JITI: Dynamic Model Serving for Just-in-Time Traffic Inference

XI JIANG, University of Chicago, USA
SHINAN LIU, University of Hong Kong, Hong Kong
SALOUA NAAMA, Université Savoie Mont Blanc, France
FRANCESCO BRONZINO, École Normale Supérieure de Lyon; Institut universitaire de France, France
PAUL SCHMITT , California Polytechnic State University, USA
NICK FEAMSTER, University of Chicago, USA

Accurate and efficient inference on network traffic through machine learning models is important for many management tasks, from traffic prioritization to anomaly detection. Existing ML inference pipelines differ primarily in their feature design: those based on summary flow statistics (*e.g.*, packet sizes, inter-arrival times) are lightweight and efficient, though they may be less accurate for fine-grained classification, whereas pipelines that consume features directly from raw packet capture data can achieve higher accuracy but at significantly greater computational and resource cost. In this paper, we develop Just-in-Time Traffic Inference (JITI), a model serving system to support fast and accurate network traffic inference in raw packet-capture-based machine learning inference pipelines. Offline, JITI builds a curated pool of diverse trained models with varied feature and performance requirements. Online, JITI responds to traffic fluctuations via an adaptive scheduler that selects the model from the pool that offers the highest accuracy-to-efficiency ratio within system resource limits, thereby providing inference accuracy comparable to the more complex and resource-intensive packet-capture-based methods, with minimal efficiency compromise. Using traffic application inference as an example task, our evaluation shows that JITI improves inference performance by 18% over flow-statistics-based methods; when benchmarked against state-of-the-art packet-capture-based methods, JITI results in a worst-case drop in F1-Score of only 12.3%, while reducing the average inference decision time by ~127x.

CCS Concepts: • **Computing methodologies** → **Machine learning approaches**; • **Networks** → **Network manageability**; **Network security**.

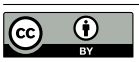
Additional Key Words and Phrases: traffic inference, online classification, network application inference

ACM Reference Format:
Xi Jiang, Shinan Liu, Saloua Naama, Francesco Bronzino, Paul Schmitt, and Nick Feamster. 2025. JITI: Dynamic Model Serving for Just-in-Time Traffic Inference. *Proc. ACM Netw.* 3, CoNEXT4, Article 45 (December 2025), 24 pages. https://doi.org/10.1145/3768992

1 Introduction

Online network traffic inference is a common practice in network management. Accurate and prompt inference decisions on traffic flows enable network operators to perform many essential network operations tasks, from monitoring bandwidth utilization to prioritize traffic (*e.g.*, for latency-sensitive applications) to ensure quality of service (QoS) or avoid network congestion, for

Authors' Contact Information: Xi Jiang, University of Chicago, Chicago, IL, USA, xijiang9@uchicago.edu; Shinan Liu, University of Hong Kong, Pok Fu Lam, Hong Kong, shinan6@hku.hk; Saloua Naama, Université Savoie Mont Blanc, Chambéry, France, saloua.naama@univ-smb.fr; Francesco Bronzino, École Normale Supérieure de Lyon; Institut universitaire de France, Lyon, France, francesco.bronzino@ens-lyon.fr; Paul Schmitt , California Polytechnic State University, San Luis Obispo, CA, USA, prs@calpoly.edu; Nick Feamster, University of Chicago, Chicago, IL, USA, feamster@uchicago.edu.



ACM 2834-5509/2025/12-ART45
https://doi.org/10.1145/3768992

capacity and resource planning, or to detect malicious traffic and infer quality of experience (QoE), and so forth [9, 17, 44, 45, 71, 77, 81, 84–86]. To support these tasks, traffic inference pipelines have been developed to manage the complete inference process, starting from capturing traffic features, preprocessing data, and applying trained models to make and deliver inference decisions in real-time. However, despite advances in inference methods, widespread deployment of inference pipelines on operational networks still face significant challenges, particularly in balancing trade-offs between inference accuracy and pipelines' computational efficiency, especially as traffic volumes increase.

As with any inference task, network traffic inference pipelines aim to maximize accuracy. However, in real-time environments, the need for timely decisions often imposes trade-offs in inference accuracy to maintain efficient operation within various constraints. Operational constraints, such as time-to-decision (TTD)—the total time from receiving the first packet to producing the final inference result, drive the efficiency of the inference pipeline. Optimizing these constraints enhances processing efficiency, reduces operational costs, and improves throughput. While failing to meet operational constraints does not always lead to pipeline failure, it can result in suboptimal performance, manifesting as increased latency or reduced throughput. In contrast, system constraints impose strict resource limitations, such as memory and CPU capacity, that the pipeline must adhere to for proper functionality. These constraints establish a performance floor that guides the optimization of operational constraints. For example, a pipeline's TTD (an operational constraint) must be short enough to process incoming traffic faster than it arrives; or packets will be dropped due to NIC buffer overflows (a system constraint). Effective management of both constraint types is crucial for maintaining accuracy and efficiency in large-scale, real-time deployments.

Over the past two decades, network traffic inference has evolved from heuristic methods to more sophisticated machine learning (ML)-based approaches. Early signature-based techniques [66, 78, 90], while efficient and able to conform to system constraints, relied on manually crafted, often brittle, features that proved less accurate in complex, evolving networks [29]. To address their limitations, two primary ML-based methodologies have emerged: Flow-statistics-based inference pipelines (FSIPs) [12, 23, 31, 49, 54, 70] and packet-capture-based inference pipelines (PCIPs) [4, 18, 28, 62, 64, 76, 79, 87, 95, 99, 103]. FSIPs utilize engineered summary statistics to maintain efficiency and low resource demand, *i.e.*, good conformance to operational and system constraints, but can compromise accuracy due to simplified traffic representations. Conversely, PCIPs apply feature-based machine learning models directly to raw packet data, significantly improving accuracy at the cost of high computational overhead, which often violates operational and system constraints (though these methods, while operating on detailed packet features, should not be confused with true representation learning techniques, *e.g.*, autoencoders or transformers, which automatically learn latent, low-dimensional representations). This trade-off between accuracy and efficiency presents challenges in large-scale, real-time network environments. Recent efforts, such as GGFAST [73], have attempted to optimize PCIPs, but struggle to generalize across diverse traffic patterns and system configurations, limiting their applicability in dynamic, real-world scenarios. As of today, existing solutions struggle to effectively balance the trade-off between inference accuracy and efficiency in real-time traffic inference pipelines.

In this paper, we address the challenge of designing a flexible and adaptive model-serving system for traffic inference pipelines that balances inference accuracy and efficiency. To this end, we introduce Just-in-Time Traffic Inference (JITI), a system that enables efficient and accurate traffic inference in packet-capture-powered pipelines without being tied to a specific hardware platform, system, or machine learning algorithm. JITI dynamically adapts to optimize operational constraints while staying within imposed system constraints by leveraging a dual-stage approach:

Stage 1: Offline Training. Deviating from traditional fixed feature inference models, JITI trains a pool of models that incorporate *varied feature sets* to suit different traffic rates and constraints.

Its heuristic-based feature exploration algorithm applies approximation techniques to quickly zero in on crucial features for a robust and efficient model set. For further adaptability, JITI also incorporates an add-on feature for adaptive batch sizes over static sizes, designed to mitigate timing uncertainties in inference model executions.

Stage 2: Online Adaptation. In real-time scenarios with fluctuating traffic rates and resource, JITI deploys an adaptive scheduler to perform constraint-bounded model selection. This scheduler leverages measured information to pick the most balanced trained model and batch size that (1) satisfies current system constraints and traffic rate, and (2) provides the highest accuracy-to-efficiency ratio, optimizing both accuracy and operational constraints.

We evaluate the effectiveness of our approach using application identification from network traffic as an example inference task. Application identification is an important function in network management for categorizing flows into corresponding applications that generated them, requiring both accuracy and efficiency to enable real-time decision-making for operations such as traffic prioritization and malicious traffic detection. Our results indicate that JITI remains optimal in both accuracy and operational constraints under various traffic rates and system constraints for the task. JITI consistently balances the trade-off between inference accuracy and efficiency by outperforming conventional PCIPs by ∼127x in terms of time-to-decision, *i.e.*, operational constraints, while largely preserving inference accuracy that is at least 18% higher than FSIPs. We also verify that JITI is able to deliver efficient and accurate inference within system constraints such as system memory and CPU capacity. To foster future research, we open-source JITI's implementation.[1]

Ethics. This work does not raise any ethical issues.

2 Motivation

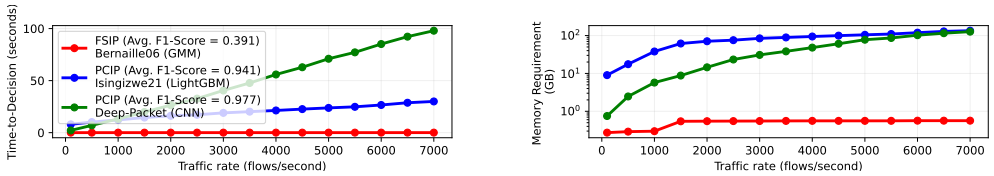
In this section, we examine the motivation behind this work by analyzing the necessary trade-offs between accuracy and constraints in a representative inference task: *early application identification*. Network traffic inference tasks—such as anomaly detection, intrusion detection, and malware classification—demand timely and accurate traffic flow classification. High accuracy is crucial to prevent performance degradation and security risks. At the same time, inference pipelines must comply with operational constraints, such as time-to-decision (TTD) for real-time decision-making, and system constraints, such as memory limitations. However, our case study reveals that existing state-of-the-art approaches fail to effectively balance accuracy and efficiency.

2.1 Existing Methods

We evaluate existing methods, *i.e.*, inference pipelines, using only the first few packets per flow, as prior work shows this suffices for ML tasks [12, 20, 40, 46, 60, 94]. Our analysis (Section 3.1) confirms that additional packets offer minimal accuracy gains. A typical inference pipeline is composed of several key stages in inference pipelines, including preliminary data collection operations, such as capturing the traffic and dumping into pcaps, and more ML-oriented procedures, including preprocessing (from extracting raw features from the captured traffic to converting these features into a format suitable for models) as well as using trained models to perform inference on flows with the processed features. We aim to evaluate the accuracy and efficiency of the ML components, *i.e.*, the latter stages of the pipelines.

Flow-statistics-based Inference Pipelines (FSIP). FSIPs, inspired by heuristic-based techniques, infer traffic behavior using computed flow statistics such as throughput and inter-arrival times (IAT). One of the first papers to propose early application identification, Bernaille *et al.* [12] employs Gaussian Mixture Models (GMM) to cluster flows based on packet sizes (pkt_len) and IATs

[1] https://github.com/noise-lab/JITI



(a) Accuracy and TTD. (b) Memory usage.

