerable Internet of Things (IoT) devices has further amplified this threat. Compromised IoT devices can be recruited into large, geographically distributed botnets (Antonakakis et al., 2017), resulting in an increasingly large population of attack sources. At the same time, the devices generating attack traffic can change rapidly over time, causing the characteristics of the observed traffic, such as packet sizes and sending rates, to evolve continuously. Such large-scale and highly dynamic behavior has been observed in real-world attacks. For example, during the Porcupine attack, Cloudflare observed approximately 200,000 source IP addresses in each attack burst and 1.2 million unique source IP addresses across 16 bursts, with an estimated 50,000–100,000 bots changing their IP addresses within only 80 minutes (Majkowski, 2017). This combination of large-scale and rapidly changing attack sources poses significant challenges to existing DDoS defense systems.

To defend against such distributed and dynamic attacks, machine learning (ML) has emerged as a promising approach for distinguishing attack traffic from legitimate traffic by learning discriminative patterns from multiple traffic features. However, effective DDoS mitigation must also process traffic at high speed to respond before attack traffic overwhelms network resources. To apply ML-based methods on high-speed network traffic, programmable switches provide an attractive platform by enabling line-rate traffic processing directly in the data plane. However, the limited hardware resources of programmable switches constrain the complexity of ML models that can be deployed.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

For example, an Intel Tofino 1 switch provides only 12 match-action stages in each ingress or egress pipeline, limiting the depth of sequential computation that can be performed on each packet. As a result, in-switch ML models must remain lightweight, restricting their ability to capture complex traffic patterns.

The combination of limited switch resources and rapidly evolving DDoS traffic introduces two key challenges. First, attackers can deliberately craft DDoS traffic to resemble legitimate traffic, making malicious and legitimate traffic difficult to distinguish. Lightweight in-switch models may lack sufficient expressive capacity to capture these subtle differences, resulting in degraded detection accuracy. Second, DDoS attacks can evolve rapidly as attackers modify their traffic patterns or change the set of bots used to generate attack traffic. Such changes can cause the traffic distribution observed during deployment to deviate substantially from the data used for training, causing models trained on previously observed attacks to become outdated quickly (Gama et al., 2014; Yang et al., 2021). Together, these challenges make it difficult for a fixed lightweight model to maintain accurate and reliable detection over long-term deployment.

Addressing these challenges requires preserving lightweight, line-rate classification in the switch while providing the ability to adapt the classifier as traffic evolves. We therefore present FLOWGUARD, a closed-loop hybrid DDoS mitigation framework that combines an in-switch decision tree with LLM-assisted traffic analysis and decision-tree adaptation. Specifically, a lightweight decision tree runs in the switch data plane to perform flow-level classification at line rate and enable immediate mitigation, while the switch simultaneously maintains compact window-level traffic summaries. These summaries are periodically exported from the switch and analyzed by an LLM to assess aggregate traffic behavior and identify attack windows and their victim subnets. Based on the LLM's analysis of these summaries, FlowGuard further derives flow-level weak labels for subsequent model adaptation. Meanwhile, a drift monitor detects significant changes in the traffic features used by the decision tree. When such drift is detected, an LLM-assisted adaptation module determines whether the current decision tree should be retrained. If retraining is warranted, FlowGuard trains a candidate decision tree using the newly derived weak labels together with the original labeled data, validates its performance, and deploys it only if it passes the safety gate. This closed loop allows FlowGuard to retain line-rate mitigation while keeping the in-switch classifier responsive to evolving traffic.

We evaluate FlowGuard using CIC-DDoS2019 and IoT-23 to examine its effectiveness under changing traffic distributions. Our evaluation shows that FlowGuard achieves a window-level $F_1$ score of $0.968$, corresponding to relative improvements of $24.6\%$ over a static in-switch decision tree and $43.8\%$ over the in-network baseline SISTAR. Separately, LLM-triggered adaptation improves the deployed tree on the evaluation stream from $F_1 = 0.777$ to $0.852$, increasing the fraction of attack packets dropped from $61.5\%$ to $69.2\%$ while reducing benign-packet drops from $0.74\%$ to $0.64\%$. Together, these results demonstrate that FlowGuard can combine line-rate DDoS mitigation with accurate window-level analysis and adaptive in-switch classification under the resource constraints of programmable switches.

Our main contributions are summarized as follows:

- FlowGuard combines a lightweight in-switch decision tree with LLM-assisted window-level traffic analysis, preserving line-rate classification and immediate mitigation while improving detection accuracy beyond what the lightweight in-switch classifier can achieve alone.
- FlowGuard derives flow-level weak labels from LLM-based window-level analysis and uses them in a drift-triggered adaptation process. When traffic drift is detected, an LLM determines whether retraining is necessary, and a candidate decision tree is trained using the weak labels together with the original labeled data, validated, and deployed only if it passes the safety gate. This design enables FlowGuard to adapt to evolving traffic while avoiding unnecessary retraining and preventing poorly performing models from being deployed.
- We evaluate FlowGuard on CIC-DDoS2019 and IoT-23 and show that it achieves an $F_1$ score of $0.968$, improving over both the static in-switch decision tree and the in-network baseline SISTAR. The evaluation further demonstrates the effectiveness of window-level LLM analysis and drift-triggered adaptation under evolving traffic distributions.

## 2 Background and Motivation

In this section, we provide background on programmable switches and review existing DDoS defense approaches that motivate the design of FlowGuard.

**Opportunities of Programmable Switches.** Programmable switches have become an attractive platform for high-speed DDoS defense due to three key advantages: (1) *line-rate processing*, supporting traffic rates of up to 6.5 Tbps for programs that fit within the available hardware resources; (2) *programmability*, allowing operators to flexibly extract and process packet-header fields required by different defense mechanisms; and (3) *cost efficiency*, with costs comparable to legacy switches of the same speed but substantially lower capital investment than dedicated DDoS mitigation appliances, such as Cisco Guard.

The line-rate processing and programmability of modern switches are enabled by the Protocol Independent Switch Architecture (PISA) (Bosshart et al., 2013; 2014), as illustrated in Figure 1. Incoming packets first pass through a programmable parser, which extracts packet-header fields. The extracted fields are then processed through a sequence of ingress match-action stages, where match tables identify packet conditions and action units perform operations such as arithmetic computation and stateful register access. However, the PISA pipeline imposes strict computational and resource constraints: packet processing is loop-free, complex operations such as multiplication and division are generally unsupported, and only limited pipeline stages, arithmetic units, and on-chip memory are available.

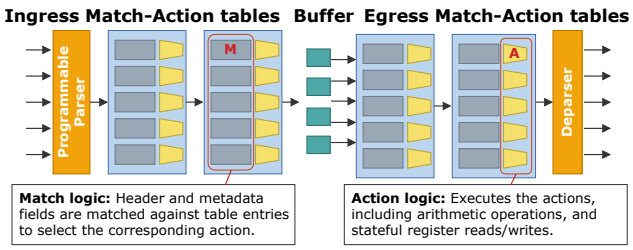


Figure 1: Programmable Switch Architecture.

**Programmable-Switch DDoS Defense.** Building on these advantages, programmable switches have been widely explored for DDoS defense (Chen et al., 2023). Early approaches often rely on predefined rules and traffic statistics. For instance, Poseidon (Zhang et al., 2020) and Jaqen (Liu et al., 2021) classify suspicious traffic using configured thresholds, while Ripple (Xing et al., 2021) and Mew (Zhou et al., 2023b) extend similar mechanisms to distributed settings, and ACC-Turbo (Gran Alcoz et al., 2022) targets pulse-wave attacks. To make statistics-based detection more memory-efficient, some approaches use compact probabilistic data structures to maintain aggregate traffic statistics. For example, P4LogLog and P4NEntropy provide memory-efficient estimates of flow cardinality and normalized traffic entropy, respectively (Ding et al., 2022). Beyond improving measurement efficiency, recent work has explored machine learning to improve detection accuracy and capture more complex relationships among traffic features. For example, pForest (Busse-Grawitz et al., 2022) and SwitchTree (Lee & Singh, 2020) map tree-based models to programmable switches, NetBeacon (Zhou et al., 2023a) reduces decision-tree inference overhead through compact range encoding, and Leo (Jafri et al., 2024) supports decision-tree deployment and runtime updates. Caravan (Zhang et al., 2024) retrains them online with labeling agents such as foundation models. Yet, even with machine learning, in-switch classifiers typically make decisions at the individual-flow level, which may miss malicious behavior that becomes apparent only when traffic is considered collectively across multiple flows or over time. This limitation has motivated approaches that analyze traffic beyond individual flows. NetRadar (Pan et al., 2026) exploits similarities among traffic directed toward multiple servers in a victim network, while DoLLM (Li et al., 2024) reorganizes flow records into sequences and applies a language model to capture cross-flow patterns, and NetLLM (Wu et al., 2024) adapts LLMs to other networking tasks.

