Harnessing Accurate and Automatic Trend Detection in Data Streams via Tbps-Level Inference

Yuhan Wu[1]*, **Qianxun Xu**[1]*, **Lida Liao**[1], **Longlong Zhu**[2], **Jiashuo Yu**[2]
Linying Zheng[2], **Hongyan Liu**[1], **Dong Zhang**[1]†, **Chunming Wu**[2], **Xiang Chen**[2]†
[1]Fuzhou University, Fuzhou, Fujian, China
[2]Zhejiang University, Hangzhou, Zhejiang, China
{102301320,832303230,241020021,zhangdong}@fzu.edu.cn
{hyliu20,wuchunming}@zju.edu.cn
{kikionepin,joshuayu.2048,zhenglinying0106,wasdnsxchen}@gmail.com

Abstract

Detecting frequency trend patterns, such as items with sustained growth or decline, is an important and challenging task in high-speed data streams. State-of-the-art solutions require manual tuning of many coupled parameters, while their bloom filters suffer from high false positive rates. In this paper, we propose NeuTrend, a framework that provides accurate and automatic trend detection with high-performance in-network inference. Our key idea is that a lightweight BRNN can learn to classify incoming items as likely trending or non-trending. Thus, it eliminates unqualified items more accurately than a bloom filter and removes the need for manual parameter tuning. The BRNN uses only XNOR and popcount operations. Thus, it aligns with the strict resource constraints of data-plane switches. NeuTrend further introduces an adaptive detection module. The module uses the BRNN confidence scores to automatically tune the growth and decay thresholds. Extensive experiments on real-world datasets show that the learned filter reduces false positive rates by 23%-44% and increases effective detector occupancy from 47% to 70%. Hence, NeuTrend improves the overall F1 score by 35%-68% over existing solutions, and achieves Tbps-level line-rate processing.

1 Introduction

Frequency estimation and pattern detection in data streams are fundamental tasks in network measurement [30, 23, 8], social network analysis [33], and cache management [14]. Traditional measurement tasks focus on identifying heavy hitters [7, 29] or estimating per-item frequencies [6], while a more recent direction studies the change of item frequencies over multiple consecutive time windows. In detail, this direction aims to measure two types of items, i.e., *promising items* and *damping items*. A promising item is one whose frequency shows sustained growth over multiple windows, which may indicate a possible DDoS attack [25], a new trending topic [33], or a new network congestion event [22]. Similarly, a damping item shows sustained frequency decay, which may indicate a topic that is losing attention or a service with declining demand.

However, existing solutions for trend detection in data streams still suffer from three limitations that prevent practical deployment and limit detection accuracy. (1) They require manual tuning of many coupled parameters, which places a heavy burden on users. For instance, the state-of-the-art solution Scout Sketch+ [13] uses a filter-detector architecture and requires tuning at least eight parameters: the growth threshold G, the decay threshold D, the maximum legal period length ρ_1, the minimum

*Equal contribution.
†Corresponding authors: Dong Zhang and Xiang Chen.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

number of legal periods ρ_2, the frequency thresholds ρ_3 and ρ_6, the protection period ρ_4, and the replacement threshold ρ_5. These parameters interact in complex ways. For example, a large G with a small D results in overly strict detection, while a small G with a large D generates many false positives. Also, existing solutions lack automated mechanisms to tune these parameters, while the optimal settings vary across traffic patterns and applications. (2) Existing solutions suffer from high false positive rates since they rely on Bloom filters. For example, Scout Sketch+ uses a bloom filter to admit items into its detector when the estimated frequency reaches $\rho_3 - 1$. However, hash collisions in the bloom filter cause non-trending items to be incorrectly admitted, wasting resources and reducing detection accuracy. This problem becomes more severe under small memory budgets, where the bloom filter collision rate increases. (3) Existing solutions are primarily designed for software execution, while line-rate processing is essential for high-speed network monitoring.

To fill this gap, we propose NeuTrend, a framework that provides accurate, automatic, and high-performance trend detection in data streams via Tbps-level in-network inference. Our observation is that recent advances in in-network machine learning enable lightweight neural networks to be deployed on data-plane switches for line-rate inference. Based on this, NeuTrend adopts two key ideas. First, NeuTrend replaces the static bloom filter with a lightweight binarized recurrent neural network (BRNN) as a learned filter while the BRNN classifies incoming items as likely trending or non-trending based on packet-level features. Second, NeuTrend uses the BRNN confidence scores to adaptively adjust the detection thresholds. Thus, manual tuning is no longer needed. Also, the BRNN runs on XNOR and popcount operations. As such, its inference can be offloaded onto the hardware data plane of data plane switches for line-rate detection. Main contributions are as follows.

1. We propose NeuTrend, a framework that introduces a BRNN-based learned filter and an adaptive threshold module for trend detection in data streams. In detail, the learned filter classifies incoming items as likely trending or non-trending with higher accuracy than traditional bloom filters, while the adaptive threshold module sets all detection parameters based on BRNN confidence scores, reducing manual tuning to only two robust bound ranges.
2. We prove two properties of NeuTrend. First, the learned filter achieves a lower false admission rate for non-trending items than the bloom filter when the BRNN classification accuracy is sufficient (Theorem 2). Second, the end-to-end F1 score is monotonically non-increasing in the BRNN false positive rate (Theorem 4).
3. We achieve high-performance, line-rate detection by offloading BRNN inference to the ASIC pipelines of data-plane switches via XNOR and popcount operations. Extensive experiments on four real-world datasets show that NeuTrend improves the F1 score by $1.35\times$-$1.68\times$ over existing solutions while operating at the Tbps-level line rate on Intel Tofino switches.

2 Related Work

Trend and pattern detection in data streams. Traditional sketch-based measurement focuses on frequency estimation [7, 6, 28], heavy hitter detection [30, 29], and heavy change detection [20]. Recent works extend sketch structures to detect time-varying patterns. In detail, BurstSketch [34] detects items with sudden frequency changes between adjacent windows, while PeriodicSketch [11] identifies items arriving at fixed intervals. SteadySketch [17] finds flows with stable arrival rates, while X-Sketch [10] detects simplex items whose frequency follows low-degree polynomial trends. Also, Scout Sketch+ [13] introduced the concepts of promising and damping items and proposed a filter-detector architecture for their detection. However, all these methods rely on static, hand-crafted parameters and do not use learned components.

Neural sketch methods. MetaSketch [5, 4] casts sketch operations as differentiable memory accesses trained via meta-learning, while LegoSketch [12] introduces modular memory blocks for elastic scaling. Also, RatioSketch [26] proposes a lightweight micro-correction approach that keeps the efficient structure of hand-crafted sketches and adds a neural correction layer on their outputs. However, these methods focus on frequency estimation rather than trend detection, and none of them consider hardware deployment on data-plane switches.