Fig. 1. Inference pipeline performance with increasing traffic rates.

from the first four non-zero packets of a TCP connection. Conversely, more modern FSIPs, such as *SwiftIDS* (LightGBM) [48], *Lai20* (XGBoost) [53], and *Flowrest* (Random Forest) [6], leverage advanced ML models and a broader set of flow statistics (*e.g.*, duration, bytes, and packets, etc.) collected from the first handful of packets of a flow. Other approaches, like GGFast [73], perform more complex tasks to perform traffic classification. However, we exclude these pipelines from our analysis, as they require up to 50 packets per flow, contradicting our goal of real-time, early traffic identification.

Packet-capture-based Inference Pipelines. Recent ML-based traffic inference pipelines, particularly PCIPs, leverage raw packet captures (*e.g.*, pcaps) for deeper traffic analysis, uncovering hidden patterns that enhance inference accuracy. Prominent PCIPs include *Isingizwe21* (LightGBM) [41], *IIsy* (XGBoost) [101], and *pForest* (RF) [19], which use similar algorithms as FSIPs. Tree-based models remain also popular for network traffic classification due to their strong accuracy and lower training costs compared to deep learning approaches [7, 42]. There also exists CNN-based PCIPs, such as *Deep-Packet* [62] where they extract the first 1,500 bytes of each packet and use a majority vote across three packet-level inferences for final flow classification. We focus here on FSIPs and PCIPs as representative baselines, while broader discussions of efficiency-oriented methods, adaptive inference serving, and in-network traffic analysis systems are deferred to Section 5, since they provide contextual background rather than direct motivation.

2.2 Accuracy and Efficiency Trade-offs

We implement the presented approaches and evaluate their accuracy and efficiency tradeoffs. To avoid bias from particular implementation choices, we standardize raw feature extraction using *nPrint*[39] for packet captures and *dpkt*/*scikit-learn*[72] for flow statistics across all pipelines. Similarly, we adopt standard ML libraries, including *scikit-learn*, *MXNet*[22], and *PyTorch*[69], except for *Deep-Packet*, which requires its original sparse matrix preprocessor. This exception also helps us confirm the preprocessing step as the efficiency bottleneck across diverse preprocessing and model implementations. We evaluate the selected pipelines on a curated dataset of 20,000 flows from ten major applications, splitting it into 10,000 flows for training (80/20 train/validation) and 10,000 for testing. The testing traffic rate varies from 10 to 7,000 flows/second, sampled randomly from the test set. We analyze performance based on weighted F1-score, TTD, and in-use memory as an example system constraint. Here, *traffic rate* refers to the number of flows processed per second. For clarity, we report averaged results from the most balanced methods with minimum accuracy-efficiency trade-offs as calculated via harmonic means. We present detailed dataset descriptions and comparisons across all methods in Section 4.

Accuracy vs Operational Constraints. An operational constraint refers to metrics that gauge the processing efficacy of the inference pipeline, where failing to optimize them results in suboptimal efficiency rather than complete failure. An ideal inference pipeline aims to maximize accuracy while optimizing these operational constraints to ensure timely and efficient operations. In this paper, we focus on *time-to-decision (TTD)* as a key operational constraint. TTD measures the total time from the start of preprocessing to the final inference decision. While minimizing TTD is critical for

real-time applications, failing to do so does not break the system, but it may lead to delays that degrade the pipeline's effectiveness for downstream tasks. Figure 1a shows that while the FSIP *Bernaille06* achieves lower accuracy, it significantly outperforms PCIPs in TTD, with *DeepPacket* (CNN), a PCIP, taking 855x longer on average.

Key Observation ❶. Accuracy and efficiency trade-offs are evident, with higher accuracy incurring greater operational and system costs. For baseline pipelines, TTD grows roughly linearly with traffic rate because the per-flow processing time is relatively fixed and, without adaptive batching, flows are handled sequentially. As the arrival rate increases beyond the classifier's capacity, flows accumulate in the input queue, leading to proportional increases in TTD and, under persistent congestion, eventual flow drops. In this experiment, however, we run for a finite duration and assume unlimited memory, so the average TTD exhibits a linear growth with the input traffic rate. The primary bottleneck in PCIPs is feature preprocessing, which accounts for over 99% of *Isingizwe21*'s (LightGBM) TTD. In a 3,500-flow inference, only ~5% of TTD is spent on feature extraction and encoding, while 95% is dedicated to data aggregation (vector padding and batch merging). Feature preprocessing also drives system resource bottlenecks, such as memory overhead. PCIPs, requiring over 3,000 features from the first three packets per flow, contrast sharply with FSIPs, which use fewer than 10 features.

Accuracy vs System Constraints. A system constraint refers to resource limitations that an inference pipeline must strictly adhere to, as violations can cause failures or crashes. An ideal pipeline maximizes accuracy while staying within these constraints. We use *in-use memory* as an example, given its relevance across networked environments. Small to mid-sized devices, such as routers, operate within a few gigabytes of memory, while even large ISPs and data centers face challenges scaling to handle thousands of flows per second [10, 26, 35, 98]. This underscores the universal impact of system constraints, irrespective of infrastructure size. While PCIPs achieve higher accuracy (F1-Score of 0.959 on average), they demand significantly more memory than FSIPs. Figure 1b shows that PCIPs (*Isingizwe21* and *Deep-Packet*) consume 38× and 163× more memory, respectively, compared to the FSIP *Bernaille06*. Our system, while demonstrated using memory, applies to other constraints as discussed in Sections 3.3 and 4.4.

Key Observation ❷. More accurate models demand higher computational complexity, significantly increasing resource consumption and potentially causing resource exhaustion in constrained environments. As traffic rates increase, operational and system costs scale accordingly. Despite modern server capabilities, adherence to these constraints remains critical, as illustrated by baseline pipelines in the figures. Managing these trade-offs is essential for scalable, real-time traffic inference.

In this paper, we explore and balance these trade-offs by introducing a traffic inference system that retains the high accuracy of PCIPs while achieving better efficiency, approaching that of FSIPs.

2.3 Deployment Scenario and System Constraints.

We ground our work in the context of high-load access networks, such as large campus networks, which are representative of environments studied in prior work like FastFlow [8] and Traffic Refinery [15]. These access networks carry substantial bandwidth (*e.g.*, a 40 Gbps link in Traffic Refinery) but are particularly challenging because they exhibit peak new flow arrival rates of approximately 15,000 flows per second. We adopt this scale in our evaluation in Section 4, as it is corroborated by real-world backbone traffic statistics from the CAIDA Chicago monitor [1, 35], which report similar average rates. This workload setting places our design in the operational realities of production-scale access networks. In addition, we assume that these inference pipelines operate in environments with fixed, provisioned resources, *i.e.*, CPU and memory are not elastic. The operational challenge is therefore not about scaling out additional resources, but about dynamically adapting to varying traffic loads under static system capacities. This assumption emphasizes

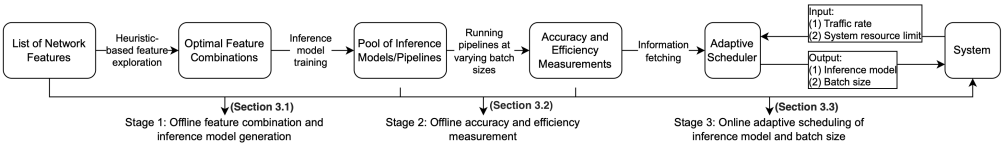


Fig. 2. Just-in-Time Traffic Inference (JITI) system overview.

the importance of balancing inference accuracy with efficiency while respecting strict system constraints, and it provides the core motivation for our adaptive approach.

3 Just-in-Time Traffic Inference

Our findings show the need for balancing accuracy and efficiency under variable system constraints is crucial to maintaining performance. We introduce Just-in-Time Traffic Inference (JITI), a system that maintains a curated pool of trained models with different feature specifications, allowing inference pipelines to swap models based on resource availability and traffic conditions. Additionally, JITI supports dynamic batch sizes, providing further adaptability to fluctuating traffic loads and constraints. As Figure 2 shows, the system design integrates a three-stage pipeline:

(1) Offline Bootstrapping: A heuristic feature exploration algorithm approximates balanced feature combinations, generating a pool of inference models with varying feature sets (Section 3.1).
(2) Offline Profiling: Models are assessed for accuracy and efficiency at different batch sizes for the target system (Section 3.2).
(3) Online Inference: An online adaptive scheduler monitors traffic rates and system constraints, dynamically selecting the optimal batch size and inference model/features to serve (Section 3.3).

3.1 Heuristics-based Feature Exploration and Inference Model Generation

The accuracy and efficiency of network traffic inference pipelines stems as much from the chosen inference model as from the input features. First, JITI identifies optimal feature subsets from a large candidate pool to generate a set of inference models with different accuracy-efficiency profiles.

Preliminary Feature Abstraction and Selection. Building a pool of models requires first choosing the set of candidate features to explore. These features are typically derived from domain expertise or determined by the capabilities of the traffic collection tool. In this implementation of JITI, we use *nPrint* [39] to encode raw packet captures into standardized bit-level features, leveraging its flexible feature subscription interface. In *nPrint*, each packet is normalized into a binary format that preserves its semantics through one-hot-encoding of header field values, providing an effective baseline for feature exploration. Yet, future implementations can replace *nPrint* with alternative encoders without affecting JITI's functionality. Encoding packet captures at the bit level significantly expands the feature space (*i.e.*, every bit in the first few packets represents a single feature), complicating efficient feature exploration. Standard ML-ready bit-level representations can exceed 1,000 features per packet (*e.g.*, UDP header bits are recorded even for TCP packets). To address this, we apply *network domain knowledge*, grouping bit-level features within the same semantic category (*i.e.*, identical header fields) into a single feature for selection. However, bit-level granularity is preserved for preprocessing, training, and inference. This abstraction reduces the feature space to 37 key features, including header fields and packet payloads as shown in Table 1.

We exclude from candidate features the packet payload, as its TLS encryption introduces noise that hampers inference. Similarly, we exclude features prone to *overfitting*, such as IP addresses and ports, due to their specificity to local networks and reduced generalizability. We focus on header fields from the first three packets of each flow. Initial packet headers capture critical details such as protocol type, session initialization, and metadata, and they provide sufficient accuracy

according to our tests and the literature. For instance, increasing to 10 packets per flow in *Isingizwe21* (LightGBM) [41] improves the F1-score by less than 0.4% while tripling the feature space. For flows with fewer than three packets, we pad missing bit-level features with −1. This selection results in 32 abstracted features for feature exploration and inference model generation as shown in Table 1.