**Motivation.** Existing DDoS defenses generally derive their detection logic from prior knowledge of attack traffic, either through manually designed rules or models learned from previously observed data. However, as attackers continuously modify their traffic characteristics and bot populations, such prior knowledge can quickly become outdated, causing detection accuracy to degrade. This motivates the need for a self-adaptive DDoS mitigation system that can continuously monitor traffic evolution, recognize when its existing detection logic becomes outdated, and adapt accordingly.

# 3 System Design

## 3.1 Threat Model and Design Requirements

**Threat Model.** We consider volumetric DDoS attacks that aim to congest a target network link by exhausting its available bandwidth. We assume that such attacks are launched by large, geographically distributed sets of compromised hosts. Because the attack traffic originates from compromised end hosts using ordinary network protocols, individual malicious flows may exhibit observable characteristics similar to legitimate flows. To evade detection systems, the adversary may periodically vary the characteristics of the generated traffic, including flow rates, packet-level statistics, and destination distributions. Consequently, the attack traffic observed by the defender may evolve substantially over time and deviate from previously observed attack patterns.

**Design Requirements.** Under the above threat model, an effective DDoS defense system should satisfy three key requirements.

- **R1 Low-Latency Mitigation.** Volumetric DDoS attacks can overwhelm link capacity and switch buffers within seconds, causing service conditions to deteriorate from increased latency and packet loss to prolonged disruption as congestion persists. The defense therefore needs to detect and mitigate attacks with sufficiently low latency to prevent congestion from escalating into severe network disruption.
- **R2 Adaptability to Evolving Attacks.** As traffic characteristics evolve, models trained on previously observed traffic may become outdated. The defense therefore needs to remain effective under changing traffic conditions by adapting its model when necessary.
- **R3 Low-Overhead Monitoring.** Large-scale DDoS attacks may involve a massive number of concurrent flows, making detailed per-flow data export impractical. The defense therefore needs to provide sufficient traffic information for higher-level analysis with low monitoring and communication overhead.

## 3.2 FlowGuard Overview

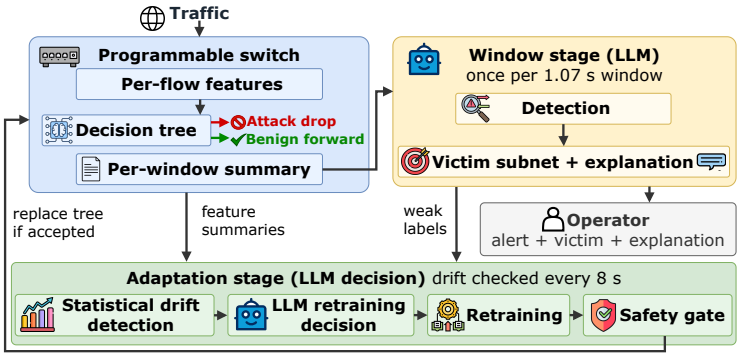



Figure 2: FlowGuard System Architecture

To meet the above requirements, FlowGuard adopts a hybrid closed-loop architecture consisting of three stages, as shown in Figure 2. In the *programmable-switch stage*, a lightweight decision tree classifies flows at line rate and immediately mitigates detected attacks, ensuring line-rate processing and rapid response. Meanwhile, the switch aggregates per-flow features into compact per-window summaries rather than exporting individual flow records, reducing export overhead while retaining the traffic information needed for higher-level analysis. In the *window stage*, each exported summary is periodically analyzed by an LLM to determine whether an attack is occurring. For suspicious windows, the LLM further identifies the victim subnet and provides an explanation, while the analysis results are used to derive weak labels for model adaptation. Since this stage operates on accumulated window-level information rather than individual packets, it can run outside the latency-critical data-plane path. Finally, the *adaptation stage* monitors statistical drift in the observed traffic and uses an LLM to determine whether the deployed decision tree should be retrained. When retraining is warranted, FlowGuard trains a candidate tree using the newly derived weak labels together with the original labeled data and deploys it only after it passes a safety gate. The accepted tree is then deployed to the programmable switch, closing the loop between line-rate mitigation, window-level traffic analysis, and classifier adaptation.

### 3.3 DATA-PLANE STAGE: PER-FLOW CLASSIFICATION

The programmable-switch stage performs two tasks in parallel at line rate: per-flow classification for immediate mitigation and compact traffic monitoring for subsequent window-level analysis. By keeping both operations in the data plane, FlowGuard can react to detected attacks immediately without introducing additional processing delay into the mitigation path.

**Switch-compatible features.** FlowGuard adapts SISTAR's (Hu et al., 2025) feature set: destination port, initial TCP window, forward header length, mean packet length, and sending rate. To derive these flow-level statistics in the data plane, FlowGuard uses a fixed observation budget of $N = 4$ packets per flow. We choose $N$ as a power of two so that operations involving division by $N$, such as computing the mean packet length, can be implemented efficiently using a bit shift: mean packet length is obtained from the shifted packet-length sum, while elapsed time across the observed packets is used in place of sending rate. Flows shorter than $N$ are forwarded without a tree verdict but still contribute to the window digest. Appendix B gives the exact feature definitions and short-flow handling.

**Classifier.** FlowGuard uses a lightweight decision tree as the in-switch classifier for distinguishing benign and attack flows. The decision tree is first trained offline and is then compiled and deployed to the programmable switch using Leo (Jafri et al., 2024). Once deployed, the switch performs only inference: when the required flow features become available, the decision tree classifies the flow without controller involvement, and the resulting verdict is applied to its subsequent packets. Flows classified as benign are forwarded, whereas attack flows are dropped directly in the data plane for immediate mitigation. The deployed tree has a depth of $4$, and implementation details of the feature encoding and per-flow state management are provided in Appendix B.

**Compact window digest.** In parallel with per-flow classification, the programmable switch maintains a compact digest $D_t$ for each traffic window to summarize aggregate traffic behavior without exporting individual flow records. As packets traverse the data plane, the switch updates LogLog cardinality sketches (Durand & Flajolet, 2003; Ding et al., 2022) for sources, destinations, source–destination pairs, destination $/24$ prefixes, and flows, together with packet and byte counters $P_t$ and $B_t$ and a hash-selected sample of packet records. At the end of each window, the switch exports the resulting digest $D_t$ to the window stage for higher-level analysis and resets the corresponding state for the next window. Appendix B provides the implementation details.

### 3.4 WINDOW STAGE: LLM-BASED ATTACK ASSESSMENT

The window stage complements the per-flow classifier by assessing whether the aggregate traffic observed in each window indicates an ongoing attack. Rather than making per-flow decisions, this stage processes the compact digest exported by the switch and produces a window-level attack assessment together with auxiliary information for suspicious windows.

**Window-level traffic characterization.** Unlike the in-switch classifier, which makes decisions on individual flows, this stage determines whether an attack is occurring from the aggregate behavior of the traffic. Such behavior may not be evident from any single flow, but can emerge when multiple flows are observed collectively over a period of time. FlowGuard therefore analyzes traffic at the window level. At the end of each window, the compact digest $D_t$ is exported to the control plane, which derives five groups of aggregate traffic features: (i) *cardinality and volume*; (ii) *spread*; (iii) *flow structure*; (iv) *cross-server similarity*; and (v) *victim localization*. Together, these features characterize the aggregate traffic behavior within the window and provide the basis for the subsequent attack assessment. Appendix B provides their exact definitions.

**Temporal change characterization.** The aggregate features above characterize the current window but do not indicate how its traffic differs from recent observations. FlowGuard therefore augments them with temporal change scores. For $x_t \in \{\hat{n}_{\text{pair}}, \hat{n}_{\text{flow}}, P_t\}$ and a history $H$ containing the last $10$ admitted values, FlowGuard computes

$$z_t = \frac{x_t - \mu_H}{\max(\sigma_H, \epsilon\mu_H)}, \tag{1}$$

where $\epsilon = 0.05$ prevents small fluctuations from being amplified when the historical baseline is nearly constant. To prevent sustained traffic surges from being gradually absorbed into the baseline,

a value satisfying $z_t \geq 3$ and $x_t > 10\,\mu_H$ is withheld from $H$ for at most 300 windows, while decreases are always admitted. The resulting change scores provide the LLM with both the current traffic state and its deviation from recent traffic.

**LLM-based attack assessment.** FlowGuard serializes the window-level features and temporal change scores into a structured record and provides it to a prompted LLM without fine-tuning or labeled deployment data. In the *detection pass*, the LLM assigns the current window an attack likelihood in $\{\text{low}, \text{medium}, \text{high}\}$, and FlowGuard flags windows rated *high* as attack windows. For windows rated *medium* or *high*, an auxiliary *explanation pass* further identifies the likely victim subnet, summarizes the observed attack behavior, and reports the digest fields supporting the assessment. Since such explanations are not required for every window, FlowGuard generates them only for the first relevant window of an attack episode and every 30 windows thereafter while the episode persists.

### 3.5 Adaptation Stage: Drift-Triggered Classifier Adaptation

The adaptation stage updates the in-switch classifier as deployment traffic evolves. First, statistical drift detection identifies shifts in the features used by the tree, indicating a change in traffic distribution. For each detected drift, an LLM decides whether the change warrants retraining based on a drift report. Approved events are then weakly labeled using the window stage, and these labels are combined with the original ground-truth data to train a candidate tree. The new tree is deployed only if it passes a safety gate: it must improve balanced accuracy on the newly observed traffic and lose at most a small margin on the original ground-truth data.