Learned data structures. The learned bloom filter [16] uses a neural network to predict set membership, with a backup traditional bloom filter that handles prediction errors. Mitzenmacher [19] proposed a multi-layer architecture that places the learned model between two bloom filters to

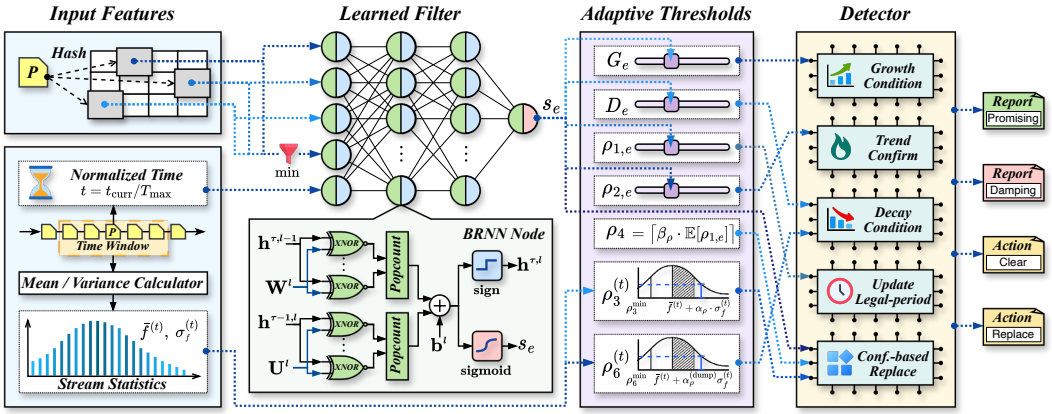

Figure 1: Overview of NeuTrend. The learned filter uses a BRNN deployed on the switch data plane to classify items as likely trending or non-trending. The BRNN confidence scores adaptively control the detection thresholds in the detector.

reduce false positive rates. Partitioned learned bloom filter [24] further improved space efficiency by allocating different backup filter sizes to different confidence regions. These works show that hash-based filters can be replaced with learned components. However, they have not been applied to trend detection in data streams.

In-network machine learning. Planter [32] provides a framework for deploying ML models on data plane switches. It supports decision trees and random forests. Also, IIsy [31] maps tree-based models onto match-action tables. pForest [2] demonstrates random forest inference on data plane switches. Brain-on-Switch (BoS) [27] enables binarized RNN inference by encoding network activations as lookup tables, achieving line-rate traffic analysis. N3IC [21] compiles BRNN models onto SmartNICs. These works show that neural network inference at line rate is feasible. However, none has applied this capability to sketch-based trend detection.

3 NeuTrend

3.1 Problem

We consider a high-speed data stream $\mathcal{S} = \{\langle e_1, t_1\rangle, \langle e_2, t_2\rangle, \ldots\}$, where e denotes the item ID and t denotes the time window index. The stream is divided into consecutive time windows of fixed length. Let $f_e^{(t)}$ denote the frequency of item e in window t. We define a *legal period* as a set of consecutive windows where the frequency of an item stays within a range. The range is controlled by a growth factor $G \geq 1$, a decay factor $0 < D \leq 1$, and a maximum period length ρ_1 [13]. An item e is called a *promising item* if its fiducial frequency V_k in the k-th legal period shows overall growth, i.e., $V_{k+1} \geq G \cdot V_k$ for at least ρ_2 consecutive legal periods. Similarly, a *damping item* shows overall decline, i.e., $V_{k+1} \leq D \cdot V_k$. Existing solutions also require a frequency threshold ρ_3 for admission, a protection period ρ_4, a replacement threshold ρ_5, and a damping frequency threshold ρ_6. There are eight coupled parameters in total.

The core challenge is to detect these items accurately and in real time on data-plane switches while avoiding the manual tuning of these eight parameters. In detail, switches have three constraints.

C1. The ASIC follows a multi-stage match-action pipeline, while the number of stages is limited, e.g., 12 stages on a Tofino switch.

C2. Each stage has only a few MB of on-chip SRAM, e.g., about 1 MB per stage. Thus, the per-item memory must be small.

C3. Each stage offers very limited compute resources. In detail, each item can issue at most one register read or write per stage, while floating-point arithmetic is not supported. Thus, the detection logic must be expressed with bounded integer or bit-level operations within a fixed pipeline depth.

3.2 NeuTrend Design

To overcome high false positive rates and manual parameter tuning, NeuTrend upgrades the filter-detector pipeline (Figure 1). A BRNN replaces the static bloom filter to make data-driven admission decisions, outputting a confidence score that automatically sets the detector's per-item growth, decay, and period parameters. This design strictly satisfies hardware constraints (Section 3): the BRNN compiles into a constant number of match-action stages (**C1**) using SRAM-compliant tables (**C2**), and relies exclusively on XNOR, popcount, and bounded integer operations to ensure line-rate execution (**C3**).

3.2.1 Learned Filter

The learned filter replaces the static bloom filter with a small BRNN. The BRNN learns to separate trending from non-trending items. Thus, admission decisions are derived from data rather than from a fixed hash structure. In detail, the BRNN uses only binarized weights $W_b \in \{-1, +1\}^{n_{\text{out}} \times n_{\text{in}}}$ and binarized activations, while these binary operations compile into XNOR and popcount operations supported by current data plane switches.

Input features. For each incoming item e in window t, the BRNN takes a continuous sequence of feature vectors built from the information available in the data plane, maintaining a training sequence length of T_{seq}. During in-network inference, the BRNN processes items step-by-step, updating the hidden state upon each arrival. Let $A[h_i(e)]$ denote the counter value at the i-th hash position of the item in the lightweight counter array of the detector. At each time step, the feature vector is given by Eq. (1), where N_w is the number of items in the current window and $t_{\text{curr}}/T_{\max}$ is the normalized progress within the current window. The full input sequence is $X_e = [\mathbf{x}_e^{(t-T_{seq}+1)}, \ldots, \mathbf{x}_e^{(t)}]$.

$$\mathbf{x}_e^{(t)} = \left[\frac{A[h_1(e)]}{N_w}, \frac{A[h_2(e)]}{N_w}, \ldots, \frac{A[h_d(e)]}{N_w}, \frac{\min_i A[h_i(e)]}{N_w}, \frac{t_{\text{curr}}}{T_{\max}} \right] \in \mathbb{R}^{d+2} \tag{1}$$

BRNN architecture. The BRNN comprises binarized recurrent and L binarized fully connected layers. To overcome SRAM bottlenecks, hidden states are maintained per hash bucket rather than per item. Since input features derive directly from bucket counts, state memory scales linearly with the fixed sketch size, completely decoupling it from the concurrent flow volume. Although state sharing causes cross-flow collisions, trending items dominate bucket counters. The BRNN robustly tracks this dominant trend, filtering out perturbations from low-frequency items. For layer l at sequence step $\tau \in [1, T_{seq}]$, the recurrent forward pass is given by Eq. (2), where $\mathbf{h}^{(\tau,l)}$ is the hidden state and $\text{sign}(\cdot)$ binarizes the pre-activation values. In hardware, the matrix multiplications $W_b^{(l)} \cdot \mathbf{h}^{(\tau,l-1)}$ and $U_b^{(l)} \cdot \mathbf{h}^{(\tau-1,l)}$ are computed via XNOR between the binarized weight bits and the binarized input/state bits, followed by a popcount operation. The final layer produces a real-valued output via Eq. (3) using the state at the last step T_{seq}, where σ is the sigmoid function and s_e is the predicted probability that item e is trending.

$$\mathbf{h}^{(\tau,l)} = \text{sign}\Big(W_b^{(l)} \cdot \mathbf{h}^{(\tau,l-1)} + U_b^{(l)} \cdot \mathbf{h}^{(\tau-1,l)} + \mathbf{b}^{(l)} \Big) \tag{2}$$

$$s_e = \sigma\Big(\mathbf{w}_{\text{out}}^{\top} \mathbf{h}^{(T_{seq},L)} + b_{\text{out}} \Big) \in (0, 1) \tag{3}$$

Multi-layer architecture. A small backup bloom filter $\mathcal{B}_{\text{backup}}$ with memory m_b is placed after the BRNN. Items rejected by the BRNN are still admitted to the detector if the backup filter accepts them. Specifically, the backup filter is queried only when $s_e \leq \tau$, giving the filter-admission condition $(s_e > \tau) \vee (\mathcal{B}_{\text{backup}}(e) = 1)$ used in Algorithm 1. This preserves the backup filter's false-negative guarantee, while the false negative rate of the combined filter does not exceed that of the backup filter. Also, Theorem 2 in Section 3.4 shows that the combined filter achieves a strictly lower false admission rate than the original bloom filter when the BRNN classification accuracy is sufficient.