Limitations and Alternatives to nPrint. Our choice of *nPrint* prioritizes maximizing accuracy from packet headers, but it introduces two considerations. The first is the potential negative impact that nPrint's fine-grained traffic feature representation can have on computational overhead, online profiling efficiency and TTD during online feature extraction. Lighter-weight alternatives, such as **packet series**—sequences of packet sizes, directions, and inter-arrival times—offer compact representations that can be advantageous in resource-constrained scenarios or for tasks where timing features are critical. In cases where packet-series features are preferable or better suited to the task, JITI can be applied to pipelines based on packet series as well. Another consideration is the risk of involving features that may lead to overfitting, since nPrint exposes a wide range of header fields and some (*e.g.*, checksums, TCP seq ack numbers) can act as shortcuts. However, users can decide which fields to include, so this is a risk to manage rather than a built-in limitation. In our study, we include them because our dataset spans multiple vantage points and a mix of services and operating systems, mitigating the risk of overfitting.

JITI is Representation-Agnostic. Regardless of the feature representation chosen or whether potential overfitting features are included or discarded, the JITI framework can operate seamlessly. Its core lies in the adaptive scheduling system, which selects models using pre-computed performance metadata—accuracy, TTD, and resource costs—without relying on raw traffic features. Offline profiling measures these metrics across candidate models, and the online scheduler selects the optimal model for the current context. Because this process relies solely on performance metadata, it can accommodate any feature type or granularity, including nPrint, packet series, or hybrid representations. This decoupling allows JITI to systematically select feature subsets from any representation. In this way, JITI provides a flexible framework that allows practitioners to tailor feature selection and model deployment to the specific requirements of their environment or task.

Heuristic-Based Feature Exploration. After selecting candidate features, JITI identifies optimal feature combinations to maximize (1) accuracy-to-operational-cost and (2) accuracy-to-resource-utilization ratios. A high accuracy-to-operational-cost ratio ensures efficiency (*e.g.*, minimizing TTD), while a high accuracy-to-resource-utilization ratio enables operation within strict resource limits (*e.g.*, memory or CPU capacity). A naïve approach to obtain these combinations would consist in exploring all possible feature combinations and select the top feature subsets that yield the highest ratios. However, this proves impractical, requiring the training and evaluation of $\sum_{k=1}^{32} \binom{32}{k} = 2^{32} - 1 \approx 4.29 \times 10^9$ models in our implementation. While non-exhaustive search methods exist, such as hill climbing [2, 34, 38, 83], they reduce search complexity but still require training and evaluating multiple intermediate models, often converging to suboptimal solutions due to local optima. For example, in our experiments, hill climbing yields F1-Scores 0.0834 and 0.0871 lower than XGBoost's and RF's global optima, respectively. Our experiments show that hill climbing frequently under-performs across all models, particularly for smaller feature sets.

We address this challenge by leveraging two key observations. First, there is a strong correlation (coefficient = 0.753) between permutation-based feature importance and the marginal accuracy gain observed when adding each feature to a model trained with all 32 features. This suggests that higher-importance features have a more direct and predictable impact on model accuracy. Second, feature size significantly affects efficiency, with a near-perfect correlation (coefficient = 0.99) between the total number of bits in a feature subset and the resulting overhead in both operational and system constraints. Larger feature sets increase resource consumption and reduce processing

Header	Field	Prelim. Sel.	Heur. Sel.	Header	Field	Prelim. Sel.	Heur. Sel.
ipv4	ver	✓	–	tcp	sport	–	–
ipv4	ihl	✓	–	tcp	dport	–	–
ipv4	tos	✓	✓	tcp	seq	✓	✓
ipv4	tot_len	✓	✓	tcp	ackn	✓	✓
ipv4	id	✓	–	tcp	doff	✓	✓
ipv4	frag_off	✓	–	tcp	res	✓	–
ipv4	ttl	✓	✓	tcp	flags: ns	✓	–
ipv4	proto	✓	✓	tcp	flags: cwr	✓	–
ipv4	cksum	✓	✓	tcp	flags: ece	✓	–
ipv4	sip	–	–	tcp	flags: urg	✓	–
ipv4	dip	–	–	tcp	flags: ack	✓	✓
ipv4	opt	✓	–	tcp	flags: psh	✓	✓
ipv4	dfbit	✓	✓	tcp	flags: rst	✓	✓
ipv4	mfbit	✓	–	tcp	flags: syn	✓	–
ipv4	rbit	✓	–	tcp	flags: fin	✓	✓
udp	len	✓	✓	tcp	wsize	✓	✓
udp	cksum	✓	✓	tcp	urp	✓	–
tcp	payload	–	–	tcp	opt	✓	✓
tcp	cksum	✓	✓				

Table 1. Abstracted header-level features after preliminary and heuristic selection.

Header	Field	Bits	FI	FI/Bits	Header	Field	Bits	FI	FI/Bits
ipv4	dfbit	1	0.048111	0.048111	tcp	cksum	16	0.007000	0.000438
tcp	fin	1	0.017778	0.017778	ipv4	tl	8	0.003444	0.000431
ipv4	ttl	8	0.121000	0.015125	tcp	opt	320	0.107000	0.000334
tcp	doff	4	0.032667	0.008167	udp	cksum	16	0.005222	0.000326
tcp	ackf	1	0.008111	0.008111	ipv4	tos	8	0.002333	0.000292
tcp	wsize	16	0.028556	0.001785	ipv4	proto	8	0.002222	0.000278
tcp	psh	1	0.001333	0.001333	tcp	rst	1	0.000111	0.000111
ipv4	cksum	16	0.009889	0.000618	tcp	seq	32	0.001444	0.000045
udp	len	16	0.007778	0.000486	tcp	ackn	32	0.000333	0.000010

Table 2. Summary of the remaining features after heuristics-based feature selection for pool of inference model generation.

efficiency. However, by carefully balancing feature importance and size, we can effectively mitigate these efficiency trade-offs.

Given these observations, we approximate the expected accuracy and efficiency trade-offs by optimizing the ratio of aggregated feature importance to feature size. This enables us to frame the feature selection process as a constrained optimization problem: $\max_{S \subseteq F} \sum_{i \in S} \frac{FI_i}{Bits_i}$, s.t.$|S| \leq n$, where F is the full feature set, S is a selected subset, FI_i represents the permutation feature importance of feature i, and $Bits_i$ denotes its corresponding bit size. To efficiently approximate the solution, we implement a heuristic-based algorithm (Figure 3) that ranks features by trade-off severity, constrains the search space, and selects efficient feature subsets. First, we compute the ratio $\frac{FI_i}{Bits_i}$ for each feature and rank them in descending order, prioritizing those that maximize accuracy while minimizing operational and resource costs. This greedy ranking criterion reduces the search space, ensuring *bounded complexity* over exponentially growing exhaustive search. Next, we control the pool size using two parameters: *sizes* (permissible feature set sizes) and num_{combos} (distinct subsets per size). Given $num_{combos} = k$ and $sizes = \{s_1, s_2, ..., s_m\}$, the total number of models is $pool\_size = k \cdot m$. This formulation efficiently samples the feature space, avoiding exhaustive evaluation on all possible subsets and requiring only a *single model training*, unlike hill climbing, which demands multiple intermediate trainings (*e.g.*, 45 model compilations for a 9-feature set), significantly reducing overhead. Finally, for each $s \in sizes$, we iteratively select the top num_{combos} subsets of size s that maximize the sum of $\frac{FI_i}{Bits_i}$. This ensures an adaptive balance between accuracy and efficiency under different system constraints. By leveraging strong correlations in feature importance and size, our selected subsets consistently yield models with F1-Scores exceeding 0.90, outperforming the often inconsistent hill climbing approach. Simultaneously, this guarantees a *balanced feature subset selection*, ensuring that selected models achieve high accuracy without excessive resource consumption. Table 2 provides a summary of the remaining features after heuristics-based selection.

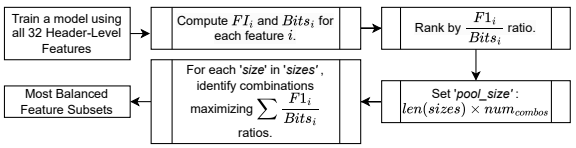


Fig. 3. Heuristic-based feature exploration.

Inference Model Generation. JITI utilizes the computed feature subsets to generate the inference models to pass to the next step in the pipeline. Our approach focuses solely on feature selection rather than model architecture. Thus, the choice of ML algorithm is secondary to its effectiveness. To ensure comprehensive evaluation, we implement three ML algorithms—LightGBM, XGBoost, and Random Forest (RF)—allowing direct comparison with baseline pipelines.

Algorithm 1 Adaptive Scheduler

Input: traffic rate, resource availability; **Data:** accuracy and efficiency measurements
$candidates \leftarrow \emptyset$ ▷ Initialize an empty list
for each $combination$ in Data **do**
  $N_t(B_t, R_t, TTD_C) \leftarrow \lceil \frac{TTD_C(B_t)}{B_t/R_t} \rceil$ ▷ Maximum concurrency
  $RES_t(N_t, RES_C) \leftarrow N_t \cdot RES_C$ ▷ Combination system resource requirement
  if $RES_t(N_t, RES_C) \leq$ System resource availability **then**
    $candidates$.append($combination$)
if len($candidates$) $\neq 0$ **then**
  $opt\_combination\ (\kappa_{opt}) \leftarrow candidates[0]$
  $opt\_score \leftarrow \frac{F1\text{-}Score(\kappa_{opt})}{TTD_C(\kappa_{opt}, B_t)}$
  for each $candidate(\kappa)$ in $candidates[1:]$ **do**
    $current\_score \leftarrow \frac{F1\text{-}Score(\kappa)}{TTD_C(\kappa, B_t)}$
    if $current\_score > opt\_score$ **then**
      $\kappa_{opt} \leftarrow \kappa$ ▷ Update most balanced combination
return $opt\_combination$

3.2 Accuracy and Efficiency Measurements at Varying Batch Sizes

After generating the inference models, JITI profiles the models and determines their accuracy and efficiency at different batch sizes and number of classifier instances on the target system.

Adaptive Batch Size. Batch size, defined as the number of traffic flows captured before initiating an inference pipeline instance and serves as a key hyperparameter for optimizing efficiency. While prior work [27, 58, 102] has optimized batch size to minimize model execution time, existing approaches do not dynamically adjust batch size to balance operational and system constraints. However, simply linearly increasing batch size proportionally to traffic rate can lead to excessive overheads at high traffic volumes. In contrast, an adaptive approach that dynamically adjusts batch size based on real-time traffic conditions, forecasting fluctuations and processing traffic without waiting for a complete second, thereby reducing operational constraints.

Number of Inference Instances. While increasing batch size can improve system throughput, it is not an effective strategy for reducing time-to-decision. Although larger batches decrease per-inference processing time, the overall TTD remains limited by the time needed to collect all inputs before inference begins. However, if sufficient system resources (*e.g.*, memory, CPU) are available, leveraging multi-core architectures and running multiple inference instances in parallel can overcome this bottleneck. By distributing inference across multiple pipeline instances, the system can process inputs concurrently, significantly reducing TTD and improving efficiency.