**Drift detection.** Adaptation runs on a slower timescale than detection. Every $\tau_a$ seconds, the control plane builds a histogram of each feature used by the in-switch tree over the flows classified in the interval and, following FeDrift (Sharma & Tiwari, 2026), measures the normalized first Wasserstein distance between consecutive intervals. A feature drifts when this distance exceeds $\mu + k\sigma$ over its recent distances, and an interval drifts when any feature does. Sparse intervals are skipped and the standard deviation is floored to avoid unstable thresholds. Appendix D gives the full formulation and parameters.

**LLM retraining decision.** Distribution shift does not necessarily imply that the deployed tree has become inaccurate, and retraining too often can itself degrade performance. For each detected drift, we construct a report describing the affected features, drift magnitude and duration, and current traffic context. An LLM evaluates this report and returns a binary decision, with a one-sentence justification on whether retraining should be attempted. Thus, statistical drift detection identifies candidate changes, while the LLM filters them before any weakly supervised update is performed.

**Weak supervision and safety gate.** Because deployment traffic has no ground-truth labels, approved intervals are labeled (Ratner et al., 2017) using the window stage. In a *high* window, a flow receives $\tilde{y} = 1$ if its source or destination lies in the victim subnet named by the LLM and $\tilde{y} = 0$ otherwise. Flows in *low* windows receive $\tilde{y} = 0$, while *medium* windows provide no labels. Using the named victim subnet rather than labeling an entire flagged window avoids treating unrelated benign traffic as malicious. Across our runs, the admitted weak labels are 96–100% correct against ground truth (Appendix F).

Weak labels accumulate in the training pool across approved intervals, and retraining is attempted at an approved interval that contributes at least 100 weak labels. A random 30% of each approved interval's weak labels is held out as a validation gate set $\mathcal{V}$, and is not used to fit the candidate tree. The remaining weak labels are combined with the original ground truth records $G$, with unit weight for ground truth labels and weight $\lambda < 1$ for weak labels. The candidate tree $T'$ replaces the deployed tree $T$ only if $\mathcal{V}$ contains both classes and

$$\mathrm{BA}(T', \mathcal{V}) > \mathrm{BA}(T, \mathcal{V}) \quad \text{and} \quad \mathrm{BA}(T', G) \geq \mathrm{BA}(T, G) - \delta, \tag{2}$$

where BA denotes balanced accuracy. The first condition requires the candidate to improve on held-out newly observed traffic, while the second limits degradation on the original task. Balanced accuracy prevents the gate from favoring a candidate that simply predicts the benign majority.

# 4 EXPERIMENTS

We ask five questions. (1) Does the pipeline detect attacks that in-network classifiers miss under distribution shift? (2) What does the LLM add over a deterministic rule on the same digest? (3) Can the pipeline localize the victim subnet? (4) Which retraining policy adapts the in-switch tree safely? (5) Is LLM inference fast enough for deployment? Section 4.2 answers (1), (3), (4) and (5), and Section 4.3 answers (2) and examines what the digest must contain.

## 4.1 SETUP

**Dataset.** We train the in-switch classifiers on CIC-DDoS2019 (Sharafaldin et al., 2019) traffic and evaluate them on a stream that switches to IoT-23 (Garcia et al., 2020) botnet captures, each recorded at a single infected device, which introduces a distribution shift. During development, two components of the window digest were added after inspecting this evaluation stream, and the flagged-packet-share threshold that converts packet- and flow-level baselines to window verdicts was selected on it. No other setting was fitted to it. We then evaluate the frozen pipeline on a validation stream built from disjoint CIC-DDoS2019 attack captures and a later slice of the IoT-23 35-1 capture with floods against two other hosts. Stream construction, development choices, and window ground truth are given in Appendices C and D.

**Baselines.** We compare against SISTAR (Hu et al., 2025), our static in-switch tree, a deterministic threshold rule over the same window digest as the LLM, and a packet-volume threshold that flags a window whose packet-volume change score (Eq. (1)) is at least 3. This is the threshold of the digest's surge gate (§3.4) and was fixed before evaluation. SISTAR is reproduced from its public data-plane program and training code (Appendix D).

**Metrics.** We report window-level precision, recall, false-alarm rate, and $F_1$. Victim localization is scored by whether the reported subnet contains the target. Adaptation is evaluated by chronological test-then-train replay averaged over five seeds, and LLM cost by latency, tokens, and calls per hour.

**Implementation details.** Our data-plane program targets an Intel Tofino Wedge100BF-32X, with the decision tree compiled to match-action tables using Leo (Jafri et al., 2024). Detection accuracy is evaluated offline from packet captures, with digest statistics computed from all packets in each window rather than from sampled switch exports. Both LLM stages use a locally served uncensored derivative of Qwen3.8-27B Qwen Team (2026), specifically the HauhauCS Aggressive GGUF checkpoint HauhauCS (2026). Detection runs at temperature 0. Drift is evaluated every $\tau_a = 8$ s. All settings are in Appendix D.

## 4.2 RESULTS AND ANALYSIS

Table 1: Window-level detection on the evaluation stream (1,183 scored windows). SISTAR and the tree are converted to window verdicts at their best flagged-packet share $\theta$. Full validation results are in Table 4.

| Method | Native granularity | Precision | Recall | False alarm | $F_1$ |
|---|---|---|---|---|---|
| SISTAR (Hu et al., 2025) (retrained) | Packet | 0.517 | 0.963 | 0.846 | 0.673 |
| Ours: in-switch tree (static) | Flow | 0.997 | 0.637 | 0.002 | 0.777 |
| Threshold rule | Window | 0.804 | 0.949 | 0.218 | 0.870 |
| Volume threshold | Window | 0.926 | 1.000 | 0.075 | 0.961 |
| Ours: window stage (LLM) | Window | 0.939 | 0.998 | 0.061 | **0.968** |
| Ours: tree OR window stage | Flow + window | 0.938 | 1.000 | 0.062 | **0.968** |

**The window stage is the most robust detector across attack segments.** Once traffic shifts to IoT-23, the two in-network classifiers fail in different ways (Table 1). SISTAR keeps its recall only by lowering its threshold until it raises alarms in 85% of attack-free windows. Our tree stays precise and handles two of the three IoT-23 captures, but misses every attack window of 35-1, which caps its recall at 0.637.

Window-level detectors are close in aggregate but differ in their worst case. On the evaluation stream, the window stage and the volume threshold reach $F_1 = 0.968$ and 0.961, a difference of nine false alarms and one missed window. Across the six attack segments of both streams, however,

the window stage never falls below 0.86, whereas the volume threshold drops to 0.78 and every other detector to 0.23 or below on at least one segment (Figure 3, Table 6). This ordering also holds on the validation stream, where the window stage reaches $F_1 = 0.894$ at a false-alarm rate of 0.303, compared with 0.602 for SISTAR (Table 4).

With a single parameter, the volume threshold is easy to tune, but the tuned value does not carry over. Tuned on the evaluation stream ($z \geq 10$), it exceeds the window stage there with $F_1 = 0.995$. On the validation stream the best threshold moves to $z \geq 50$, and the evaluation-tuned value reaches 0.889 against 0.894 for the window stage, which uses no stream-specific threshold.

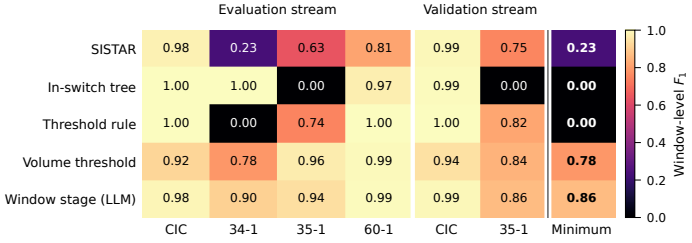



Figure 3: Window-level $F_1$ across evaluation and validation attack segments. CIC denotes the CIC-DDoS2019 NTP segment.

**The LLM localizes victims without a traffic-specific heuristic.**

For each of the 572 explained attack windows, the explanation pass names a victim subnet (Table 5). Because these attacks each target a single address, ranking prefixes by packet volume already finds the victim in 99.8% of these windows. The LLM matches this, naming a subnet containing the target in every window and the exact /24 in 98.1% of them (Figure 4). By contrast, the deterministic ranking of Appendix B, built for attacks spread across a subnet, reports a /24 only when it contains at least $\eta = 8$ destinations and therefore finds the target in only 14.2% of these single-target windows.