3.2.2 Adaptive Threshold

The adaptive threshold module derives detection parameters from BRNN confidence scores, stream statistics, and learned bounds. For items with high confidence, the growth and decay thresholds are relaxed, and the required period count is shortened. For items with low confidence, the thresholds are

made stricter and thus require more evidence. This adaptive design removes the need to set any of the eight parameters globally.

Confidence-controlled growth and decay. For each item stored in a detector cell, we define adaptive growth and decay thresholds via Eq. (4), where $G_{\min}, G_{\max} \in (1, 2]$ and $D_{\min}, D_{\max} \in (0, 1)$ are the bounds learned during training. When the BRNN is highly confident that e is trending, i.e., $s_e \to 1$, the growth threshold G_e is relaxed toward $G_{\min}$ and the decay threshold D_e is relaxed toward $D_{\max}$, which makes detection more sensitive. In contrast, when the BRNN confidence is low, i.e., $s_e \to 0$, the thresholds become strict, which reduces false positives.

$$G_e = G_{\min} + (G_{\max} - G_{\min}) \cdot (1 - s_e), \quad D_e = D_{\max} - (D_{\max} - D_{\min}) \cdot (1 - s_e) \tag{4}$$

Adaptive period parameters. The legal period length ρ_1 and minimum number of legal periods ρ_2 are controlled by the BRNN confidence score via Eq. (5), where $\rho_1^{\min}, \rho_1^{\max}, \rho_2^{\min}, \rho_2^{\max}$ are bounds learned during training. Items with high BRNN confidence receive longer legal periods, allowing larger frequency variation. Also, they require fewer legal periods for detection, while they are reported faster. Conversely, items with low confidence require more evidence to be reported.

$$\rho_{1,e} = \left\lfloor \rho_1^{\max} \cdot s_e + \rho_1^{\min} \cdot (1 - s_e) \right\rfloor, \quad \rho_{2,e} = \left\lceil \rho_2^{\max} \cdot (1 - s_e) + \rho_2^{\min} \cdot s_e \right\rceil \tag{5}$$

Frequency thresholds from load estimation. The frequency threshold ρ_3 is set automatically from the stream statistics rather than manually. In detail, let $\bar{f}^{(t)}$ denote the average item frequency in window t and let $\sigma_f^{(t)}$ denote the standard deviation. Then we set $\rho_3^{(t)}$ via Eq. (6), where $\alpha_\rho > 0$ is a learned scaling parameter and $\rho_3^{\min}$ is a minimum threshold. This admits only items whose frequency is above the window average. Also, the protection period ρ_4 prevents new items from being removed before they show a trend while we derive it from the expected legal period length via Eq. (7). Here $\beta_\rho > 0$ is a learned scaling parameter. The damping frequency threshold ρ_6 requires damping items to start from a sufficiently high frequency, while we set it in the same way as ρ_3 via Eq. (8). Here $\alpha_\rho^{(\text{damp})} > 0$ is a learned parameter and $\rho_6^{\min}$ is a minimum threshold.

$$\rho_3^{(t)} = \max\left(\bar{f}^{(t)} + \alpha_\rho \cdot \sigma_f^{(t)},\ \rho_3^{\min}\right) \tag{6}$$

$$\rho_4 = \left\lceil 0.5 \cdot \beta_\rho \cdot \left(\rho_1^{\max} + \rho_1^{\min}\right) \right\rceil \tag{7}$$

$$\rho_6^{(t)} = \max\left(\bar{f}^{(t)} + \alpha_\rho^{(\text{damp})} \cdot \sigma_f^{(t)},\ \rho_6^{\min}\right) \tag{8}$$

Among the eight original parameters: (1) G, D, ρ_1, ρ_2, ρ_3, and ρ_6 are now set by the adaptive module; (2) ρ_4 is derived from the learned period bounds; (3) ρ_5 is replaced by the confidence-based replacement strategy in Algorithm 1. All adaptive bounds, including $(G_{\min}, G_{\max})$ and $(D_{\min}, D_{\max})$, are jointly fine-tuned during training. Section 4 shows that the final performance is not sensitive to the initial values of these two bound pairs. Thus, the user only needs to provide approximate initial ranges for them.

3.2.3 Insertion and Detection

The insertion operation of NeuTrend relies on the BRNN inference to replace the bloom filter check, and a confidence-based strategy to replace the static replacement threshold. The detection operation at the end of each time window applies the per-item adaptive thresholds G_e, D_e, $\rho_{1,e}$, and $\rho_{2,e}$ to each non-empty cell. It reports promising items when the growth condition holds and damping items when the decay condition holds. Pseudocode for both insertion and detection operations is provided in Appendix C.1.

Hardware deployment. The BRNN inference compiles into match-action tables and stateful register arrays on the data plane switch. In detail, each binarized layer is partitioned into groups of g inputs and groups of g outputs, while it is implemented as $\lceil n_{\text{in}}/g \rceil \times \lceil n_{\text{out}}/g \rceil$ parallel tables with at most 2^g entries each. Then the partial results are combined by popcount across stages. The total table size fits within the capacity of a Tofino switch. The per-layer details and the stage allocation are given in Appendix C.3.

3.3 Training

NeuTrend is trained in two phases. We first pre-train the BRNN on synthetic Zipf-distributed streams. Each item is labeled as trending if it matches the promising or damping definition under default parameters ($G = 1.2, D = 0.8, \rho_1 = 5, \rho_2 = 4$). We use the binary cross-entropy loss in Eq. (9) with the straight-through estimator [1] for gradient computation through the sign function.

$$\mathcal{L}_{\mathrm{BRNN}} = -\frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\left[y_i \log s_{e_i} + (1-y_i)\log(1-s_{e_i})\right] \tag{9}$$

Then we fine-tune the BRNN parameters jointly with all adaptive threshold bounds. Since the F1 score is not differentiable, the objective is a soft-F1 approximation, while it is combined with the BRNN loss via the multi-task weighting scheme of [15]. The full loss formulation, the list of jointly trained bounds, and the optimizer settings are given in Appendix C.2.

3.4 Theoretical Analysis

This section provides a formal analysis of the baseline filter-detector architecture's shortcomings, and proves the correctness, admission purity, and complexity properties of NeuTrend.

False admission traffic. We first quantify the cost of using a bloom filter in front of the detector. Proposition 1 gives the fraction of the insertion traffic caused by non-trending items as a function of the bloom filter false positive rate. For typical streams where most items are non-trending, this fraction is large even at small values of p_{BF}, causing severe eviction pressure. Theorem 2 then shows that the multi-layer learned filter achieves a strictly lower combined false admission rate than the original bloom filter when the BRNN classification effectively filters non-trending items.

Proposition 1. *In the filter-detector architecture, the bloom filter admits non-trending items at a false positive rate p_{BF}. The expected fraction of false admissions (i.e., insertion traffic caused by non-trending items entering the detector) is $n_{\mathrm{nt}} \cdot p_{\mathrm{BF}}/(n_t + n_{\mathrm{nt}} \cdot p_{\mathrm{BF}})$, where n_t and n_{nt} denote the numbers of trending and non-trending items in the stream.*

Theorem 2. *Let p_{BRNN} denote the false positive rate of the BRNN classifier for non-trending items. Let $p_{\mathrm{BF}}(m_b)$ denote the false positive rate of the backup bloom filter with memory m_b. The combined false positive rate of the multi-layer learned filter is $p_{\mathrm{LF}} = p_{\mathrm{BRNN}} + (1 - p_{\mathrm{BRNN}}) \cdot p_{\mathrm{BF}}(m_b)$. Let $p_{\mathrm{BF}}^{\mathrm{orig}}$ denote the false admission rate of the original bloom filter. The learned filter achieves $p_{\mathrm{LF}} < p_{\mathrm{BF}}^{\mathrm{orig}}$ when $p_{\mathrm{BRNN}} < (p_{\mathrm{BF}}^{\mathrm{orig}} - p_{\mathrm{BF}}(m_b))/(1 - p_{\mathrm{BF}}(m_b))$.*

Admission purity. A lower false admission rate at the filter directly increases the purity of the candidate pool entering the detector, thereby reducing the probability of true trending items being prematurely evicted.