Profiling of System Resource Requirements. Given inference models, maximum batch sizes and number of parallel instances, JITI collects offline measurements to build a accuracy-efficiency map for available models. These measurements inform the adaptive scheduler (presented in the next section), which selects the optimal batch size and model configuration. For any batch size B and inference pipeline C, we define: (1) End-to-end processing time: $TTD_C(B)$; and (2) the resource requirement for a classifier instance: $RES_C(B)$. Additionally, we take into consideration that multiple inference instances may run in parallel if the pipeline's TTD exceeds the batch fill-up time and sufficient system resources (*e.g.*, memory, CPU) are available. In this case, the total resource requirement is the aggregate needed to sustain concurrent execution without failure or delay. Thus, given batch size B_t and traffic rate R_t at time T_t, the maximum number of concurrently running instances of C is $N_t(B_t, R_t, TTD_C) = \left\lceil \frac{TTD_C(B_t)}{B_t/R_t} \right\rceil$. The total system resource requirement is then $RES_t(N_t, RES_C) = N_t \cdot RES_C$. For any inference pipeline instance with a trained model and fixed batch size, its resource demand can be computed accordingly. For example, in our evaluation of FSIPs and PCIPs, where batch size is set equal to traffic rate ($B_t = R_t$), the total system resource requirement simplifies to $\lceil TTD_C(B_t = R_t) \rceil \cdot RES_C(B_t)$.

3.3 Adaptive Scheduler

Given the pool of inference models (each with different feature sets) and their corresponding efficiency and accuracy measurements, we introduce an adaptive scheduler that continuously monitors traffic rate and resource availability to dynamically select the model and batch size that optimize the trade-off between accuracy and efficiency.

Selection Objective. For a given traffic rate R_t, the scheduler seeks to determine the optimal combination of inference model C and batch size B that maximizes accuracy while minimizing operational constraints. Formally, this can be expressed as $(C^*, B^*) = \text{argmax}_{C,B} \frac{ACC_C(B)}{OC_C(B)}$ subject to system resource constraints $(RES_t(N_t, RES_C) \leq RES_{max})$, where $ACC_C(B)$ represents the accuracy of model C at batch size B, $OC_C(B)$ is the operational cost, and $RES_t(N_t, RES_C)$ denotes the total system resource requirement. The scheduler employs an iterative optimization approach, evaluating accuracy-efficiency trade-offs across the available configurations to select the optimal combination.

Minimum F1-Score Requirement (μ). The scheduler prioritizes maximizing the accuracy-to-operational-cost ratio but does not inherently enforce a minimum accuracy threshold. In real-world deployments, a minimum acceptable F1-Score (MFR) may be required. To accommodate this, we introduce a tunable parameter μ, ensuring that only model-batch combinations satisfying $ACC_C(B) \geq \mu$ are considered: $(C^*, B^*) = \text{argmax}_{\substack{C,B \\ ACC_C(B) \geq \mu}} \frac{ACC_C(B)}{OC_C(B)}$. If no combination meets μ, the scheduler selects the configuration with the highest achievable accuracy. The default $\mu = 0$ implies no MFR, and the accuracy metric can be replaced with alternative evaluation criteria in future implementations. Algorithm 1 outlines the scheduler's operation after enforcing the MFR. Given system constraints, the scheduler evaluates resource requirements at the current traffic rate and eliminates infeasible configurations. For instance, if memory availability is 5 GB and traffic rate is 1,500 flows/sec, a combination with TTD = 1.2s, batch size = 500, and unit memory requirement = 2 GB is discarded, as its total memory demand ($\left\lceil \frac{1.2}{500/1500} \right\rceil \times 2 = 7.2$ GB) exceeds availability. After filtering, the scheduler iteratively selects the configuration maximizing the accuracy-to-operational-cost ratio. The chosen inference model and batch size dictate traffic inference, dynamically adapting as traffic rates and resource availability fluctuate. Designed for practical deployment, our implementation of the scheduler operates efficiently within 37 MB of memory and makes decisions in under 10 milliseconds. Since it processes only precomputed tabular data, increasing the model pool size has minimal execution overhead.

4 Evaluation

We evaluate the performance of JITI against state-of-the-art inference pipelines. We focus on the early application identification task previously introduced, and perform flow-level application identification on a network traffic flow dataset. Our experiments show that, under both resource-rich and resource-scarce environments, JITI balances accuracy and efficiency, achieving better inference results than FSIPs while maintaining better operational constraints, *i.e.*, lower TTD, than conventional PCIPs. Further, we explore the robustness of the system by profiling its behavior under varying traffic rates and system constraints, using memory and CPU availability as examples.

4.1 Evaluation Setup

Prototype, Baselines, and Experimental Setup. We implement JITI's prototype, as well as all baselines introduced in Section 2 for comparison. We implement raw feature extraction using *nPrint* [39] for packet captures and *dpkt*/*scikit-learn* [72] for flow statistics. We adopt standard ML libraries, including *scikit-learn*, *MXNet* [22], and *PyTorch* [69], except for *Deep-Packet*, which requires its original sparse matrix preprocessor. Finally, we conduct all experiments on a machine

Macro Services	Total Flows	Application Labels (Flows)	Collection Date
Video Streaming [16]	9465	Netflix (4104), YouTube (2702), Amazon (1509), Twitch (1150)	2018-06-01
Video Conferencing [65]	6511	MS Teams (3886), Google Meet (1313), Zoom (1312)	2020-05-05
Social Media	3610	Facebook (1477), Twitter (1260), Instagram (873)	2022-02-08

Table 3. Summary of the network traffic flow dataset: 10 applications across three macro service types.

Type	Method	F1	TTD (s)
FSIPs	Bernaille06 [12]	0.39	0.04
	Lai20 [53]	0.72	1.08
	Flowrest [6]	0.72	0.63
	SwiftIDS [48]	0.67	0.06
PCIPs	**JITI(XGB)**	**0.85**	**1.66**
	JITI(RF)	**0.86**	**0.49**
	JITI(LGB)	**0.86**	**0.24**
	IIsy [101]	0.95	95.13
	pForest [19]	0.94	96.33
	Isingizwe21 [41]	0.94	96.10
	Deep-Packet [62]	0.98	30.50

Table 4. Evaluation: JITI achieves best tradeoff in F1 and TTD.

running Ubuntu 20.04.4 LTS, Linux 5.4.0-135-generic kernel, and x86-64 architecture. It features an AMD EPYC 7502P 32-Core Processor with 64 threads, operating at 1.49GHz base and 2.5GHz boost clock speeds. All inference pipelines in this study default to distributing workloads across available cores for parallel computing.

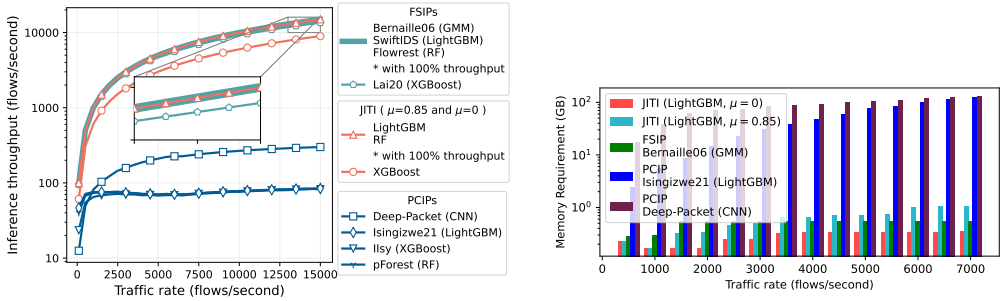
Operational and System Resource Constraints. To replicate real-world requirements, we enforce both operational and system resource constraints. As an operational constraint, we focus on *time-to-decision (TTD)*, which measures the responsiveness of the inference pipeline. Minimizing TTD is crucial for real-time systems, where processing delays can lead to suboptimal results. However, exceeding the TTD threshold does not necessarily cause system failure. For system resource constraints, we consider two key factors. First, memory requirements serve as a primary example. Memory optimization is particularly important in traffic inference, where real-time systems process large data volumes. This includes memory used during inference, including intermediate storage for preprocessing (including raw feature extraction and transformation into ML-compatible formats), model loading, and inference. We measure memory usage (including both RAM and swap space) using Linux `cgroup` controls and ensure stable pipeline operation without excessive TTD or premature termination. For concurrent instances, we compute the total memory requirement, as discussed in Section 3.2. Second, we demonstrate that JITI can readily accommodate additional constraint metrics by profiling its behavior under varying CPU capacity constraints.

Dataset. We evaluate using a curated dataset (Table 3) comprising pcap files collected over multiple dates, covering traffic flows from ten applications across *video streaming* [16], *video conferencing* [65], and *social media*. These traces are widely used for networking tasks such as latency analysis, QoE inference, and application identification. To preprocess the dataset, we: (1) Extract DNS queries to identify application-related IP addresses. (2) Filter traffic to retain only packets associated with these IPs. (3) Segment traffic into individual flows based on the 5-tuple attributes (source/destination IPs and ports, transport protocol). Each flow is labeled with its respective application (*e.g.*, Netflix, Zoom) for inference accuracy evaluation. Since all traffic is TLS-encrypted, we aggregate flows into a unified dataset for benchmarking JITI and baseline pipelines.[2]

4.2 Accuracy in Unconstrained Environments

We first evaluate JITI's accuracy and efficiency in resource-rich settings, performing application identification on traffic flows ranging from 100 to 15,000 flows/second, following the deployment scenario described in Section 2.3. To emulate different traffic rates, we randomly sample and aggregate flows from the test set. We compare JITI's accuracy against both the FSIPs and PCIPs

[2]All traces used in this paper are sanitized and contain no personally identifiable information (PII).



(a) Without memory constraint: JITI (LightGBM, RF) matches upper-bound throughput, surpassing conventional PCIPs.

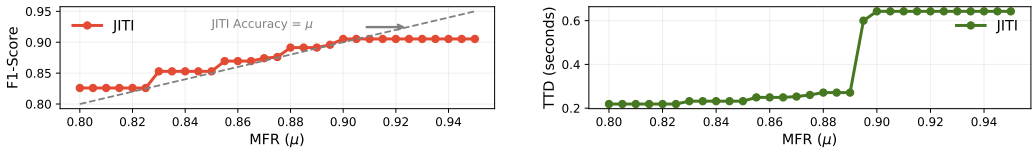
(b) Memory requirement comparison of JITI vs. conventional inference pipelines across 500–7000 flows/sec.

Fig. 4. Throughput and memory usage comparison of JITI against baselines under increasing traffic rates.

detailed in Section 2. We consider feature set sizes ranging from one to nine, with 10 combinations per size—yielding a pool of 90 inference models. The feature set size is capped at nine, as additional features introduce high overhead with diminishing accuracy gains. With finer-grained abstraction, this limit can be adjusted. Example inference models from this pool are in Appendix A, Table 5.