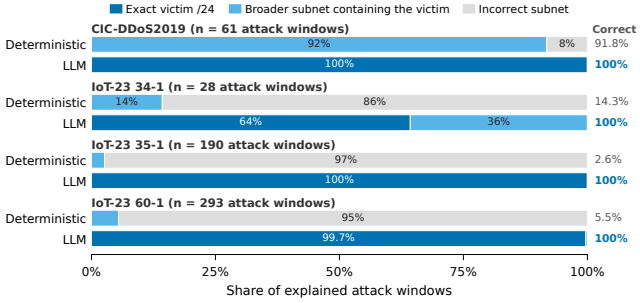
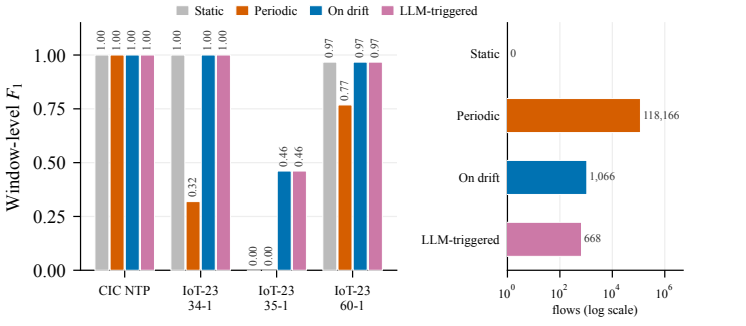



Figure 4: Victim subnet named per explained attack window.

**Gated adaptation is predictable where periodic retraining is not.** We replay the pipeline under five retraining policies that differ only in what triggers retraining: *never*, *periodic* (every 32 s), *on drift* (after every detected drift), *rule-triggered* (persistent drift in a feature used by the tree), and *LLM-triggered*. All policies share the same safety gate.

Under LLM-triggered adaptation, tree $F_1$ on 35-1 rises from 0.000 to 0.461 while the other three segments are unchanged (Figure 5). In the switch, the share of attack packets dropped rises from 61.5% to 69.2%, and the benign drop rate falls slightly (Figure 7). Pipeline $F_1$ stays at 0.968, since the window stage already covers the windows the tree misses.




(a) Tree window-level $F_1$ by segment (b) Weakly labelled flows absorbed

Figure 5: Retraining-policy effects on deployed-tree performance and weakly labeled traffic absorbed on the evaluation stream.

Under periodic retraining, the damage follows the number of weak labels absorbed. On both streams, every cadence that admits more than 50k weakly labeled flows lowers pipeline $F_1$, and every cadence at around 30k or fewer leaves it unchanged (Table 9). At the 32 s cadence on the evaluation stream, periodic retraining absorbs over 100k flows, degrades the 34-1 and 60-1 segments, and lowers pipeline $F_1$ to 0.805. Across the nine drift-monitor settings, no gated policy lowers pipeline $F_1$ by more than 0.002, whereas the periodic tree ranges from 0.291 to 0.959 (Table 10). Recovery of 35-1 itself occurs only at $\tau_a = 8$ s.

Filtering drift events with the LLM reaches the same tree and pipeline $F_1$ as retraining on every drift, on both streams, with fewer retraining attempts and over a third fewer weak labels on the evaluation stream. Rule-triggered retraining admits no candidate and behaves like the static tree (Table 8).

**Only full-digest detection exceeds the real-time budget.** At $1.47$ s per call, most of it spent on the prompt, a single sequential worker processes $0.68$ windows/s against an arrival rate of $0.93$ (Table 2). The minimal digest (§4.3) cuts detection to $0.54$ s and $1.85$ windows/s at a cost of $0.008$ $F_1$ on validation. Explanations take $14.9$ s but run only at the start of an attack episode and every 30 windows thereafter, and the victim prefix is available $4.2$ s after a window closes. Adaptation decisions take $3.5$ s on average, within the $8$ s drift interval.

Table 2: Language-model inference cost per call on one GPU with sequential requests. †Adaptation p95 is the slowest of 11 calls and includes model loading.

| Call | Invocation | Calls/hour | Mean latency (s) | p95 (s) | Total tokens/call |
|---|---|---|---|---|---|
| Detection (full digest) | Every window | 3,353 | 1.47 | 1.62 | 1,004 |
| Detection (minimal digest) | Every window | 3,353 | 0.54 | 0.61 | 439 |
| Explanation | Per attack episode | 139 | 14.9 | 18.3 | 1,389 |
| Adaptation decision | On detected drift | 17 | 3.5 | 10.8† | 775 |

### 4.3 Ablation study

**The LLM's advantage lies on traffic the rule was not calibrated for.** On identical digests, the deterministic rule is marginally better on the CIC-DDoS2019 traffic it was calibrated on, and the LLM is better on the shifted IoT-23 traffic in both streams (Figure 8). Their agreement falls from 98.6% of calibration windows to 86.1% of shifted windows on the evaluation stream, so disagreements concentrate where the rule's thresholds no longer apply.

**Additional digest features mainly add false alarms on the evaluation stream.** Three of the five reduced digests score higher $F_1$ than the full digest, with recall essentially unchanged and precision higher (Table 12). The exception is victim localization, whose removal lowers $F_1$ from $0.968$ to $0.923$. We keep the full digest because the explanation pass requires its prefix features, and the minimal digest is also slightly lower on validation ($0.886$ versus $0.894$).

### 4.4 Limitations of the evaluation

Each attack segment comes from a single source address and targets one or two hosts, so the spread, similarity, and victim-ranking components of the digest are not exercised. The surge hold excludes only windows exceeding ten times the baseline, so a sustained moderate increase is eventually absorbed. Weak labels inherit window-stage errors despite the safety gate. Sustaining the window rate with the full digest requires concurrent serving or the minimal digest (§4.2). A vantage point with higher flow churn needs more switch state. Finally, we evaluate only one LLM.

## 5 Conclusion and Future Work

**Conclusion** We presented a DDoS detection pipeline that combines an in-switch decision tree with LLM-based window analysis and drift-triggered adaptation. On traffic shifting from CIC-DDoS2019 to IoT-23, SISTAR transfers poorly and the tree misses 35-1, whereas the window stage detects 99.8% of attack windows at a 6.1% false-alarm rate on the evaluation stream and reaches $F_1 = 0.894$ on the held-out validation stream. It also localizes the victim subnet and is more robust than a calibrated threshold rule outside its calibration traffic. On the evaluation stream, selective adaptation improves the deployed tree from $F_1 = 0.777$ to $0.852$ while avoiding the degradation caused by frequent periodic retraining. **Future Work** Our evaluation does not contain subnet-wide attacks, and evaluating them is therefore the most important next step. The full-digest detection pass also does not yet keep pace with a one-second window on one GPU, motivating more compact inputs and faster inference. Finally, future work should validate detections directly and evaluate the entire pipeline end to end using live traffic and switch-exported digests.

## AI USE STATEMENT

Large language models were used only for language editing and code polishing, including grammar correction, stylistic polishing of the manuscript, debugging support, and improvements to code clarity and readability. They were not used for research ideation, experimental design, data analysis, or the generation of scientific content or technical contributions. All research decisions, analyses, and technical contributions are the authors' own.

## ETHICS STATEMENT

This work studies detection and adaptation for distributed denial-of-service (DDoS) traffic. All experiments use publicly available datasets, including CIC-DDoS2019 and IoT-23, and are conducted in a controlled test environment under our own administration. No experiments are performed against deployed third-party systems or real users, and the work does not involve human subjects or the collection of new personal data.

Although the paper studies adversarial network traffic, its purpose is defensive: to improve the detection of evolving and distributed attacks at programmable network switches. We report only the information necessary to reproduce and evaluate the defense, and do not introduce new attack techniques or deployment instructions intended to facilitate misuse.

## REPRODUCIBILITY STATEMENT

Our evaluation uses the publicly available CIC-DDoS2019 and IoT-23 datasets. The traffic replay procedure, baselines, evaluation metrics, and experimental protocol are described in Section 4.1. The same section reports the programmable-switch and LLM configuration, and the chronological data split. Table 3 in the appendix lists the complete experimental settings. The full window-level detection and drift-triggered adaptation algorithms are provided in the appendix. Adaptation results are averaged over five random seeds. The window stage is run once per stream: detection is deterministic (temperature $0$), while explanations are sampled at temperature $0.7$. The code and experiment configurations are released at `https://anonymous.4open.science/r/FlowGuard`.

## REFERENCES

Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the Mirai botnet. In *USENIX Security Symposium*, pp. 1093–1110, 2017.

Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM*, pp. 99–110, 2013.

Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44 (3):87–95, 2014.

Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv*, 2022.

Xiang Chen, Chunming Wu, Xuan Liu, Qun Huang, Dong Zhang, Haifeng Zhou, Qiang Yang, and Muhammad Khurram Khan. Empowering network security with programmable switches: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, pp. 1653–1704, 2023.

Damu Ding, Marco Savi, and Domenico Siracusa. Tracking normalized network traffic entropy to detect ddos attacks in p4. *IEEE Transactions on Dependable and Secure Computing*, pp. 4019–4031, 2022.

Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities. In *European Symposium on Algorithms (ESA)*, volume 2832 of *Lecture Notes in Computer Science*, pp. 605–617, 2003.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, 2014.

Sebastian Garcia, Agustin Parmisano, and Maria Jose Erquiaga. IoT-23: A labeled dataset with malicious and benign IoT network traffic. Zenodo, 2020.

Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. Aggregate-based congestion control for pulse-wave DDoS defense. In *ACM SIGCOMM*, pp. 693–706, 2022.

HauhauCS. Qwen3.8-27B-Uncensored-HauhauCS-Aggressive-MTP-GGUF. `https://huggingface.co/HauhauCS/Qwen3.8-27B-Uncensored-HauhauCS-Aggressive-MTP-GGUF`, 2026. Hugging Face model repository.

Junjie Hu, Feng Guo, Qihang Zhou, Yixin Zhang, Zibo Gao, Yinglong Han, and Zhiqiang Lv. Sistar: An efficient ddos detection and mitigation framework utilizing programmable data planes. In *ACM SIGSAC Conference on Computer and Communications Security*, 2025.

Syed Usman Jafri, Sanjay Rao, Vishal Shrivastav, and Mohit Tawarmalani. Leo: Online ML-based traffic classification at Multi-Terabit line rate. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2024.

Jong-hyouk Lee and Kamal Deep Singh. Switchtree: in-network computing and traffic analyses with random forests. *Neural Computing and Applications*, pp. 23143–23154, 2020.

Qingyang Li, Yihang Zhang, Zhidong Jia, Yannan Hu, Lei Zhang, Jianrong Zhang, Yongming Xu, Yong Cui, Zongming Guo, and Xinggong Zhang. Dollm: How large language models understanding network flow data to detect carpet bombing ddos. *arXiv*, 2024.

Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches. In *USENIX Security*, 2021.

Marek Majkowski. The Porcupine attack: Investigating millions of junk requests. Cloudflare Blog, January 2017. URL `https://blog.cloudflare.com/the-porcupine-attack-investigating-millions-of-junk-requests/`.

Junchen Pan, Jie Zhang, Lei Zhang, Xinggong Zhang, Xiaoyong Si, and Yong Cui. Netradar: Enabling robust carpet bombing ddos detection. In *Network and Distributed System Security Symposium (NDSS)*, 2026.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026. URL `https://qwen.ai/blog?id=qwen3.8`.

Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282, 2017.

Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A. Ghorbani. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *International Carnahan Conference on Security Technology (ICCST)*, pp. 1–8, 2019.

Amrendra Kumar Sharma and Mamta Tiwari. Fedrift: Feature-level drift evolution analysis for adaptive ddos detection in dynamic networks. *International Journal of Engineering Research & Technology*, 2026.

Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. NetLLM: Adapting large language models for networking. In *ACM SIGCOMM*, pp. 661–678, 2024.

Jiarong Xing, Wenqing Wu, and Ang Chen. Ripple: A programmable, decentralized Link-Flooding defense against adaptive adversaries. In *USENIX Security*, pp. 3865–3881, 2021.

Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. CADE: Detecting and explaining concept drift samples for security applications. In *USENIX Security Symposium*, pp. 2327–2344, 2021.

Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qi Li, Mingwei Xu, and Jianping Wu. Poseidon: Mitigating volumetric DDoS attacks with programmable switches. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

Qizheng Zhang, Ali Imran, Enkeleda Bardhi, Tushar Swamy, Nathan Zhang, Muhammad Shahbaz, and Kunle Olukotun. Caravan: Practical online learning of in-network ML models with labeling agents. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 325–345, 2024.

Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. An efficient design of intelligent network data plane. In *USENIX Security*, pp. 6203–6220, 2023a.

Huancheng Zhou, Sungmin Hong, Yangyang Liu, Xiapu Luo, Weichao Li, and Guofei Gu. Mew: Enabling large-scale and dynamic link-flooding defenses on programmable switches. In *IEEE Symposium on Security and Privacy*, pp. 3178–3192, 2023b.

## APPENDIX OVERVIEW

The appendix contains:

- **Section A:** the window-level detection and drift-triggered adaptation procedures.
- **Section B:** the switch architecture, digest features, and the deterministic baseline.
- **Section C:** construction of the evaluation and validation streams, and window ground truth.
- **Section D:** hardware, LLM configuration, experimental settings, and the SISTAR baseline.
- **Section E:** full validation results, per-segment detection, and victim localization.
- **Section F:** adaptation-policy, cadence, and sensitivity analyses.
- **Section G:** window-decision and digest ablations.

## A ALGORITHMS

Algorithm 1 summarizes the per-window detection loop, and Algorithm 2 gives the drift-triggered adaptation procedure. Retraining is performed after each approved drift interval, and the candidate is compared with the currently deployed tree. Appendix D gives the subsample bound $D_{\max}$ and the minimum distinct-target count $\eta$.

In Algorithm 1, $\hat{n}_\bullet$ denotes the window-level cardinality estimates, $P_t$ and $B_t$ the total packet and byte counts in window $t$, $F$ the sampled traffic records, $H_x$ the change-score history of statistic $x$, and $C$ the largest similarity neighborhood. In Algorithm 2, $\mathcal{H}^f$ stores recent drift scores for feature $f$, $\mathcal{F}_t$ denotes the features that drift in interval $t$, and $V$ the window-stage verdicts.

**Algorithm 1** Window-level detection (two passes)

**input:** digest $D_t = (\hat{n}_\bullet, P_t, B_t, F)$; change-score histories $H_x$
$S \leftarrow$ spread features from $\hat{n}_\bullet, P_t, B_t$; $G \leftarrow$ flow-structure features
**for** $x \in \{\hat{n}_{\text{pair}}, \hat{n}_{\text{flow}}, P_t\}$ **do**
    $z_x \leftarrow (x - \mu_{H_x}) / \max(\sigma_{H_x}, \epsilon\mu_{H_x})$ ▷ Eq. (1)
    **if** $z_x < 3$ **or** $x \leq 10\,\mu_{H_x}$ **or** surge held for 300 windows **then**
        append $x$ to $H_x$ ▷ a surge is kept out of the baseline
    **end if**
**end for**
$C \leftarrow$ LARGESTSIMILARITYNEIGHBORHOOD$(F, \theta, D_{\max})$
$R \leftarrow$ top /24 and /16 prefixes in $F$ with distinct destinations, flows, packets
$\phi(D_t) \leftarrow$ combine cardinality, volume, spread, flow structure, change scores, $C$, $R$
$v \leftarrow$ DETECT$(\phi(D_t)) \in \{\text{low}, \text{medium}, \text{high}\}$ ▷ LLM, one word
**if** $v \in \{\text{medium}, \text{high}\}$ **and** EXPLAINPOLICY$(t)$ **then**
    $(\pi, s, \text{shape}, \text{reasons}) \leftarrow$ EXPLAIN$(\phi(D_t), v)$ ▷ continues the conversation
    **report** victim subnet $\pi$ and explanation
**end if**
**return** verdict $v$

**Algorithm 2** Drift-triggered adaptation

**input:** feature histograms $h_t$, deployed tree $T$, ground-truth records $G$, window verdicts
$\mathcal{H}^f \leftarrow \langle\rangle$ for each feature $f$; $\text{driftCount} \leftarrow 0$
$\mathcal{P} \leftarrow \emptyset$; $\mathcal{V} \leftarrow \emptyset$ ▷ training pool and gate set of weak labels
**for** $t = 1, 2, \ldots$ **do**
    $\mathcal{F}_t \leftarrow \emptyset$
    **for** each feature $f$ **do**
        $\mathcal{W}_t^f \leftarrow$ WASSERSTEIN$(h_t^f, h_{t-1}^f)$ ▷ Eq. (4)
        **if** ISDRIFT$(\mathcal{W}_t^f, \mathcal{H}^f)$ **then** ▷ Eq. (5)
            $\mathcal{F}_t \leftarrow \mathcal{F}_t \cup \{f\}$
        **end if**
        update $\mathcal{H}^f$ with $\mathcal{W}_t^f$, retaining the last $L$ values
    **end for**
    **if** $\mathcal{F}_t = \emptyset$ **then**
        $\text{driftCount} \leftarrow 0$; **continue**
    **end if**
    $\text{driftCount} \leftarrow \text{driftCount} + 1$
    $\rho \leftarrow$ DRIFTREPORT$(\mathcal{F}_t, \{\mathcal{W}_t^f\}, \text{driftCount}, \text{traffic context})$
    **if** DECIDE$(\rho)$ **then** ▷ LLM
        $R_t \leftarrow$ flows classified in interval $t$ that the window stage labels (§3.5)
        $\tilde{y}_t \leftarrow 1$ if the flow meets the victim prefix named for its window, else 0
        split $(R_t, \tilde{y}_t)$ 70/30; add the parts to the training pool $\mathcal{P}$ and the gate set $\mathcal{V}$
        **if** $|R_t| \geq 100$ **then**
            $T' \leftarrow \text{FIT}\big(G \cup \mathcal{P}, \lambda\big)$
            **if** $\mathcal{V}$ holds both classes **and** $\text{BA}(T', \mathcal{V}) > \text{BA}(T, \mathcal{V})$ **and** $\text{BA}(T', G) \geq \text{BA}(T, G) - \delta$ **then** ▷ $\mathcal{V}$: held-out weak labels; $G$: ground truth
                $T \leftarrow T'$ ▷ Eq. (2)
            **end if**
        **end if**
    **end if**
**end for**

## B SWITCH ARCHITECTURE AND DIGEST DETAILS

**Sketches and sampling.** Following P4LogLog (Ding et al., 2022), cardinalities are maintained as LogLog register arrays in the data plane and read by the control plane at window boundaries using double buffering. Packet records are sampled at roughly $1/2^{10}$ by hashing the flow key and packet timestamp, making the sampling packet-level rather than flow-level. Each record contains source and destination addresses, protocol, ports, and packet length. Evaluation uses all packets to compute digest statistics and therefore does not measure sampling error (§4.1).