Theorem 3. *Let $\mathcal{P}^{\mathrm{BF}} = n_t/(n_t + n_{\mathrm{nt}} \cdot p_{\mathrm{BF}}^{\mathrm{orig}})$ denote the admission purity (the probability that an admitted item is truly trending) under the original bloom filter. Let $\mathcal{P}^{\mathrm{LF}}$ denote the admission purity under the learned filter. When $p_{\mathrm{LF}} < p_{\mathrm{BF}}^{\mathrm{orig}}$, we have $\mathcal{P}^{\mathrm{LF}} > \mathcal{P}^{\mathrm{BF}}$.*

End-to-end accuracy. The previous results bound the filter's behavior. We next show that reducing the false positive rate of the BRNN does not degrade the end-to-end F1 score.

Theorem 4. *Assume the true positive rate (TPR) of the BRNN is non-decreasing. If the adaptive thresholds are jointly optimized with the BRNN, the end-to-end F1 score of NeuTrend is monotonically non-increasing with respect to the BRNN false positive rate p_{BRNN} (i.e., the F1 score improves or remains stable as false positives decrease).*

Time complexity. NeuTrend retains the $O(1)$ per-item complexity of the original pipeline.

Theorem 5. *The time complexity of all NeuTrend operations on each item is $O(1)$. In hardware, the BRNN inference occupies a constant number of pipeline stages determined by the layer dimensions and the group size used for table partitioning.*

4 Experiments

Our evaluation shows (1) whether NeuTrend achieves higher detection accuracy than existing solutions and other baselines, (2) whether the BRNN-based learned filter reduces false positive rates compared

to the bloom filter, (3) whether the adaptive threshold module prevents parameter tuning, and (4) whether NeuTrend can operate at line rate when handling Tbps-level data streams.

4.1 Experimental Setup

Datasets. We evaluate on four real-world traces: MAWI 2024-10-23, MAWI 2025-11-26, MAWI 2026-03-06 [18], and a CAIDA 2018 anonymized Internet trace [3]. Detailed dataset statistics, including the trace lengths, window counts, flow counts, and the fraction of trending items, are provided in Appendix A.

Evaluation protocol. For each real-world trace, we allocate the first 80% of windows to training and the next 10% to validation in chronological order, rounding both counts to the nearest integer and assigning all remaining windows to testing. We jointly optimize the BRNN parameters and adaptive threshold bounds on the training split, and select the checkpoint and its associated classification threshold τ using validation macro-F1. After model selection, we fix the BRNN parameters, learned bounds, and τ, and evaluate the model once on the later test windows in a forward-only pass.

Baselines. We compare NeuTrend against six baselines. (1) AC-CM [7] consists of a CM sketch for recording item frequency in each time window and multiple buckets where each cell stores the item ID, the legal period count, the starting window index, and $\rho_2{+}1$ additional counters for tracking frequency values over the most recent $\rho_2{+}1$ windows. (2) AC-CU [9] has the same structure as AC-CM. It adopts the conservative update strategy for the CM sketch to reduce overestimation. (3) FC-CM [7, 13] also consists of a CM sketch and multiple buckets. It simplifies the legal period determination by replacing the $\rho_2{+}1$ frequency counters with two compact fields, i.e., the fiducial frequency V_m and the current period length N_d. (4) FC-CU [9, 13] is similar to FC-CM. It uses the conservative update strategy for the CM sketch. (5) We configure Scout Sketch+ with manually tuned parameters, or (6) with grid-searched parameters that select the best F1 score over a grid of 1000 settings. AC-CM, AC-CU, FC-CM, FC-CU, and Scout Sketch+ with manually tuned parameters use $G{=}1.2$, $D{=}0.8$, $\rho_1{=}5$, $\rho_2{=}4$, $\rho_3{=}16$, $\rho_4{=}5$, $\rho_5{=}4$, and $\rho_6{=}64$.

NeuTrend trains its BRNN via truncated backpropagation through time (BPTT) with sequence length $T_{seq} = 12$. The 0.6 KB model uses 5 input features, a 16-dimensional binarized recurrent state, and a scalar output, backed by a bloom filter using $d = 3$ hash functions. All software experiments run on a server with one NVIDIA RTX A6000 GPU (48 GB VRAM), AMD EPYC 7F72 24-Core Processor CPUs at 3.20 GHz, and 512 GB RAM. The hardware experiments are evaluated on an Intel Tofino Wedge 100BF-32X hardware switch.

4.2 Detection Accuracy

Setting. We compare all methods on four datasets with memory ranging from 5 KB to 25 KB.

Result. Figure 2 summarizes the F1 scores across datasets at 10 KB memory. NeuTrend achieves the highest F1 score on all datasets, while it outperforms Scout Sketch+ with manual tuning by $1.41\times$-$1.58\times$ and Scout Sketch+ with grid search by $1.28\times$-$1.62\times$. The improvement is largest on datasets where the traffic distribution is irregular and manual parameter tuning is difficult. Also, across all memory budgets, e.g., the smaller allocations in Figure 3, the improvement over manually tuned Scout Sketch+ reaches up to $1.68\times$.

Figure 2 also reveals the precision–recall trade-off. Scout Sketch+ with manual tuning achieves high precision but low recall. Grid search selects configurations that maximize F1, increasing recall at the expense of precision. Consequently, fewer true trends are missed, but a larger fraction of the reported trends are false positives. NeuTrend achieves a better balance between precision and recall, yielding the highest F1 score across all four datasets.

Figure 3 shows how the F1 score varies with the memory budget. NeuTrend outperforms the other methods shown in Figure 3, while it achieves the most benefits at small memory budgets, where bloom filter collisions are the most severe.

4.3 Filter Analysis

Setting. We measure the false positive rate of the learned filter of NeuTrend when compared with the standard bloom filter used in existing solutions at different memory allocations on each dataset.

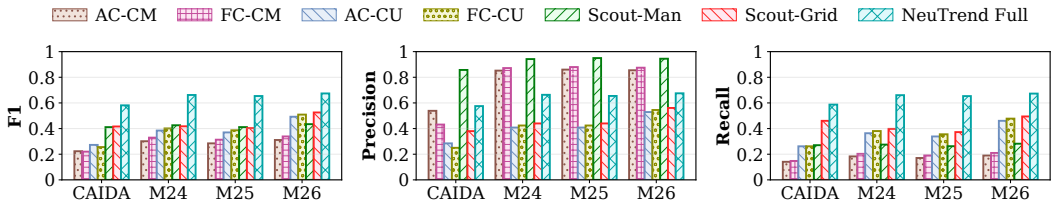

Figure 2: Comparison of F1 score, precision, and recall under the 10 KB memory budget, shown from left to right.