JITI Outperforms PCIPs in TTD. Table 4 shows that all JITI implementations significantly reduce TTD compared to PCIPs, approaching FSIP levels. The fastest JITI implementation (LightGBM) achieves a TTD of 0.24s, over 2X faster than *Deep-Packet* (CNN), the fastest PCIP. To interpret TTD in a network-centric way, we express it as *inference throughput*—the number of flows an inference pipeline processes per second post-capture. Throughput is derived as $\frac{traffic\ rate}{\max(TTD,1)}$, where lower TTD increases throughput. For instance, if 7,000 flows are captured per second with a TTD of 1.4s, inference throughput is 5,000 flows/s, covering 71.4% of the traffic rate. Figure 4a shows that JITI (LightGBM, RF) matches the FSIP throughput upper bound and outperforms PCIPs by nearly 50x at 15,000 flows/s. With μ set at 0.85 and 0, JITI maintains high throughput by dynamically selecting models with optimal accuracy-to-operational-cost ratios and running as many instances of the chosen inference pipeline (with the selected inference model) as required since no memory constraint is enforced. This allows JITI to scale with traffic demand, sustaining near-maximal throughput when resources permit. Although the inference throughput (*i.e.*, the number of flows preprocessed and predicted within a second post data collection) may reach 100%, the actual TTD may be less than one second. As Table 4 indicates, there are cases where our system implementation meets the upper bound throughput but has a higher average TTD than the FSIPs. We carry out analysis on TTD variations in Section 4.4. Despite this, all JITI implementations still surpass all evaluated PCIPs with considerably lower average TTDs. In some cases, XGBoost-based inference models/pipelines, including the JITI implementation, show reduced throughput, *i.e.*, higher average TTD, than those using other algorithms. This agrees with past work [13, 21, 50, 74], suggesting that XGBoost models tend to exhibit slower inference speeds given high traffic volume and complexity.

JITI Achieves Higher Accuracy Than the FSIPs. We evaluate JITI's inference accuracy against existing pipelines. As expected, PCIPs generally achieve higher F1-Scores than FSIPs, with *Bernaille06* (GMM) performing the worst at 0.39. With μ set to 0.85, the most accurate JITI implementation (LightGBM) achieves an average F1-Score of 0.86—just 0.12 below *Deep-Packet* (CNN), the best-performing PCIP (Table 4). All JITI variants outperform the best FSIP by at least 18%, and even with $\mu = 0$, JITI maintains a strong average F1-Score of 0.82—13.9% higher than the FSIP baseline.



(a) JITI improves accuracy as MFR (μ) increases, until no model in the pool can satisfy the requirement.

(b) Higher MFR (μ) triggers selection of more complex models from the pool, increasing TTD.

Fig. 5. JITI's performance sensitivity to changes in MFR.

To understand how JITI adapts under varying accuracy demands, we conduct a detailed sensitivity analysis of the minimum F1-score requirement (MFR, μ), with results shown in Figures 5a and 5b. We incrementally varied μ from 0.80 to 0.95 (in 0.005 steps) under a traffic rate of 15,000 flows/second and no memory constraints. As illustrated in Figure 5a, increasing μ prompts the adaptive scheduler to select the most accurate model from the pool that satisfies the current threshold while minimizing computational cost. In our experiments, JITI is able to meet accuracy requirements up to an F1-Score of 0.91, beyond which no available model satisfies the target. In such cases, the scheduler defaults to the most accurate model in the pool. This limitation can be alleviated by increasing the $pool_size$ to include models with richer feature sets. Figure 5b further demonstrates that higher μ values inevitably lead to increased TTD, as the system selects more complex models with higher feature-processing overhead. Finally, we compare JITI to fixed-feature LightGBM-based PCIPs in Appendix A.1, showing that JITI consistently achieves a superior balance between accuracy and efficiency through its adaptive model selection and scheduling. These results highlight that, absent system constraints, JITI outperforms PCIPs in TTD/throughput, *i.e.*, operational constraint, and FSIPs in inference accuracy.

4.3 Suitability for Resource-Constrained Environments

Our results show that in unconstrained settings, the fastest JITI implementations achieve low TTD (100% throughput) and high accuracy across all evaluated traffic rates. We now assess whether JITI maintains this balance under limited system resources, using memory as the primary constraint. The evaluation ranges from 500 to 7,000 flows/second and we test JITI both with and without setting an explicit minimum F1-Score requirement.

JITI Requires Significantly Less Memory Than PCIPs. Figure 6 shows that JITI consistently requires far less memory than PCIPs across all traffic rates, reducing the risk of memory exhaustion. The most efficient implementations (LightGBM, RF) even use less memory than *Bernaille06* (GMM), an FSIP. Compared to *SwiftIDS* (LightGBM), the least memory-intensive FSIP, JITI requires only 1.7x more memory at 7,000 flows/s, while the most efficient PCIP (*Isingizwe21*) demands 974x more. JITI's efficiency stems from its heuristic-based inference model generation, which optimizes operational cost and resource use. The scheduler selects models that minimize memory consumption, making JITI highly suitable for memory-constrained deployments. Fine-grained memory comparisons of example pipelines are in Figure 4b.

Imposing a Minimum F1-Score Requirement Increases Memory Requirements. Allowing JITI to operate without a minimum F1-Score requirement (μ) minimizes memory usage but results in lower accuracy (F1-Score = 0.82). Setting $\mu = 0.85$ increases memory demand to sustain high throughput (Figure 6), as more accurate models require additional features, increasing memory consumption during preprocessing and inference. However, JITI's memory use remains comparable to FSIPs and far lower than PCIPs. At 7,000 flows/s, a LightGBM-based JITI implementation with $\mu = 0.85$ still requires ~120x less memory than PCIPs.

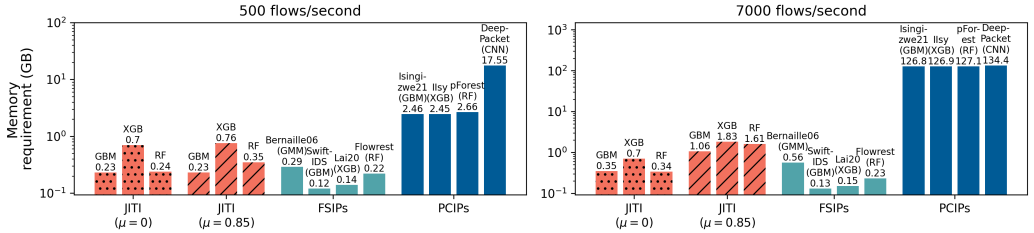
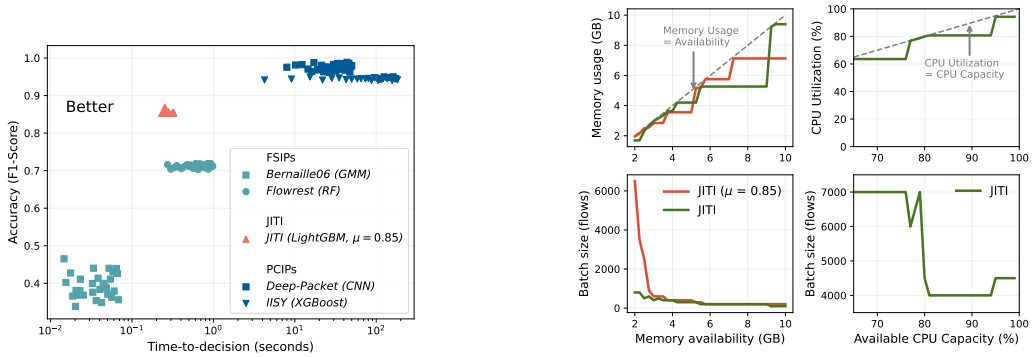


Fig. 6. JITI requires less memory than PCIPs at both low and high traffic rates.



(a) At 100% throughput, JITI balances trade-offs better than FPIPs and PCIPs across traffic rates.

(b) JITI behaviors under fixed traffic rate and varying system resources.

Fig. 7. Evaluation of JITI's trade-off handling and behavioral adaptability.

JITI Balances Accuracy and Efficiency with 100% Throughput. We further assess JITI's ability to balance accuracy and efficiency under memory constraints, evaluating F1-Score and TTD at traffic rates up to 15,000 flows/s. Focusing on the LightGBM-based JITI with $\mu = 0.85$, we find it achieves the best trade-off between inference throughput ($\frac{traffic\ rate}{TTD}$) and accuracy. Using the same selection methodology, we compare JITI to *Flowrest* (RF) and *IIsy* (XGBoost) as representative FSIP and PCIP baselines, along with *Bernaille06* (GMM) and *Deep-Packet* (CNN). As Figure 7a shows, JITI consistently maintains high F1-Scores and low TTD (high throughput), unlike baselines that trade off one metric for another. By dynamically selecting inference models and batch sizes, JITI optimizes for both accuracy and efficiency without compromising either.

4.4 Behaviors Under Varying System Constraints and Traffic Rates

Finally, we evaluate JITI behaviors under changing memory constraints and traffic rates to reflect realistic deployment where traffic rates (and hence memory demands) change during the course of a day. To simplify presentation, we use the JITI implementation with LightGBM for subsequent analysis. Observed behaviors generalize across JITI implementations as the choice of ML algorithms doesn't significantly affect its accuracy and efficiency, as shown in Table 4.

JITI Adjustments Facing Varying Memory Availability. To assess the influence of memory accessibility on JITI's behavior, we conduct experiments at a fixed traffic rate of 15,000 flows/second while gradually increasing the memory available to JITI. The selection of the traffic rate is arbitrary as the goal of this experimental setup is for us to isolate and discern the relationship between memory availability and the system's behavior. In all subsequent results, the *inference throughput maintains a consistent 100% of the incoming traffic rate.* Experiments are conducted with MFR (μ) set to 0 and 0.85 in F1-Score.

As shown in Figure 7b, regardless of whether the MFR is set, JITI exhibits gradual increases in memory utilization and reductions in batch size as the system memory availability increases. This is in line with the scheduler's objective to maximize the accuracy-to-operational-cost ratio because reducing the batch size and choosing inference models with smaller feature sets can both lead to a decrease in TTD. However, this behavior results in an increase in memory usage, as a reduced batch size results in more concurrent inference pipeline instances. At an MFR of 0.85, batch sizes are typically larger. In this case, the inference model selection process becomes less flexible due to the necessity of selecting models with better accuracy and, consequentially, more memory usage per pipeline instance. As a result, the system opts for larger batch sizes to reduce the number of concurrently running instances and to conform to the available memory constraints. Also note that the observed memory utilization exhibits jumps instead of a smooth increase with available memory. This behavior stems from the granularity of evaluation batch sizes; finer-grained batch sizes in real implementations can rectify this and avoid potential memory under-utilization.