**Flow features.** For a flow $f$ with packets $p_1, p_2, \ldots$ of lengths $\ell_i$ and packet timestamps $t_i$, the features are computed over the first $n = \min(|f|, N)$ observed packets:

$$\mathbf{x}_f = \Big(\mathrm{dport}(p_1),\ \mathrm{win}(p_1),\ \sum_{\substack{i \le n \\ p_i \text{ fwd}}} h_i,\ \Big(\sum_{i=1}^{n} \ell_i\Big) \gg \log_2 N,\ t_n - t_1\Big), \tag{3}$$

where $\mathrm{win}(p_1)$ is the TCP window of the first forward packet ($-1$ for non-TCP flows), $h_i$ the transport-header length of forward packet $p_i$, and $\gg$ a right shift. Right-shifting replaces division when computing the mean packet length. In place of sending rate, the tree uses the elapsed time $t_n - t_1$ across these packets, which contains equivalent information since $N$ is fixed.

**Per-flow state.** Computing Eq. (3) requires state for the first $N$ packets of each flow. This state is held in a fixed table of $2^{16}$ slots storing a flow signature, first-packet timestamp, packet count, length and header sums, and the verdict. A symmetric hash of the 5-tuple maps both directions of a flow to the same slot, while a second hash provides the signature. A slot is reused on a signature mismatch or after 120 s of inactivity, matching the flow timeout used in training. Memory use is therefore constant, at the cost of possible collision eviction. In a packet-level simulation over 2 M training packets, only $0.006\%$ of flows lost their slot before classification. All others reproduced the offline features exactly, except for duration, which the switch measures in $1.024\,\mu$s clock units.

**Flows shorter than $N$ packets.** A flow that ends before its $N$-th packet is not classified by the in-switch tree and is therefore forwarded without a verdict. Its features are still computed offline for training and flow-level evaluation with Eq. (3), so the packet-length sum is divided by $N$ rather than by the observed packet count $n$. Little traffic is affected. Such flows constitute $2.4\%$ and $3.4\%$ of flows on the evaluation and validation streams, respectively, but only $0.07\%$ and $0.11\%$ of packets and $0.06\%$ and $0.10\%$ of attack packets.

Because these flows receive no tree verdict, they contribute no flagged packets to the tree's window-level alarms. Their packets do, however, enter the window digest and remain visible to the window stage. Attacks made mostly of very short flows, such as single-packet floods, would therefore escape the tree. Mitigating them in the switch would need a smaller $N$ or classification at flow end.

**Feature definitions.** From the cardinality estimates $\hat{n}_{\mathrm{src}}$, $\hat{n}_{\mathrm{dst}}$, $\hat{n}_{\mathrm{pair}}$, $\hat{n}_{\mathrm{pre}}$ and $\hat{n}_{\mathrm{flow}}$ and the counters $P_t$ and $B_t$, the spread features are $\hat{n}_{\mathrm{pair}}/\hat{n}_{\mathrm{dst}}$, $\hat{n}_{\mathrm{dst}}/\hat{n}_{\mathrm{pre}}$, $B_t/\hat{n}_{\mathrm{pair}}$ and $P_t/\hat{n}_{\mathrm{pair}}$, and the flow-structure features are $\hat{n}_{\mathrm{flow}}$, $P_t/\hat{n}_{\mathrm{flow}}$, $B_t/P_t$, and the share of sampled packets addressed to the most frequent destination. Ratios are computed in the control plane, where division is available. For cross-server similarity, each sampled destination $d$ is represented by $\big(\log(1+\mathrm{thr}_d), \bar{\ell}_d\big)$, its throughput and mean packet length, standardized across destinations and normalized to a unit vector $u_d$. With $\mathcal{N}(d) = \{d' : \langle u_d, u_{d'}\rangle \ge \theta\}$, we take $c = \arg\max_d |\mathcal{N}(d)|$ and report $|\mathcal{N}(c)|$, its fraction of observed destinations, its mean pairwise similarity, and its mean throughput, packet length and approximate packet spacing. Windows with more than $D_{\max}$ destinations are subsampled. For victim localization, sampled destinations are grouped into $/24$ and $/16$ prefixes, and each prefix has the number of distinct destinations reached and its share of sampled traffic. The deterministic ranking reports the top $/24$ if it holds at least $\eta$ distinct destinations, else the top $/16$.

**Switch encoding.** Leo evaluates a comparison as the sign bit of a 16-bit sum, so every feature value and threshold must lie in $[0, 32767]$. We therefore halve the destination port, shift the TCP window by one so that the non-TCP sentinel $-1$ maps to 0, and quantize the duration to units of

$2^{20}$ ns. Trees are trained in the units of Eq. (3) and their thresholds mapped onto this encoding at compile time, which moves duration boundaries by less than half a quantum. The compiled tree agrees with the offline tree on all training and evaluation-stream flows, so no prediction changes.

**Deterministic baseline.** The threshold rule flags a window if either of two conditions holds. The *spread* condition requires more than 100 source–destination pairs and either fewer than 5 packets per pair or at least 60% of destinations in the largest similarity neighborhood. The *surge* condition requires the flow count or packet volume to exceed its 99.9th percentile over attack-free training windows, with at least half of the packets addressed to one destination.

**Explanation episodes and drift history.** Windows rated *high* or *medium* that are at most two windows apart belong to the same attack episode. Drift testing begins once three distances have accumulated for a feature.

## C Evaluation and validation streams

**Evaluation stream.** The CIC-DDoS2019 portion uses five minutes of benign traffic followed by NTP capture files 1–30 in original timing. The IoT-23 portion follows 1 s later and concatenates 240 s of capture 34-1 (Mirai), 433 s of 60-1 (Gafgyt), and 440 s of 35-1 (Mirai), with each device's background and scanning traffic retained. Benign Philips Hue and Amazon Echo honeypot traffic is overlaid throughout. Training uses the preceding benign traffic and NTP files 31–60.

**Window ground truth.** Attack periods are measured from the labeled connections, or for 35-1 from the packets addressed to the victim, and extended through the contiguous windows that still carry attack-labeled traffic. Figure 6 shows the window counts of both streams.

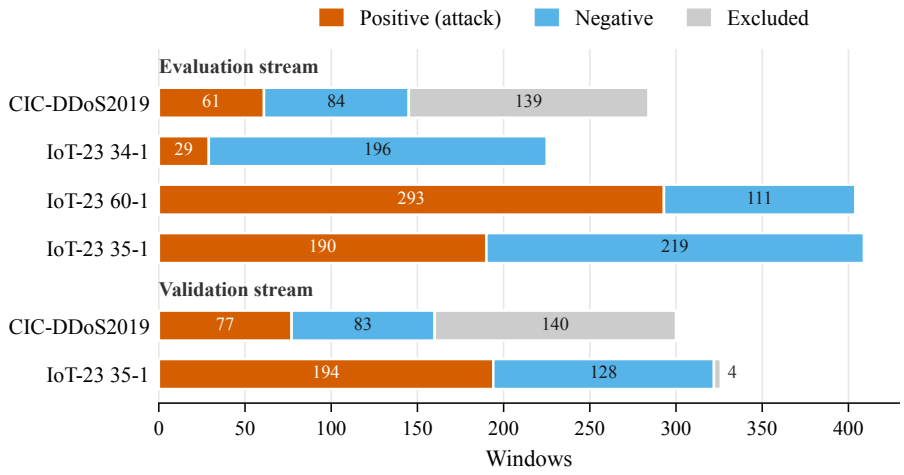



Figure 6: Window categories per segment of both streams ($\tau = 2^{30}$ ns). Totals are 573 positive, 610 negative and 139 excluded windows on the evaluation stream, and 271, 211 and 144 on the validation stream.

**Validation stream.** The CIC-DDoS2019 portion uses NTP capture files 61–90 after the same benign slice, and the IoT-23 portion uses 350 s of capture 35-1 with floods against two other hosts. The honeypot overlay has the same density but starts slightly later in the honeypot captures than in the evaluation stream. The stream has 626 windows (Figure 6).

## D Implementation details

**Switch implementation.** Our data-plane program targets an Intel Tofino Wedge100BF-32X using the Tofino Native Architecture, and Leo (Jafri et al., 2024) compiles the tree to match-action tables.

**Language-model configuration.** Both LLM stages use Qwen3.8-27B-Uncensored with `Q8_0` quantization (Ollama artifact ID `6a5fae387fd7`). Ollama v0.33.3 serves the model locally inside an NVIDIA-enabled Docker container on a single NVIDIA A40 GPU, with reasoning disabled during inference and with no task-specific fine-tuning. Decoding parameters are reported in Table 3.