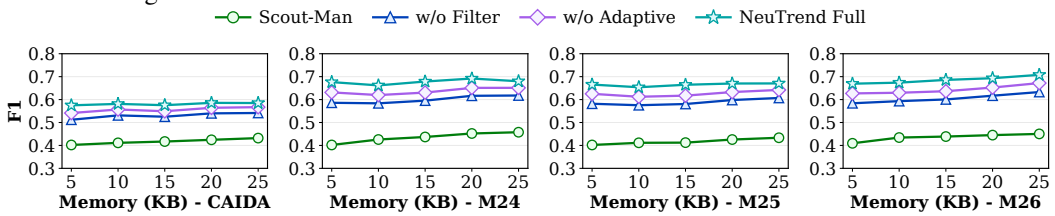

Figure 3: F1 score as a function of memory budget.

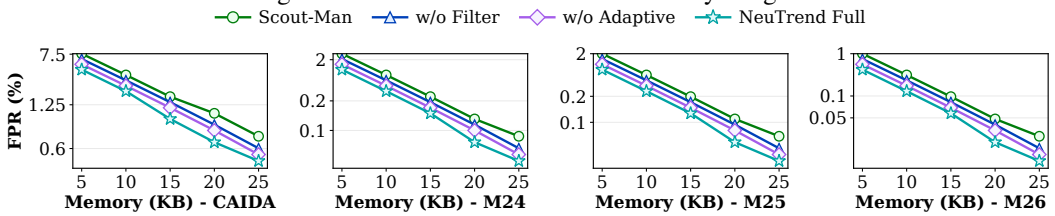

Figure 4: False positive rate (FPR) as a function of memory budget.

Result. Figure 4 shows that the learned filter achieves 1.3×-1.8× lower false positive rates than the standard bloom filter under the same memory budget. Also, Figure 5 confirms that the learned filter increases the fraction of detector cells occupied by truly trending items from 47% to 70%, while this result is consistent with Theorem 3.

4.4 Parameter Sensitivity and Score Distribution

Setting. We examine the internal confidence score distribution of the learned filter and compare the F1 score variance of Scout Sketch+ across random parameter settings against NeuTrend.

Result. Figure 6 demonstrates that the BRNN confidence scores span the probability range without collapsing into trivial predictions. The distribution naturally skews toward lower values, accurately reflecting the inherent scarcity of true trending items in real-world traffic. Furthermore, Figure 7 highlights the clear performance gap: even across extensive sampled configurations, the F1 scores of Scout Sketch+ remain strictly lower. In contrast, by deriving thresholds automatically from the BRNN scores, NeuTrend eliminates manual tuning and achieves a significantly superior F1 score.

4.5 Hardware Deployment and Resource Utilization

Setting. We evaluate the deployment feasibility of NeuTrend and Scout Sketch+ on an Intel Tofino Wedge 100BF-32X hardware switch using the Intel P4 Studio compiler toolchain. Implemented in P4 and tested under a fixed memory budget using the MAWI workload, we verify that all methods compile successfully and sustain Tbps-level line-rate processing on the physical hardware. To adapt the BRNN to the multi-stage pipeline, NeuTrend maintains hidden states directly within the sketch bucket SRAM. Since BRNN input features derive from aggregate bucket states rather than individual packet histories, the network computes recurrent updates step-by-step across consecutive match-action stages.

Result. Table 1 summarizes the deployment metrics based on the compiler's resource allocation. All evaluated methods theoretically guarantee the 1 Tbps (1000 Gbps) line-rate throughput on the

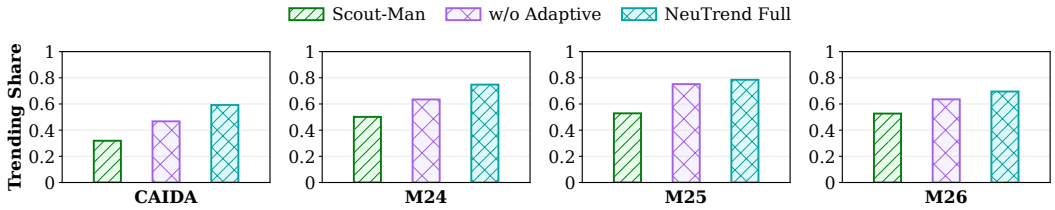

Figure 5: Detector cell occupancy analysis.

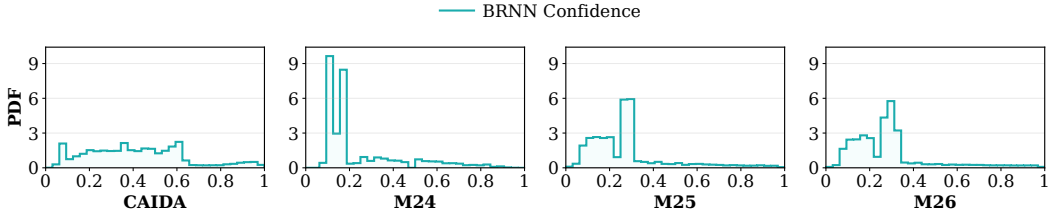

Figure 6: Learned-filter score histograms.

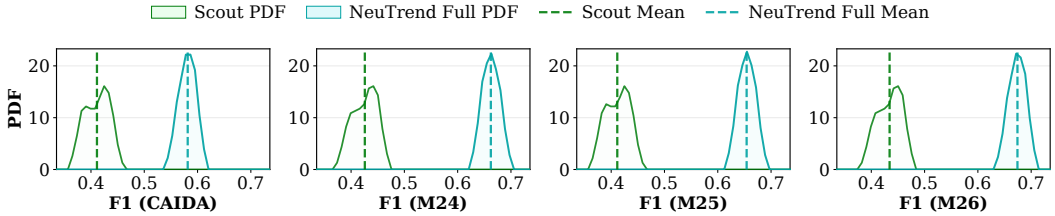

Figure 7: Distribution of sampled F1 scores.

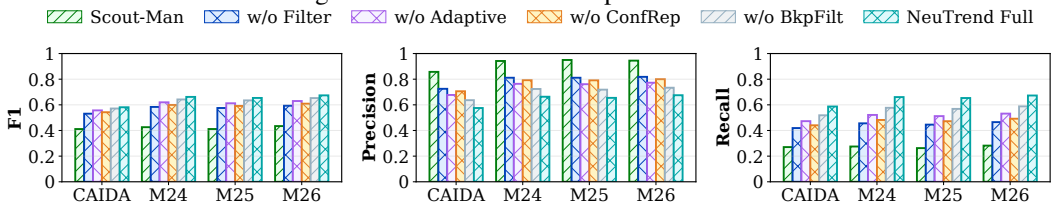

Figure 8: Ablation study: F1 score comparison across different model variants.

Table 1: Hardware throughput and resource footprint under a 10 KB logical memory budget.

Method	Thpt. (Gbps)	Latency (µs)	Stages	Tables	Entries	SRAM (KiB)
Scout Sketch+	1000	1.0	6	3	6,331	10.46
NeuTrend	1000	1.2	12	17	4,128	17.06

Tofino ASIC. While NeuTrend requires more logical stages than the Scout Sketch+ to accommodate its lookup tables, its total SRAM footprint fits entirely within the Tofino pipeline capacity limits. This footprint includes the memory required for the learned filter, the adaptive detector, and the state storage.

4.6 Ablation Study

Setting. We evaluate each component by removing it individually on each dataset at 10 KB.

Result. Figure 8 shows that all components contribute to the overall performance. In detail, the adaptive threshold module provides the largest individual improvement, i.e., $+0.07$ F1. Also, the combination of all components provides a larger improvement than the sum of individual gains. The reason is that the adaptive thresholds use the BRNN confidence scores to adjust detection sensitivity on a per-item basis, while these confidence scores are only available in the learned filter.