JITI Adjustments Facing Varying Traffic Rate. Following the same approach, we examine the effects of traffic rate on JITI by gradually increasing the traffic rate from 100 to 15,000 flows/second while fixing the memory availability at an indicative capacity of 10 GB (similar results are observed at different memory capacities). We observe that the system increases its batch size as the traffic rate grows as shown in Figure 9a. Without additional memory, heightened traffic rate can lead to excessive concurrent pipeline instances, risking memory exceedance. To mitigate this, JITI's adaptive scheduler increases batch size, curtailing concurrent instances. Similarly, when MFR is set, batch sizes rise to compensate for the larger feature sets and memory consumption in selected inference pipeline. We also show that TTD rises with increasing batch size and traffic rate as expected. Yet, even at a peak traffic rate of 15,000 flows/second, TTD remains well below 1 second, significantly outperforming traditional PCIPs.

4.5 Analysis of JITI's Performance Under Variable Load.

We analyze JITI's adaptive behavior as traffic rate increases from 500 to 50,000 flows/s. Figure 8 shows how F1-score, batch size, memory, TTD, and the resulting optimization score evolve under rising load (with memory capped at 7 GB for visualization).

At low rates (500–4k flows/s), JITI selects a single classifier: F1≈0.826, batch size 30, and TTD≈0.22s. Memory grows linearly from 6.7 GB, nearly reaching the limit. The optimization score, Score = $\frac{\text{F1-Score}(B)}{OC_C(B)}$, remains high (∼3.77). JITI does not simply pick the most accurate model, but the one maximizing this ratio. At 4.5k flows/s, a larger batch (40) allows a slightly more accurate classifier (F1≈0.827), but higher TTD (0.23s) lowers the score to ∼3.64. In the mid-range (5k–20k flows/s), batch size grows (60–100), memory again nears the ceiling, and TTD rises (∼0.25s). Around 11k flows/s, JITI switches to a faster but less accurate classifier (F1 drops to 0.77–0.74), improving TTD slightly (0.23–0.25s) but reducing the score (3.35⟶3.13). At 20.5k flows/s, a previously infeasible high-accuracy model fits under batch size 300, restoring F1 to ∼0.826 but with TTD≈0.29s and score≈2.86. The scheduler thus optimizes F1-to-cost balance, not F1 alone. At high rates (20k–50k flows/s), JITI alternates between accurate and efficient models depending on batch feasibility. Around 27.5–30k, larger batches (300–400) lift F1 to ∼0.77 but increase TTD to 0.30–0.31s, dropping the score to ∼2.61. Beyond 30k, batch size escalates (up to 800), memory saturates near 7 GB, F1 declines to ∼0.69, and TTD stabilizes near 0.33s, causing a gradual score decay.

In short, the figure shows why the optimization score must decline monotonically with traffic: $OC_C(B)$—the operational cost dominated by TTD and memory overhead—grows with load, while $ACC_C(B)$—the F1-Score—can at best stabilize or partially rebound. As traffic increases, certain classifier/batch-size combinations that previously maximized the score become infeasible under the

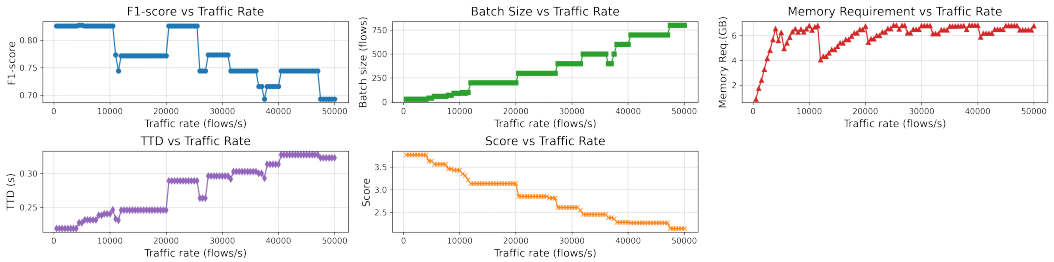
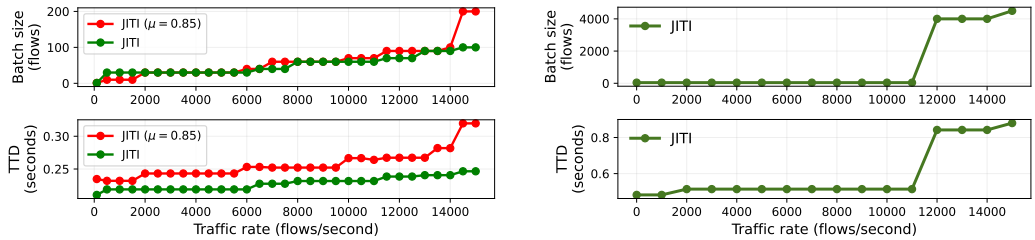


Fig. 8. JITI evaluation across traffic rates (500–50k flows/s): (a) F1-Score, (b) batch size, (c) memory usage, (d) time-to-decision (TTD), and (e) selection score.



(a) JITI increases batch size at higher traffic rates, increasing TTD to stay within memory limits.

(b) With a 100% CPU limit, JITI also increases batch size and TTD to conform to CPU availability.

Fig. 9. JITI's adaptive batching behavior under memory and CPU constraints at increasing traffic rates.

memory constraint, forcing the scheduler to defer to the best remaining feasible option. This is not a "bad choice," but the intended behavior of JITI, which continuously re-optimizes the *ACC*/*OC* trade-off to maintain feasibility and efficiency under system limits. Occasional rebounds in F1 occur when, at higher load, batching shifts resource usage so that a more accurate classifier–batch size pair once again becomes the best feasible option. Nevertheless, the overall optimization score necessarily drifts downward, correctly reflecting the growing cost of inference at scale.

4.6 Generalization to Additional Resource Constraints

While memory serves as the primary constraint in our evaluation, JITI readily adapts to other system resources such as CPU capacity. To demonstrate this, we evaluate a Random Forest (**RF**) implementation of JITI under CPU constraints. As Figure 7b shows, under fixed traffic rate, JITI adjusts batch sizes and model selections in response to varying CPU availability, maintaining optimal TTD without exceeding limits. The trends mirror those under memory constraints, underscoring JITI's robustness. As CPU usage nears saturation, the system proactively enlarges batches to reduce concurrent pipelines and prevent over-utilization, sustaining low TTD within resource limits.

Moreover, Figure 9b demonstrates how JITI handles varying traffic rates when CPU capacity is fixed. As traffic rates increase, JITI intelligently scales up the batch size to maintain efficient throughput, thereby preventing CPU saturation. Despite these increases, the TTD remains well within acceptable bounds, further confirming the efficiency of the scheduler in managing resource constraints. Memory and CPU are just examples of system constraints. Thanks to its modular design, JITI can extend to other limitations such as bandwidth or storage without major changes. Its adaptability comes from decoupling inference model selection from resource monitoring, allowing any metric to be incorporated into scheduling. The core functions—dynamic batching and optimal model selection—remain intact, ensuring balanced accuracy and efficiency across environments.

5 Related Work

Traffic inference has been a long-standing objective for researchers and practitioners. Early methods often relied on heuristics such as TCP/UDP port numbers and payload signatures [3, 66, 78, 90, 100]. With the availability of large network traffic datasets and increased computing power, two prominent ML-based method categories emerged: flow-statistics-based and packet-capture-based. FSIPs train inference models using information from network flows, such as packet counts and sizes, flow duration, and inter-arrival time [12, 49, 54, 55], or statistics from tools such as NetFlow [24] and IPFIX [25]. While efficient in terms of operational constraint, *e.g.*, TTD, and system constraints, *e.g.*, memory usage, they are relatively brittle due to the constantly evolving nature of the Internet [14, 47, 59, 67, 80, 93, 97]. Changes over time can alter the landscape of used traffic features, as domain expert-crafted features are static and hard to update, especially with increasing level of encryption making them unavailable or inaccurate. JITI aims to achieve similar efficiency of these methods while overcoming their inherent limitations. Recent advancements in computing and ML have fueled representation learning in traffic inference, offering inference pipelines that do not require domain experts for feature identification [11, 12, 18, 28, 49, 62, 79, 87, 103]. These approaches, often more accurate and robust, excel in detecting complex patterns in detailed packet-level features, even amidst encryption. Yet, the higher inference accuracy comes at the cost of increased overhead: preprocessing raw network data for ML model compatibility is both time and memory-intensive [18, 28, 87, 103]. JITI achieves similar accuracy to these methods while overcoming resource inefficiencies.

Beyond FSIPs and PCIPs, there also exists efficiency-oriented methods. Existing low-latency network traffic inference pipelines mostly focus on enhancing efficiency by reducing model complexity through lightweight models or simpler feature representations [27, 51, 56, 57, 73, 75, 91]. Others use ensemble techniques or tree-structured ML to improve inference efficiency using diverse families of ML algorithms [30, 33, 61]. These efforts focus primarily on model execution in inference pipelines, but lack emphasis on optimizing inference pipelines for computational resources such as memory, affecting their practicality. Current solutions often rely on specialized hardware like P4 switches, which constrains generalizability [82, 88, 89].

Lastly, two additional strands of research provide useful context for our work: (1) general-purpose adaptive inference serving systems that dynamically switch models or configurations, and (2) network-specific analysis frameworks that explore in-network or runtime adaptation. Adaptive inference serving frameworks dynamically select models or execution paths to meet resource or latency constraints. Representative systems include Clipper [27], Jellyfish [68], and CascadeServe [52], which switch among models or configurations at runtime to balance accuracy and efficiency. While effective in generic ML serving contexts, these approaches are not designed for network traffic inference, which must process high-throughput packet streams under strict operational and system constraints. JITI differs by co-optimizing both the inference model (via feature set selection) and batch size, tailored for traffic analysis workloads. Another line of work integrates machine learning directly into network-specific inference systems. Jewel [5], for instance, combines packet- and flow-level information but depends on programmable switch hardware (P4), limiting deployability. Other systems such as FENXI [36] and Leo [43] introduce adaptive mechanisms but only partially address the design space: FENXI supports adaptive batching without model switching, while Leo provides runtime model switching without batch-level adaptation. In contrast, JITI is a software-based solution on commodity hardware that unifies both adaptive batching and model selection, enabling constraint-aware inference in high-load access networks.

6 Discussion and Future Work

Beyond Packet Headers. Our evaluation has focused on packet header features to validate JITI. Incorporating additional derived or cross-layer features could further improve inference accuracy and efficiency. Future work will test JITI's robustness under such extended feature spaces to provide deeper insights into its effectiveness across diverse networked systems.

Ensemble Models. The flexible feature sets of JITI allow selecting models best suited to chosen inputs, and integrating diverse model types and architectures may improve efficiency and accuracy. While preprocessing efficiency is our main goal, ensemble approaches remain a promising extension.

Model Retraining and Offline Cost. Retraining to handle new inference tasks and data/model drift is outside our scope but well studied [37, 63, 96]. Our focus is efficient deployment of trained models. Although JITI's offline optimization steps can be computationally intensive, they are essential for tuning to deployment environments and do not affect real-time inference.