**Drift detection.** With $F^f_{t,k}$ the cumulative distribution of feature $f$ up to bin $k$, $\Delta e_k$ the bin width and $[e_0, e_K]$ the histogram range, the shift between consecutive intervals is

$$\mathcal{W}^f_t = \frac{1}{e_K - e_0} \sum_{k=1}^{K-1} \left| F^f_{t,k} - F^f_{t-1,k} \right| \Delta e_k, \tag{4}$$

and feature $f$ drifts when

$$\mathcal{W}^f_t > \mu(\mathcal{H}^f) + k\,\sigma(\mathcal{H}^f), \tag{5}$$

where $\mathcal{H}^f$ contains its previous $L$ distances.

**Experimental settings.** Table 3 summarizes the parameters used throughout the experiments.

Table 3: Settings used in all experiments.

| Component | Setting | Value |
|---|---|---|
| In-switch tree | Packets per feature vector $N$; tree depth | 4; 4 |
| Change scores | History length; $\epsilon$; surge hold | 10 windows; 0.05; at most 300 windows |
| Similarity | Threshold $\theta$; destination subsample bound $D_{\max}$ | 0.9; 1,500 |
| Victim ranking | Minimum distinct destinations in a /24, $\eta$ | 8 |
| Threshold rule | Surge thresholds (99.9th pct. of 1,371 attack-free windows) | 192 flows; 3,464 packets |
| Explanations | Episode gap; re-explanation interval | 2 windows; 30 windows |
| Drift detection | $\tau_a$; $k$; history $L$ | 8 s; 3; 8 intervals |
| | Minimum flows per compared interval; $\sigma$ floor | 50; 0.005 |
| Retraining | Minimum weak labels; weak-label weight $\lambda$ | 100; 0.3 |
| Safety gate | Held-out share; tolerance $\delta$ | 30%; 0.01 |
| Periodic policy | Retraining interval | every 4 intervals (32 s) |
| LLM | Detection: temperature, max tokens | 0, 5 |
| | Explanation: temperature, max tokens | 0.7 (fixed seed), 400 |

**Development choices.** Two components of the window digest were selected while inspecting the evaluation stream: the packet-volume change score and the ratio condition used by the digest's surge-hold gate. A rolling baseline can absorb a sustained flood within a few windows, so sufficiently large surges are withheld from the history. However, a $z$-score condition alone can also withhold ordinary bursty traffic, such as a new device appearing, when the historical standard deviation is small. The ratio condition separates the two by also requiring a large rise over the historical mean.

Packet- and flow-level baselines are converted to window verdicts by flagging a window when the share of packets they flag is at least $\theta$. We give each baseline its best $\theta$ on the evaluation stream, with $\theta = 0$ for SISTAR and $\theta = 0.5$ for our in-switch tree, and reuse these values unchanged on the validation stream. The window stage receives no comparable tuning.

No other setting was selected or tuned on the evaluation stream. Surge thresholds of the deterministic rule are calibrated on benign training windows only. We set $\tau_a$ from operational considerations rather than a parameter sweep, and the volume baseline reuses the gate threshold of Eq. (1). We constructed the validation stream after these refinements and evaluated it with all settings frozen.

**SISTAR baseline.** Where the released code departs from the paper, we make four corrections: we enable UDP parsing, correct the one-second timestamp comparison, fix the running-mean update, and update the header-length register directly. The placeholder installation rules are unused.

We train with SISTAR's unmodified DT-CTS code at depth 4 on all 195,044 benign training packets and an equal-size uniform sample of the 12,149,355 attack packets, with features computed over the full ordered traffic before sampling. On a 30% stratified hold-out the tree reaches $F_1 = 0.998$, and its compiled switch form reproduces its predictions on every training sample.

## E ADDITIONAL DETECTION RESULTS

Table 4 gives the full detection results on the validation stream, Table 6 gives the per-segment values plotted in Figure 3, and Table 5 reports victim localization.

Table 4: Window-level detection on the held-out validation stream (482 scored windows, a later slice of the 35-1 capture against two other hosts, all settings fixed on the evaluation stream).

| Method | Precision | Recall | False alarm | $F_1$ |
|---|---|---|---|---|
| SISTAR (retrained) | 0.678 | 0.985 | 0.602 | 0.803 |
| Ours: in-switch tree (static) | 0.987 | 0.280 | 0.005 | 0.437 |
| Threshold rule | 0.764 | 0.989 | 0.393 | 0.862 |
| Volume threshold | 0.766 | 1.000 | 0.393 | 0.867 |
| Ours: window stage (LLM) | 0.809 | 1.000 | 0.303 | **0.894** |
| Ours: tree OR window stage | 0.807 | 1.000 | 0.308 | 0.893 |

Table 5: Victim-subnet identification over the 572 explained attack windows of the evaluation stream: fraction of windows whose named subnet contains the target address, and fraction naming the target's exact $/24$. Ranking prefixes by packet volume is included as a heuristic baseline that suits single-target floods.

| | Contains target | | | | | |
|---|---|---|---|---|---|---|
| Method | CIC | 34-1 | 35-1 | 60-1 | All | Exact $/24$ |
| Deterministic ranking | 0.918 | 0.143 | 0.026 | 0.055 | 0.142 | 0.000 |
| Most packets $/24$ | 1.000 | 1.000 | 1.000 | 0.997 | 0.998 | 0.998 |
| LLM (explanation pass) | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 0.981 |

Table 6: Window-level $F_1$ per segment of both streams: the values plotted in Figure 3. CIC denotes the CIC-DDoS2019 NTP segment, and the last column is the minimum over all six segments.

| | Evaluation stream | | | | Validation stream | | |
|---|---|---|---|---|---|---|---|
| Method | CIC | 34-1 | 35-1 | 60-1 | CIC | 35-1 | Minimum |
| SISTAR (retrained) | 0.976 | 0.230 | 0.634 | 0.811 | 0.987 | 0.747 | 0.230 |
| Ours: in-switch tree (static) | 1.000 | 1.000 | 0.000 | 0.967 | 0.993 | 0.000 | 0.000 |
| Threshold rule | 1.000 | 0.000 | 0.742 | 0.998 | 1.000 | 0.816 | 0.000 |
| Volume threshold | 0.917 | 0.784 | 0.964 | 0.992 | 0.939 | 0.842 | 0.784 |
| Ours: window stage (LLM) | 0.984 | 0.903 | 0.936 | 0.993 | 0.987 | 0.862 | **0.862** |

## F ADAPTATION STUDY

All results are means over five seeds of the time-ordered replay (§4.1). Tree and pipeline columns report window-level $F_1$.

**Per-segment effect of each policy.** Table 7 gives the values in Figure 5 plus the rule-triggered policy.

**Retraining policies across both streams.** Table 8 gives the aggregate scores behind Table 7. The weak labels every policy admits are 96–100% correct against ground truth, so the harm of periodic retraining does not come from label noise.

**The damage from periodic retraining follows the volume of weak labels absorbed.** Table 9 varies the periodic retraining cadence. Gated policies admit fewer than 1.5k weak labels (Table 8), far below the 50k at which periodic retraining starts to lower pipeline $F_1$.

**Gated policies barely change pipeline $F_1$, but their benefit depends on the monitor interval.** Table 10 varies $\tau_a \in \{4, 8, 16\}$ s and $k \in \{2, 2.5, 3\}$.

Table 7: What each retraining policy does to the deployed tree, per segment of the evaluation stream (window $F_1$, mean over five seeds). The window stage on the same windows is shown for contrast, it is not a retraining policy. Retraining counts, weak labels, honeypot false alarms and pipeline $F_1$ are in Table 8.

| | Deployed tree, window $F_1$ per segment | | | |
|---|---|---|---|---|
| Policy | CIC | 34-1 | 35-1 | 60-1 |
| Never (static) | 1.000 | 1.000 | 0.000 | 0.967 |
| Periodic | 1.000 | 0.319 | 0.000 | 0.768 |
| On drift | 1.000 | 1.000 | **0.461** | 0.967 |
| Rule-triggered | 1.000 | 1.000 | 0.000 | 0.967 |
| LLM-triggered (ours) | 1.000 | 1.000 | **0.461** | 0.967 |
| *Window stage (LLM)* | 0.984 | 0.903 | 0.936 | 0.993 |

Table 8: Retraining policies, mean over five seeds. *Accepted* counts candidates that pass the safety gate (fractional because the gate outcome varies with the seed). *Honeypot FA* is the share of benign honeypot flows the deployed tree flags.