5 Conclusion

We propose NeuTrend, a framework for accurate, automatic, and high-performance trend detection in data streams. Its BRNN-based learned filter achieves lower false positive rates than the Bloom filter while maintaining the backup filter's false-negative guarantee. Also, the adaptive thresholds are controlled by the BRNN confidence scores while they automatically adjust detection sensitivity based on per-item trend evidence and reduce the number of manually tuned parameters. Theoretical analysis proves two properties of NeuTrend. First, the learned filter improves the effective detector capacity (Theorem 3). Second, the F1 score is monotonically non-increasing in the BRNN false positive rate (Theorem 4). Experiments on four real-world datasets show $1.35\times$-$1.68\times$ F1 improvement over existing solutions and Tbps-level line-rate operation on Intel Tofino switches.

Limitations. The BRNN requires offline training on labeled data streams, which may not always be available. Also, the model capacity of the BRNN is constrained by the lookup table size on data plane switches, which limits its ability to capture complex trend patterns. We leave online learning and larger BRNN architectures on next-generation switch hardware as future work.

Broader Impacts. This work contributes positively to more efficient and accurate network monitoring, which can enhance network security and infrastructure management. We do not foresee any direct negative societal impacts or ethical risks arising from the deployment of this framework.

Acknowledgments and Disclosure of Funding

This work was supported by the Science and Technology Program of Fujian Province (Grant No. 2025H6007). The authors declare no competing interests.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[2] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680*, 2019.

[3] CAIDA. The CAIDA anonymized internet traces. `https://www.caida.org/catalog/datasets/passive_dataset/`. Accessed: 2026-04.

[4] Yukun Cao, Yuan Feng, Hairu Wang, Xike Xie, and S Kevin Zhou. Learning to sketch: A neural approach to item frequency estimation in streaming data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(11):7136–7153, 2024.

[5] Yukun Cao, Yuan Feng, and Xike Xie. Meta-sketch: A neural data structure for estimating item frequencies of data streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6916–6924, 2023.

[6] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International colloquium on automata, languages, and programming*, pages 693–703. Springer, 2002.

[7] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[8] Rui Ding, Shibo Yang, Xiang Chen, and Qun Huang. Bitsense: Universal and nearly zero-error optimization for sketch counters with compressive sensing. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 220–238, 2023.

[9] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[10] Zhuochen Fan, Jiarui Guo, Xiaodong Li, Tong Yang, Yikai Zhao, Yuhan Wu, Bin Cui, Yanwei Xu, Steve Uhlig, and Gong Zhang. Finding simplex items in data streams. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1953–1966. IEEE, 2023.

[11] Zhuochen Fan, Yinda Zhang, Tong Yang, Mingyi Yan, Gang Wen, Yuhan Wu, Hongze Li, and Bin Cui. Periodicsketch: Finding periodic items in data streams. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 96–109. IEEE, 2022.

[12] Yuan Feng, Yukun Cao, Hairu Wang, Xike Xie, and S Kevin Zhou. Lego sketch: A scalable memory-augmented neural network for sketching data streams. *arXiv preprint arXiv:2505.19561*, 2025.

[13] Guoju Gao, Tianyu Ma, He Huang, Yu-E Sun, Haibo Wang, Yang Du, and Shigang Chen. Scout sketch+: Finding both promising and damping items simultaneously in data streams. *IEEE/ACM Transactions on Networking*, 32(6):5491–5506, 2024.

[14] Raf Huysegems, Bart De Vleeschauwer, Koen De Schepper, Chris Hawinkel, Tingyao Wu, Koen Laevens, and Werner Van Leekwijck. Session reconstruction for http adaptive streaming: Laying the foundation for network-based qoe monitoring. In *2012 IEEE 20th International Workshop on Quality of Service*, pages 1–9. IEEE, 2012.

[15] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[16] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 international conference on management of data*, pages 489–504, 2018.

[17] Xiaodong Li, Zhuochen Fan, Haoyu Li, Zheng Zhong, Jiarui Guo, Sheng Long, Tong Yang, and Bin Cui. Steadysketch: Finding steady flows in data streams. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, pages 01–09. IEEE, 2023.

[18] MAWI Working Group. MAWI working group traffic archive. `https://mawi.wide.ad.jp/mawi/`. Accessed: 2026-04.

[19] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. *Advances in neural information processing systems*, 31, 2018.

[20] Robert Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 207–212, 2004.

[21] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. Re-architecting traffic analysis with neural network interface cards. In *19th USENIX symposium on networked systems design and implementation (NSDI 22)*, pages 513–533, 2022.

[22] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. Programmable packet scheduling at line rate. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 44–57, 2016.

[23] Lu Tang, Qun Huang, and Patrick PC Lee. Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2026–2034. IEEE, 2019.

[24] Kapil Vaidya, Eric Knorr, Tim Kraska, and Michael Mitzenmacher. Partitioned learned bloom filter. *arXiv preprint arXiv:2006.03176*, 2020.

[25] Kleber Vieira, Alexandre Schulter, Carlos Westphall, and Carla Westphall. Intrusion detection for grid and cloud computing. *It Professional*, 12(4):38–43, 2009.

[26] Mengbo Wang, Zhuochen Fan, Dayu Wang, Guorui Xie, Qing Li, Zeyu Luan, Yong Jiang, Tong Yang, and Mingwei Xu. Ratiosketch: Towards more accurate frequency estimation in data streams via a lightweight neural network. 2026.

[27] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, Ke Xu, Mingwei Xu, and Jianping Wu. {Brain-on-Switch}: Towards advanced intelligent network data plane via {NN-Driven} traffic analysis at {Line-Speed}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 419–440, 2024.

[28] Kaicheng Yang, Sheng Long, Qilong Shi, Yuanpeng Li, Zirui Liu, Yuhan Wu, Tong Yang, and Zhengyi Jia. Sketchint: Empowering int with towersketch for per-flow per-switch measurement. *IEEE Transactions on Parallel and Distributed Systems*, 34(11):2876–2894, 2023.

[29] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. Heavyguardian: Separate and guard hot items in data streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2584–2593, 2018.

[30] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.

[31] Changgang Zheng, Zhaoqi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. Iisy: Hybrid in-network classification using programmable switches. *IEEE/ACM Transactions on Networking*, 32(3):2555–2570, 2024.

[32] Changgang Zheng, Mingyuan Zang, Xinpeng Hong, Liam Perreault, Riyad Bensoussane, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. Planter: Rapid prototyping of in-network machine learning inference. *ACM SIGCOMM Computer Communication Review*, 54(1):2–21, 2024.

[33] Zuowu Zheng, Xiaofeng Gao, Xiao Ma, and Guihai Chen. Predicting hot events in the early period through bayesian model for social networks. *IEEE Transactions on Knowledge and Data Engineering*, 34(3):1390–1403, 2020.

[34] Zheng Zhong, Shen Yan, Zikun Li, Decheng Tan, Tong Yang, and Bin Cui. Burstsketch: Finding bursts in data streams. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2375–2383, 2021.

A Dataset Details

Figure 9 and Figure 10 summarize the detailed statistics of the evaluated traces, including the trace lengths, window counts, flow counts, and the fraction of trending items. Note that Figure 10 provides a formatted visualization uniformly cropped from the initial continuous segment of the windows across all traces.

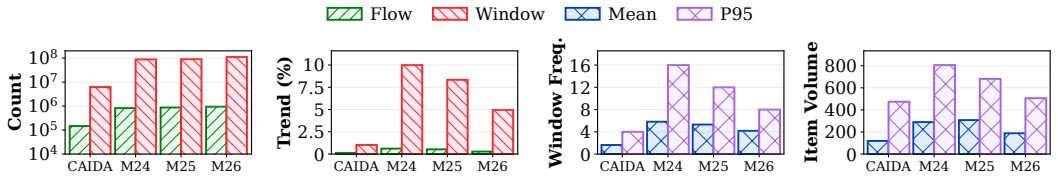

Figure 9: Dataset statistics: flow and window counts, trending-positive rates, alongside mean and P95 distributions of window frequencies and per-item volumes.