Generalization to System Constraints and Models. The scheduler can optimize accuracy-to-operational-cost ratios for any measurable resource metric and filter out combinations that violate constraints, including multiple simultaneous ones through extra condition checks. Its modular design ensures adaptability to diverse models and environments, as demonstrated in Section 4.4 using memory as the main constraint and CPU capacity as an example.

Encrypted Network Traffic. Highly encrypted traffic (*e.g.*, QUIC, VPN) obscures packet contents, making ML-based inference harder. Evaluating JITI under different encryption levels is promising, as its adaptable feature sets may mitigate risks of overfitting to limited observable features.

Dataset Limitations. Curated datasets may be outdated or misrepresent Internet topology, as seen with ISCX VPN-NonVPN [32], CAIDA [1], and QUIC [92]. Many also lack flow-level application/service labels critical for inference evaluation. Future work should assess JITI with real-time, labeled traffic data to strengthen practical applicability.

Accuracy Guarantees Under Peak Load. A natural concern in real-world deployments is the acceptability of accuracy losses during peak traffic conditions, precisely when traffic classification is most critical. JITI partially addresses this by allowing deployments to specify a minimum F1-score requirement (μ). The MFR is optional because different deployments may prioritize efficiency over accuracy, or conversely demand strict accuracy floors. By making μ tunable rather than fixed, JITI ensures that operators can explicitly control this trade-off. However, using MFR also limits the usable model set, which can cause JITI to become load-limited if load exceeds what this subset of models can support—a potential direction for future research.

7 Conclusion

Network traffic inference at varying traffic rates with limited resources is challenging for a spectrum of network management tasks. Conventional flow-statistics-based methods are efficient but struggle with accuracy as the Internet evolves, whereas raw packet-capture-based inference pipelines boost accuracy but are slow and resource-intensive. This paper introduces the Just-in-Time Traffic Inference (JITI) system, which balances inference accuracy and efficiency under different traffic rates while meeting system constraints. We develop an efficient exploration algorithm to create a pool of inference models and an online adaptive scheduler to choose the most appropriate model and batch size considering resource availability and traffic rate. Evaluation shows that the system (1) performs better (>18% higher in F1-Score) than flow-statistics-based methods, and (2) processes traffic ~127x faster compared to conventional packet-capture methods. The system's potential extends beyond traffic inference, with prospects for addressing other tasks that rely on ML-based inference in networked systems.

References

[1] 2019. The caida anonymized internet traces dataset (April 2008 - January 2019). https://www.caida.org/catalog/datasets/passive_dataset/

[2] Laith Mohammad Abualigah, Ahamad Tajudin Khader, Mohammed Azmi Al-Betar, Zaid Abdi Alkareem Alyasseri, Osama Ahmad Alomari, and Essam Said Hanandeh. 2017. Feature selection with β-hill climbing search for text clustering application. In *2017 Palestinian International Conference on Information and Communication Technology (PICICT)*. IEEE, 22–27.

[3] Giuseppe Aceto, Alberto Dainotti, Walter De Donato, and Antonio Pescapé. 2010. PortLoad: taking the best of two worlds in traffic classification. In *2010 INFOCOM IEEE Conference on Computer Communications Workshops*. IEEE, 1–5.

[4] Iman Akbari, Mohammad A Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. 2021. A look behind the curtain: traffic classification in an increasingly encrypted web. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 1 (2021), 1–26.

[5] Aristide Tanyi-Jong Akem, Beyza Bütün, Michele Gucciardo, and Marco Fiore. 2024. Jewel: Resource-efficient joint packet and flow level inference in programmable switches. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 1631–1640.

[6] Aristide Tanyi-Jong Akem, Michele Gucciardo, Marco Fiore, et al. 2023. Flowrest: Practical Flow-Level Inference in Programmable Switches with Random Forests. In *IEEE International Conference on Computer Communications*.

[7] Ons Aouedi, Kandaraj Piamrat, and Benoît Parrein. 2021. Performance evaluation of feature selection and tree-based algorithms for traffic classification. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 1–6.

[8] Rushi Jayeshkumar Babaria, Minzhao Lyu, Gustavo Batista, and Vijay Sivaraman. 2025. FastFlow: Early Yet Robust Network Flow Classification using the Minimal Number of Time-Series Packets. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 9, 2 (2025), 1–27.

[9] Fred Baker, Bill Foster, and Chip Sharp. 2004. Cisco architecture for lawful intercept in IP networks. *Internet Engineering Task Force, RFC* 3924 (2004).

[10] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 267–280.

[11] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. 2006. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review* 36, 2 (2006), 23–26.

[12] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. 2006. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*. 1–12.

[13] Leo Breiman. 2001. Random forests. *Machine learning* 45 (2001), 5–32.

[14] Francesco Bronzino, Nick Feamster, Shinan Liu, James Saxon, and Paul Schmitt. 2021. Mapping the digital divide: before, during, and after COVID-19. In *TPRC48: The 48th Research Conference on Communication, Information and Internet Policy*.

[15] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Hyojoon Kim, Renata Teixeira, and Nick Feamster. 2021. Traffic refinery: Cost-aware data representation for machine learning on network traffic. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 3 (2021), 1–24.

[16] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 3, 3 (2019), 1–25.

[17] Jacob Alexander Markson Brown, Xi Jiang, Van Tran, Arjun Nitin Bhagoji, Nguyen Phong Hoang, Nick Feamster, Prateek Mittal, and Vinod Yegneswaran. 2023. Augmenting Rule-based DNS Censorship Detection at Scale with Machine Learning. *arXiv preprint arXiv:2302.02031* (2023).

[18] Zhiyong Bu, Bin Zhou, Pengyu Cheng, Kecheng Zhang, and Zhen-Hua Ling. 2020. Encrypted network traffic classification using deep and parallel network-in-network models. *IEEE Access* 8 (2020), 132950–132959.

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

[20] Christian Callegari, Sandrine Vaton, and Michele Pagano. 2008. A new statistical approach to network anomaly detection. In *2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*. IEEE, 441–447.

[21] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[22] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274. https://arxiv.org/abs/1512.01274

[23] Kimberly C Claffy. 1995. Internet traffic characterization. (1995).

[24] Benoit Claise. 2004. *Cisco systems netflow services export version 9.* Technical Report.

[25] Benoit Claise. 2008. *Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information.* Technical Report.

[26] Alessandro Cornacchia. 2020. *Optimized Flow Scheduling for Low Latency Data Center Networks.* Ph. D. Dissertation. Politecnico di Torino.

[27] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System.. In *NSDI*, Vol. 17. 613–627.

[28] Susu Cui, Bo Jiang, Zhenzhen Cai, Zhigang Lu, Song Liu, and Jian Liu. 2019. A session-packets-based encrypted traffic classification using capsule neural networks. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS).* IEEE, 429–436.

[29] Alberto Dainotti, Antonio Pescape, and Kimberly C Claffy. 2012. Issues and future directions in traffic classification. *IEEE network* 26, 1 (2012), 35–40.

[30] Kayathri Devi Devprasad, Sukumar Ramanujam, and Suresh Babu Rajendran. 2022. Context adaptive ensemble classification mechanism with multi-criteria decision making for network intrusion detection. *Concurrency and Computation: Practice and Experience* 34, 21 (2022), e7110.

[31] Christian Dewes, Arne Wichmann, and Anja Feldmann. 2003. An analysis of Internet chat systems. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement.* 51–64.

[32] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2016. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP).* sn, 407–414.

[33] Dewan Md Farid, Li Zhang, Alamgir Hossain, Chowdhury Mofizur Rahman, Rebecca Strachan, Graham Sexton, and Keshav Dahal. 2013. An adaptive ensemble classifier for mining concept drifting data streams. *Expert Systems with Applications* 40, 15 (2013), 5895–5906.

[34] Michael E Farmer, Shweta Bapna, and Anil K Jain. 2004. Large scale feature selection using modified random mutation hill climbing. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, Vol. 2. IEEE, 287–290.

[35] Center for Applied Internet Data Analysis (CAIDA). 2023. Passive Monitor Dataset at Equinix in Chicago. https://www.caida.org/catalog/datasets/monitors/passive-equinix-chicago/ Accessed: 2023-06-12.

[36] Massimo Gallo, Alessandro Finamore, Gwendal Simon, and Dario Rossi. 2021. FENXI: Deep-learning Traffic Analytics at the edge. In *2021 IEEE/ACM Symposium on Edge Computing (SEC).* IEEE, 202–213.

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

[38] Saptarsi Goswami, Sanjay Chakraborty, Priyanka Guha, Arunabha Tarafdar, and Aman Kedia. 2019. Filter-based feature selection methods using hill climbing approach. *Natural computing for unsupervised learning* (2019), 213–234.

[39] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2021. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security.* 3366–3383.

[40] Ren-Hung Hwang, Min-Chun Peng, Chien-Wei Huang, Po-Ching Lin, and Van-Linh Nguyen. 2020. An unsupervised deep learning model for early network traffic anomaly detection. *IEEE Access* 8 (2020), 30387–30399.

[41] Didier Frank Isingizwe, Meng Wang, Wenmao Liu, Dongsheng Wang, Tiejun Wu, and Jun Li. 2021. Analyzing learning-based encrypted malware traffic classification with automl. In *2021 IEEE 21st International Conference on Communication Technology (ICCT).* IEEE, 313–322.

[42] Kleidi Ismailaj, Miguel Camelo, and Steven Latré. 2021. When deep learning may not be the right tool for traffic classification. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM).* IEEE, 884–889.

[43] Syed Usman Jafri, Sanjay Rao, Vishal Shrivastav, and Mohit Tawarmalani. 2024. Leo: Online {ML-based} Traffic Classification at {Multi-Terabit} Line Rate. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24).* 1573–1591.

[44] Hongbo Jiang, Andrew W Moore, Zihui Ge, Shudong Jin, and Jia Wang. 2007. Lightweight application classification for network management. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management.* 299–304.

[45] Xi Jiang and Noah Apthorpe. 2021. Automating Internet of Things network traffic collection with robotic arm interactions. *arXiv preprint arXiv:2110.00060* (2021).

[46] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2024. Netdiffusion: Network data augmentation through protocol-constrained traffic generation. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–32.

[47] Xi Jiang, Saloua Naama Shinan Liu, Francesco Bronzino, Paul Schmitt, and Nick Feamster. [n. d.]. Towards Designing Robust and Efficient Classifiers for Encrypted Traffic in the Modern Internet. ([n. d.]).