| | | | | Deployed tree | | Pipeline | |
|---|---|---|---|---|---|---|---|
| Policy | Retrains | Accepted | Weak labels | $F_1$ | Honeypot FA | $F_1$ | False alarm |
| *Evaluation stream* (7 drift intervals) | | | | | | | |
| Never (static) | 0 | 0.0 | 0 | 0.777 | 0.159 | **0.968** | 0.062 |
| Periodic | 17 | 4.4 | 118,166 | 0.575 | 0.096 | 0.805 | 0.454 |
| On drift | 3 | 1.6 | 1,066 | **0.852** | 0.101 | **0.968** | 0.062 |
| Rule-triggered | 1 | 0.0 | 116 | 0.777 | 0.159 | **0.968** | 0.062 |
| LLM-triggered (ours) | 2 | 0.8 | 668 | **0.852** | 0.118 | **0.968** | 0.062 |
| *Validation stream* (4 drift intervals) | | | | | | | |
| Never (static) | 0 | 0.0 | 0 | 0.437 | 0.143 | **0.893** | 0.308 |
| Periodic | 11 | 6.8 | 51,200 | 0.431 | 0.243 | 0.870 | 0.384 |
| On drift | 2 | 0.4 | 1,488 | **0.442** | 0.125 | 0.891 | 0.314 |
| Rule-triggered | 0 | 0.0 | 0 | 0.437 | 0.143 | **0.893** | 0.308 |
| LLM-triggered (ours) | 1 | 0.4 | 793 | **0.442** | 0.125 | 0.891 | 0.314 |

Table 9: The periodic policy at four cadences. Never retraining and the gated policies are in Table 8.

| | Evaluation stream | | | | Validation stream | | | |
|---|---|---|---|---|---|---|---|---|
| Retrain every | Retrains | Weak labels | Tree | Pipeline | Retrains | Weak labels | Tree | Pipeline |
| 16 s | 34 | 208,657 | 0.620 | 0.805 | 23 | 188,549 | 0.511 | 0.877 |
| 32 s | 17 | 118,166 | 0.575 | 0.805 | 11 | 51,200 | 0.432 | 0.870 |
| 64 s | 8 | 30,978 | 0.852 | **0.968** | 6 | 28,011 | 0.437 | **0.893** |
| 128 s | 4 | 5,614 | 0.814 | **0.968** | 3 | 16,947 | 0.437 | **0.893** |

Table 10: Sensitivity to the drift monitor: deployed tree / pipeline $F_1$ per interval length, over $k \in \{2, 2.5, 3\}$ (ranges where the value depends on $k$). Never retraining gives $0.777$ / $0.968$ on the evaluation stream and $0.437$ / $0.893$ on the validation stream.

| Stream | $\tau_a$ | Drift events | Periodic | On drift | LLM-triggered |
|---|---|---|---|---|---|
| Evaluation | 4 s | 5–8 | **0.959** / 0.968 | 0.777 / 0.968 | 0.777 / 0.968 |
| | 8 s | 7–10 | 0.575 / 0.805 | 0.852–0.870 / 0.968 | 0.852 / 0.968 |
| | 16 s | 3–5 | 0.291 / 0.847 | 0.777 / 0.968 | 0.777 / 0.968 |
| Validation | 4 s | 3–9 | 0.437 / 0.893 | 0.437–0.447 / 0.893 | 0.437–0.440 / 0.893 |
| | 8 s | 4–8 | 0.431 / 0.870 | 0.442 / 0.891 | 0.442 / 0.891 |
| | 16 s | 0–1 | **0.798** / 0.872 | 0.437 / 0.893 | 0.437 / 0.893 |

The 35-1 blind spot is recovered only at $\tau_a = 8\,\text{s}$, for two different reasons. At $4\,\text{s}$ an interval contributes few weak labels, and a candidate then ties the deployed tree on the small gate set (for example $1.000$ against $1.000$), which the strict inequality in Eq. (2) rejects. At $16\,\text{s}$ candidates are accepted, but the training pool is by then dominated by the preceding 60-1 interval ($3{,}972$ weak labels against $596$ from 35-1, compared with $116$ against $552$ at $8\,\text{s}$), and the retrained tree no longer flags the 35-1 flood.

**What the switch drops.** Figure 7 reports packet-level mitigation: the share of attack packets dropped in the data plane and the share of benign packets dropped with them, over all scored flows. A flagged flow is dropped only from the decision packet onward, so even a correctly classified flow is not dropped in full. On the validation stream the gain from gated retraining is small, and periodic retraining roughly quadruples benign drops on both streams.

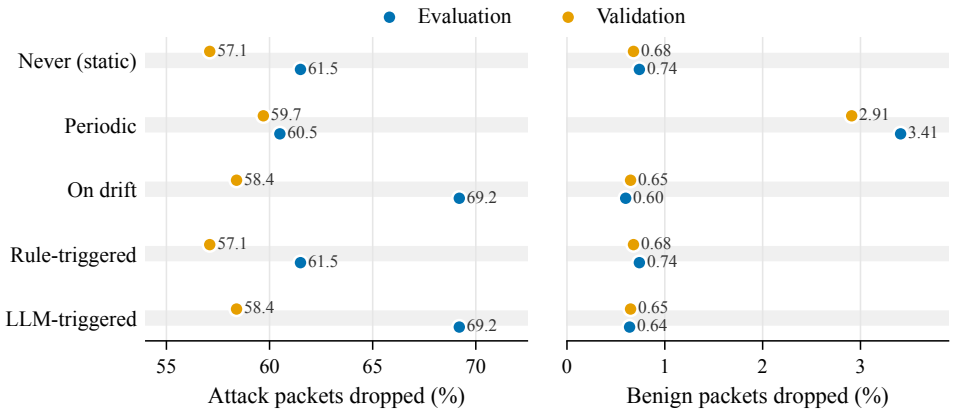



Figure 7: Packets dropped in the switch under each retraining policy, mean over five seeds.

**LLM decisions align with the helpful retraining events.** Five drift events admit enough weak labels to train a tree independently (Table 11). Two improve tree $F_1$ by more than $0.005$ without comparable pipeline degradation, and the LLM requests retraining for both. The deterministic rule requests neither. Because only five events can be judged this way, we treat this as a case analysis rather than a statistical result.

Table 11: Quality of individual retraining decisions. *Requested* counts requests over all and over qualifying drift events; *Correct* scores decisions only on qualifying events, where a retraining counterfactual exists.

| | Evaluation stream | | | Validation stream | | |
|---|---|---|---|---|---|---|
| | Requested | | | Requested | | |
| Decision procedure | All | Qualifying | Correct | All | Qualifying | Correct |
| Never (static) | 0 | 0/3 | 2/3 | 0 | 0/2 | 1/2 |
| On drift | 7 | 3/3 | 1/3 | 4 | 2/2 | 1/2 |
| Rule-triggered | 1 | 1/3 | 1/3 | 0 | 0/2 | 1/2 |
| LLM-triggered (ours) | 3 | 2/3 | **2/3** | 1 | 1/2 | **2/2** |

## G Ablation results

**LLM versus deterministic decision rule.** Figure 8 compares the LLM and deterministic threshold rule using identical window digests. We calibrate the rule on CIC-DDoS2019 training traffic and apply it to IoT-23 without recalibration.

**Digest representation.** Table 12 ablates feature groups from the detection-pass input while holding the prompt and decision procedure fixed. Cutting the digest to cardinalities and volume reduces tokens and latency by more than half.

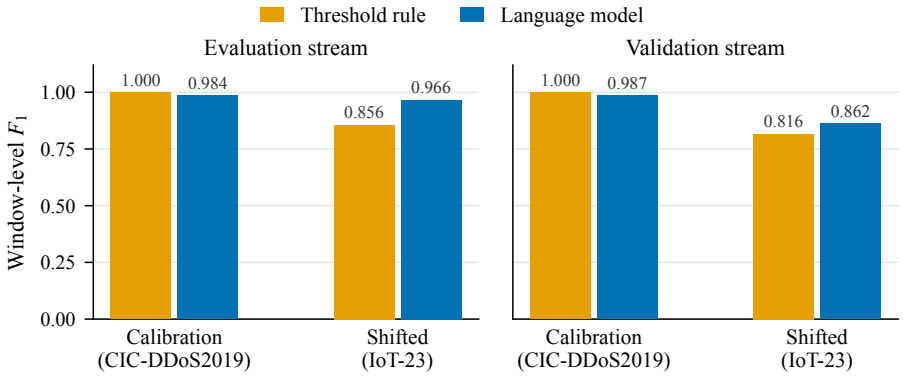



Figure 8: Window-level $F_1$ of the LLM and deterministic rule on calibration and shifted traffic.

Table 12: Digest ablation on the evaluation stream. Tokens and latency are per call.

| Input representation | Precision | Recall | $F_1$ | Prompt tokens | Latency (s) |
|---|---|---|---|---|---|
| Full digest | 0.939 | 0.998 | 0.968 | 1,002 | 1.47 |
| without flow structure | 0.960 | 0.997 | 0.978 | 941 | 1.32 |
| without cross-server similarity | 0.966 | 0.998 | 0.982 | 920 | 1.32 |
| without victim-localization features | 0.858 | 0.998 | 0.923 | 652 | 0.68 |
| Cardinalities, volume and spread | 0.966 | 0.955 | 0.960 | 508 | 0.57 |
| Cardinalities and volume only | 0.985 | 1.000 | 0.992 | 437 | 0.54 |