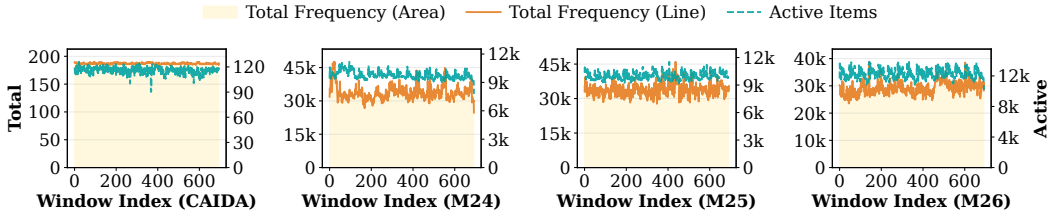

Figure 10: Window-level activity observed across the traces.

B Proofs

B.1 Proof of Proposition 1

Proof. In each time window, items are admitted to the detector if they pass the filter. The total number of expected admissions is the sum of all trending items n_t (assuming the filter does not reject true trending items) and the falsely admitted non-trending items $n_{\text{nt}} \cdot p_{\text{BF}}$. Thus, the proportion of insertion operations triggered by non-trending items in the detector is exactly $n_{\text{nt}} \cdot p_{\text{BF}} / (n_t + n_{\text{nt}} \cdot p_{\text{BF}})$. For $n_{\text{nt}}/n > 0.95$ and $p_{\text{BF}} = 0.1$, this fraction exceeds 65%, indicating that a majority of the detector's insertion traffic is wasted on evaluating non-trending items, causing severe eviction pressure. □

B.2 Proof of Theorem 2

Proof. The learned filter admits a non-trending item e under two mutually exclusive cases. First, the BRNN falsely predicts the item as trending (probability p_{BRNN}). Second, the BRNN correctly rejects it, but the backup bloom filter falsely accepts it (probability $(1 - p_{\text{BRNN}}) \cdot p_{\text{BF}}(m_b)$). Summing these gives $p_{\text{LF}} = p_{\text{BRNN}} + (1 - p_{\text{BRNN}}) \cdot p_{\text{BF}}(m_b)$. Rearranging $p_{\text{LF}} < p_{\text{BF}}^{\text{orig}}$ yields the condition $p_{\text{BRNN}} < (p_{\text{BF}}^{\text{orig}} - p_{\text{BF}}(m_b))/(1 - p_{\text{BF}}(m_b))$. The original bloom filter admits any item whose frequency reaches ρ_3, while the BRNN is explicitly trained to separate trending from non-trending patterns. □

B.3 Proof of Theorem 3

Proof. The admission purity $\mathcal{P}$ is defined as the expected ratio of true trending items to all admitted items. For a generic filter with false positive rate p_{filter}, $\mathcal{P} = n_t/(n_t + n_{\text{nt}} \cdot p_{\text{filter}})$. Taking the derivative of $\mathcal{P}$ with respect to p_{filter} yields a strictly negative value, meaning $\mathcal{P}$ is monotonically decreasing with p_{filter}. By Theorem 2, when $p_{\text{LF}} < p_{\text{BF}}^{\text{orig}}$, it strictly follows that $\mathcal{P}^{\text{LF}} > \mathcal{P}^{\text{BF}}$. Higher purity directly reduces the eviction probability of true trending items in the finite detector cells. □

B.4 Proof of Theorem 4

Proof. The end-to-end precision (PR) and recall (RR) depend on the filter rates ($\mathrm{TPR}_F, \mathrm{FPR}_F$) and detector rates ($\mathrm{TPR}_D, \mathrm{FPR}_D$). Decreasing p_{BRNN} (which dictates FPR_F) strictly decreases the number of non-trending items admitted. Under a non-decreasing TPR_F, this guarantees that the candidate pool purity increases (Theorem 3), which reduces eviction pressure. Consequently, FPR_D is applied to a smaller pool of false candidates, strictly increasing PR. Meanwhile, the reduced eviction pressure ensures that true trending items are less likely to be discarded, maintaining or increasing RR. Since both PR and RR are non-decreasing as p_{BRNN} decreases, the F1 score is monotonically non-increasing with respect to p_{BRNN}. Joint optimization ensures that detector parameters adapt optimally to the current p_{BRNN} to maximize this score. □

B.5 Proof of Theorem 5

Proof. The BRNN inference requires L binarized matrix-vector multiplications, each $O(n_{\mathrm{in}} \cdot n_{\mathrm{out}})$. Since $L, n_{\mathrm{in}}, n_{\mathrm{out}}, T_{seq}$ are fixed constants independent of stream length N or distinct items n, the inference is $O(1)$. Detector operations are $O(c)$, where c is a constant number of cells per bucket. Thus, total per-item complexity is $O(1)$. In hardware, mapping bounded vector sizes to fixed PISA match-action stages yields a constant pipeline depth. □

C Additional Details

C.1 Detection Algorithm

Algorithm 1 gives the pseudocode of the insertion operation. Algorithm 2 gives the pseudocode of the detection operation in Section 3.2. At the end of each window t, every non-empty cell is examined. The per-item adaptive thresholds G_e, D_e, $\rho_{1,e}$, and $\rho_{2,e}$ are recomputed from the stored confidence score s_e via Eqs. (4)-(5). The algorithm then handles growth, decay, and legal-period transitions for each cell.

Algorithm 1 NeuTrend Insertion

Input: item e in window t, BRNN model $\mathcal{B}$, detector $\mathcal{F}$

if e is in bucket $\mathcal{F}[g(e)]$ **then**
  $e.V_c \leftarrow e.V_c + 1$
else
  Compute BRNN score: $s_e \leftarrow \mathcal{B}(X_e)$ ▷ learned filter
  if $s_e > \tau$ or $\mathcal{B}_{\mathrm{backup}}(e) = 1$ **then** ▷ query backup only if $s_e \leq \tau$
    Compute adaptive $\rho_3^{(t)}$ via Eq. (6)
    if $\min_i A[h_i(e)] \geq \rho_3^{(t)} - 1$ **then**
      if empty cell exists in $\mathcal{F}[g(e)]$ **then**
        Insert e with $V_c \leftarrow \rho_3^{(t)}$, $V_m \leftarrow 0$, $T_s \leftarrow t$, $N_p \leftarrow 0$, $N_d \leftarrow 1$, s_e stored
      else
        Find cell k^* with lowest BRNN confidence among unprotected items:
        $k^* \leftarrow \arg\min_{k: t - e_k.T_s > \rho_4} s_{e_k}$ ▷ adaptive ρ_4 via Eq. (7)
        if $s_{e_{k^*}} < s_e$ **then**
          Replace e_{k^*} with e (initialized as above)

C.2 Training Procedure

The end-to-end fine-tuning phase jointly optimizes the BRNN parameters and the threshold bound vector ($G_{\min}$, $G_{\max}$, $D_{\min}$, $D_{\max}$, $\rho_1^{\min}$, $\rho_1^{\max}$, $\rho_2^{\min}$, $\rho_2^{\max}$, α_ρ, $\rho_3^{\min}$, β_ρ, $\alpha_\rho^{(\mathrm{damp})}$, $\rho_6^{\min}, \tau$) to maximize the trend-detection F1 score. Since the F1 score is not differentiable, we use the soft-F1 approximation in Eq. (10), where $\mathcal{L}_{\mathrm{det}}$ is the soft-F1 loss in Eq. (11). During training, we use arctangent surrogate gradients for threshold comparisons, allowing gradients from the soft-F1 loss to propagate to the adaptive threshold bounds. $\mathcal{L}_{\mathrm{reg}}$ regularizes the threshold bounds to prevent

Algorithm 2 NeuTrend Detection (at end of window t)

Input: detector $\mathcal{F}$, BRNN model $\mathcal{B}$, window index t
Output: detected promising and damping items

for each non-empty cell in $\mathcal{F}$ **do**
  Retrieve stored s_e and compute adaptive $G_e, D_e, \rho_{1,e}, \rho_{2,e}$ via Eqs. (4)-(5)
  if $e.V_c \geq G_e \cdot e.V_m$ **then** ▷ growth detected
    $e.V_m \leftarrow e.V_c$; $e.N_p \leftarrow e.N_p + 1$
    if $e.N_p = \rho_{2,e}$ and $e.\text{flag} = 0$ **then**
      Report $\langle e.\text{ID}, e.T_s, t, \text{promising}\rangle$; $e.N_p \leftarrow e.N_p - 1$
    $e.N_d \leftarrow 1$; $e.V_c \leftarrow 0$
  else if $e.V_c \leq D_e \cdot e.V_m$ **then** ▷ decay detected
    Handle flag transition or report damping
  else ▷ within legal period
    if $e.N_d = \rho_{1,e}$ **then**
      Clear cell
    else
      $e.N_d \leftarrow e.N_d + 1$; $e.V_c \leftarrow 0$

degenerate solutions. The two weights λ_1 and λ_2 are set by the self-adjusting weighting of [15].

$$\mathcal{L}_{\text{E2E}} = \mathcal{L}_{\text{BRNN}} + \lambda_1 \cdot \mathcal{L}_{\text{det}} + \lambda_2 \cdot \mathcal{L}_{\text{reg}}, \tag{10}$$

$$\mathcal{L}_{\text{det}} = 1 - \frac{2\sum_i s_{e_i} \cdot y_i}{\sum_i s_{e_i} + \sum_i y_i + \epsilon}. \tag{11}$$

Ground-truth labels for real-world traces are fixed and independent of the adaptive thresholds. The BRNN is trained using the Adam optimizer with learning rate 2×10^{-3} and weight decay 10^{-5}. The straight-through estimator [1] provides gradients through the sign function. Pre-training on synthetic Zipf streams runs for up to 500 episodes with skewness in $[0.1, 1.5]$ and 10K-100K items per stream, with early stopping after 50 consecutive episodes without improvement in validation F1. End-to-end fine-tuning runs for up to 10 episodes on the training split of each real-world dataset. We select the checkpoint with the highest validation macro-F1 and retain its associated classification threshold τ for testing. The adaptive threshold bounds are initialized as $G_{\min} = 1.1$, $G_{\max} = 1.3$, $D_{\min} = 0.7$, $D_{\max} = 0.9$, $\rho_1^{\min} = 4.0$, $\rho_1^{\max} = 6.0$, $\rho_2^{\min} = 3.0$, $\rho_2^{\max} = 5.0$, $\rho_3^{\min} = 12.0$, $\rho_6^{\min} = 48.0$, and $\tau = 0.5$. These approximate defaults are refined during training. The two bound ranges that require user initialization, $(G_{\min}, G_{\max})$ and $(D_{\min}, D_{\max})$, are robust. Varying their initial values within $\pm 20\%$ of the defaults changes the F1 score by less than 0.03 across all datasets.

C.3 Hardware Deployment Details

On Tofino, the BRNN layers are compiled into match-action table entries. Switch register arithmetic quantizes the real-valued features $\mathbf{x}_e$ in Eq. (1) to 8-bit fixed-point values, yielding $5 \times 8 = 40$ input bits. These 40 bits are flattened into independent Boolean features for both training and inference, ensuring exact mathematical equivalence between software training and hardware XNOR/popcount execution as the network implicitly learns bit significance. The BRNN computation is then partitioned into groups of 8 bits. For the recurrent layer, the input projection (W_b, 40 quantized input bits to 16 binary pre-activations) uses $\lceil 40/8 \rceil \times \lceil 16/8 \rceil = 10$ tables with at most $2^8 = 256$ entries each, totaling 2,560 entries. The recurrent state projection (U_b, 16 hidden state bits to 16 binary pre-activations) uses $\lceil 16/8 \rceil \times \lceil 16/8 \rceil = 4$ tables with $2^8 = 256$ entries each, totaling 1,024 entries. The output layer ($\mathbf{w}_{\text{out}}$, 16 bits to 1) uses 2 tables with 256 entries each, totaling 512 entries. The final sigmoid activation is approximated by a piecewise linear function encoded in one additional table. The BRNN table entries are mapped to Tofino's multi-stage match-action pipeline. The feature-extraction counter array and detector logic for hash computation and cell operations are pipelined in subsequent stages.

The 10 KB budget applies uniformly to the variable sketch, filter, and detector state in the accuracy experiments. Table 1 reports the total physical SRAM footprint, including fixed storage. For NeuTrend, feature counters and the backup Bloom filter occupy 7.00 KiB, and detector state occupies 2.00 KiB, totaling 9.00 KiB within the logical budget. The BRNN lookup tables, including the sigmoid table, contribute a fixed 8.06 KiB, bringing the total physical SRAM footprint to 17.06 KiB.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims, including the accurate trend detection via a BNN-based learned filter, the elimination of manual tuning through adaptive thresholds, and the Tbps-level line-rate processing, are supported by theoretical analysis in Section 3.4 and empirical evaluation in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: A **Limitations** paragraph appears at the end of Section 5, explicitly discussing the requirement for offline training on labeled data streams and the constrained model capacity due to lookup table sizes on data plane switches.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All theoretical properties regarding false admission rates, effective capacity, end-to-end accuracy, and time complexity are formally stated in Section 3.4. The full corresponding proofs are provided in Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Experimental setup, datasets, and baseline configurations are detailed in Section 4. Full training procedures, objective functions, hyperparameter initialization, and hardware deployment specifics are given in Appendix C.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The implementation is not publicly available. The MAWI traces can be downloaded from the archive [18], and researchers can request access to the 2018 CAIDA trace directly from CAIDA [3].

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Data settings and metrics are described in Section 4. Full details regarding the Adam optimizer, learning rates, loss functions, straight-through estimator, and initialization of adaptive threshold bounds are explicitly stated in Appendix C.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The results are averaged over 5 independent runs, and it is explicitly noted that the standard deviations are small (less than 3% relative).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The computing resources are fully specified: software-based experiments were executed on a server with one NVIDIA RTX A6000 GPU (48 GB VRAM), AMD EPYC 7F72 24-Core Processor CPUs at 3.20 GHz, and 512 GB RAM, while hardware deployments were evaluated on an Intel Tofino Wedge 100BF-32X hardware switch.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research methodology conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper notes that the work contributes positively to more efficient network monitoring. It also explicitly states that there are no foreseeable negative societal impacts directly tied to this framework.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: This paper does not release any high-risk models, generative systems, or newly scraped datasets that would necessitate specific safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Section 4 credits the MAWI Working Group and CAIDA. The MAWI archive [18] permits research use subject to its privacy rules, and access to the 2018 CAIDA trace [3] requires approval under the CAIDA Acceptable Use Agreement, which prohibits re-identification and restricts redistribution.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: We release no new assets with this paper. The evaluation uses existing MAWI and CAIDA traces.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This research exclusively concerns network traffic analysis and does not involve human subjects or crowdsourcing.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This research does not involve human subjects, hence IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core methodology of NeuTrend strictly relies on binarized neural networks (BNNs) and algorithmic switch-data plane designs, and does not involve the use of LLMs as a component.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.