[48] Dongzi Jin, Yiqin Lu, Jiancheng Qin, Zhe Cheng, and Zhongshu Mao. 2020. SwiftIDS: Real-time intrusion detection system based on LightGBM and parallel intrusion detection mechanism. *Computers & Security* 97 (2020), 101984.
[49] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. 2005. BLINC: multilevel traffic classification in the dark. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*. 229–240.
[50] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).
[51] Oguz Kaan Koksal, Recep Temelli, Huseyin Ozkan, and Ozgur Gurbuz. 2022. Markov Model Based Traffic Classification with Multiple Features. In *2022 International Balkan Conference on Communications and Networking (BalkanCom)*. IEEE, 173–177.
[52] Ferdi Kossmann, Ziniu Wu, Alex Turk, Nesime Tatbul, Lei Cao, and Samuel Madden. 2024. CascadeServe: Unlocking Model Cascades for Inference Serving. *arXiv preprint arXiv:2406.14424* (2024).
[53] Cing-Han Lai, Chao-Tung Yang, Endah Kristiani, Jung-Chun Liu, and Yu-Wei Chan. 2020. Using xgboost for cyberattack detection and analysis in a network log system with elk stack. In *Frontier Computing: Theory, Technologies and Applications (FC 2019) 8*. Springer, 302–311.
[54] Tanja Lang, Grenville Armitage, Phillip Branch, and Hwan-Yi Choo. 2003. A synthetic traffic model for Half-Life. In *Australian Telecommunications Networks & Applications Conference*, Vol. 2003.
[55] Tanja Lang, Philip Branch, and Grenville Armitage. 2004. A synthetic traffic model for Quake3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*. 233–238.
[56] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. 2020. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on machine learning*. PMLR, 5958–5968.
[57] Eric Liang, Hang Zhu, Xin Jin, and Ion Stoica. 2019. Neural packet classification. In *Proceedings of the ACM Special Interest Group on Data Communication*. 256–269.
[58] Hyun-Kyo Lim, Ju-Bong Kim, Joo-Seong Heo, Kwihoon Kim, Yong-Geun Hong, and Youn-Hee Han. 2019. Packet-based network traffic classification using deep learning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 046–051.
[59] Shinan Liu, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2021. Characterizing service provider response to the COVID-19 pandemic in the United States. In *International Conference on Passive and Active Network Measurement*. Springer, 20–38.
[60] Shinan Liu, Ted Shaowang, Gerry Wan, Jeewon Chae, Jonatas Marques, Sanjay Krishnan, and Nick Feamster. 2024. ServeFlow: A Fast-Slow Model Architecture for Network Traffic Analysis. *arXiv preprint arXiv:2402.03694* (2024).
[61] Zhen Liu, Nathalie Japkowicz, Ruoyu Wang, and Deyu Tang. 2019. Adaptive learning on mobile network traffic data. *Connection Science* 31, 2 (2019), 185–214.
[62] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 3 (2020), 1999–2012.
[63] Jun Lu, Shazia Sadiq, and Yan Qu. 2018. Data quality in machine learning: A review. *arXiv preprint arXiv:1812.02250* (2018).
[64] Qianli Ma, Wei Huang, Yanliang Jin, and Jianhua Mao. 2021. Encrypted traffic classification based on traffic reconstruction. In *2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, 572–576.
[65] Kyle MacMillan, Tarun Mangla, James Saxon, and Nick Feamster. 2021. Measuring the performance and network utilization of popular video conferencing applications. In *Proceedings of the 21st ACM Internet Measurement Conference*. 229–244.
[66] Andrew W Moore and Konstantina Papagiannaki. 2005. Toward the accurate identification of network applications. In *International workshop on passive and active network measurement*. Springer, 41–54.
[67] David Moore, Colleen Shannon, Douglas J Brown, Geoffrey M Voelker, and Stefan Savage. 2006. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems (TOCS)* 24, 2 (2006), 115–139.
[68] Vinod Nigade, Pablo Bauszat, Henri Bal, and Lin Wang. 2022. Jellyfish: Timely inference serving for dynamic edge networks. In *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 277–290.
[69] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library.
[70] Vern Paxson. 1994. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM transactions on Networking* 2, 4 (1994), 316–336.

[71] Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks* 31, 23-24 (1999), 2435–2463.

[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[73] Julien Piet, Dubem Nwoji, and Vern Paxson. 2023. GGFAST: Automating Generation of Flexible Network Traffic Classifiers. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 850–866.

[74] Philipp Probst, Marvin N Wright, and Anne-Laure Boulesteix. 2019. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: data mining and knowledge discovery* 9, 3 (2019), e1301.

[75] Kun Qiu, Harry Chang, Ying Wang, Xiahui Yu, Wenjun Zhu, Yingqi Liu, Jianwei Ma, Weigang Li, Xiaobo Liu, and Shuo Dai. 2022. Traffic Analytics Development Kits (TADK): Enable Real-Time AI Inference in Networking Apps. In *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 392–398.

[76] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2017. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376* (2017).

[77] Martin Roesch. 2005. Snort-the de facto standard for intrusion detection/prevention.

[78] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. 2004. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th international conference on World Wide Web*. 512–521.

[79] Tal Shapira and Yuval Shavitt. 2019. Flowpic: Encrypted internet traffic classification is as easy as image recognition. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 680–687.

[80] Yuval Shavitt and Eran Shir. 2005. DIMES: Let the Internet measure itself. *ACM SIGCOMM Computer Communication Review* 35, 5 (2005), 71–74.

[81] Pallavi Singhal, Rajeev Mathur, and Himani Vyas. 2013. State of the Art Review of Network Traffic Classification based on Machine Learning Approach. *International Journal of Computer Applications* 975 (2013), 8887.

[82] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. 2022. Re-architecting Traffic Analysis with Neural Network Interface Cards. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[83] David B Skalak. 1994. Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Machine Learning Proceedings 1994*. Elsevier, 293–301.

[84] Robin Sommer. 2003. Bro: An open source network intrusion detection system. *Security, E-learning, E-Services, 17. DFN-Arbeitstagung über Kommunikationsnetze* (2003).

[85] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*. 305–316. https://doi.org/10.1109/SP.2010.25

[86] Lawrence Stewart, Grenville Armitage, Philip Branch, and Sebastian Zander. 2005. An architecture for automated network control of QoS over consumer broadband links. In *TENCON 2005-2005 IEEE Region 10 Conference*. IEEE, 1–6.

[87] Boyu Sun, Wenyuan Yang, Mengqi Yan, Dehao Wu, Yuesheng Zhu, and Zhiqiang Bai. 2020. An encrypted traffic classification method combining graph convolutional network and autoencoder. In *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.

[88] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. 2022. Taurus: A Data Plane Architecture for Per-Packet ML. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[89] Tushar Swamy, Annus Zulfiqar, Luigi Nardi, Muhammad Shahbaz, and Kunle Olukotun. 2023. Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[90] Geza Szabo, Istvan Szabo, and Daniel Orincsay. 2007. Accurate Traffic Classification. In *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*. 1–8. https://doi.org/10.1109/WOWMOM.2007.4351725

[91] Da Tong, Yun R Qu, and Viktor K Prasanna. 2014. High-throughput traffic classification on multi-core processors. In *2014 IEEE 15th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 138–145.

[92] Van Tong, Hai Anh Tran, Sami Souihi, and Abdelhamid Mellouk. 2018. A novel quic traffic classifier based on convolutional neural networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[93] Martino Trevisan, Danilo Giordano, Idilio Drago, Marco Mellia, and Maurizio Munafo. 2018. Five Years at the Edge: Watching Internet from the ISP Network. In *CoNEXT 2018*. Heraklion, Greece.

[94] Gerry Wan, Shinan Liu, Francesco Bronzino, Nick Feamster, and Zakir Durumeric. 2024. CATO: End-to-End Optimization of ML-Based Traffic Analysis Pipelines. *arXiv preprint arXiv:2402.06099* (2024).

[95] Maonan Wang, Kangfeng Zheng, Dan Luo, Yanqing Yang, and Xiujuan Wang. 2020. An encrypted traffic classification framework based on convolutional neural networks and stacked autoencoders. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*. IEEE, 634–641.

[96] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai L Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Mining and Knowledge Discovery* 30, 4 (2016), 964–994.
[97] Carey Williamson. 2001. Internet traffic measurement. *IEEE internet computing* 5, 6 (2001), 70–74.
[98] Xuwei Xue, Shaojuan Zhang, Bingli Guo, Wei Ji, Rui Yin, Bin Chen, and Shanguo Huang. 2023. Optical Switching Data Center Networks: Understanding Techniques and Challenges. *arXiv preprint arXiv:2302.05298* (2023).
[99] Haipeng Yao, Chong Liu, Peiying Zhang, Sheng Wu, Chunxiao Jiang, and Shui Yu. 2019. Identification of encrypted traffic through attention mechanism based long short term memory. *IEEE Transactions on Big Data* (2019).
[100] Sung-Ho Yoon, Jin-Wan Park, Jun-Sang Park, Young-Seok Oh, and Myung-Sup Kim. 2009. Internet application traffic classification using fixed IP-port. In *Asia-Pacific Network Operations and Management Symposium*. Springer, 21–30.
[101] Changgang Zheng, Zhaoqi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2022. IIsy: Practical in-network classification. *arXiv preprint arXiv:2205.08243* (2022).
[102] Wenbo Zheng, Chao Gou, Lan Yan, and Shaocong Mo. 2020. Learning to classify: A flow-based relation network for encrypted traffic classification. In *Proceedings of The Web Conference 2020*. 13–22.
[103] Weiping Zheng, Jianhao Zhong, Qizhi Zhang, and Gansen Zhao. 2022. MTT: an model for encrypted network traffic classification using multi-task transformer. *Applied Intelligence* (2022), 1–16.

A Additional JITI Feature Analysis

Feature Sets	TTD (seconds)	Memory Requirement (MB per instance)	Performance (F1-Score)
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-ackf	0.303	315	0.744
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&tcp-ackf	0.318	323	0.772
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff	0.32	317	0.773
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&tcp-psh	0.326	321	0.774
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&tcp-ackf&tcp-psh	0.334	323	0.791
ipv4-dfbit&tcp-fin&ipv4-ttl	0.293	317	0.693
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-psh	0.307	321	0.716
ipv4-dfbit&tcp-fin&tcp-wsize	0.316	321	0.736
ipv4-dfbit&ipv4-ttl	0.282	331	0.655
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-wsize	0.357	317	0.826
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&ipv4-tos	0.352	323	0.797
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&tcp-ackf&ipv4-tos	0.354	317	0.792
ipv4-dfbit&tcp-fin&ipv4-ttl&ipv4-tl	0.362	319	0.8
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&udp-cksum	0.373	338	0.815
ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&tcp-ackf&udp-cksum	0.376	312	0.815

Table 5. Top 15 inference models in the pool generated for evaluation purpose, with the highest performance-to-efficiency ratios at a selected batch size of 500 flows.

A.1 JITI Comparison to Simple Lightweight Pipelines

In addition to our primary baseline comparisons, we provide supplementary analysis (Table 6) to further demonstrate the effectiveness of JITI's flexibility in feature selection and its adaptive scheduler in achieving high F1-Scores and low TTDs. Specifically, we created five additional lightweight PCIPs using the LightGBM algorithm. These pipelines use models that rely on a limited, fixed set of heuristic-based features selected based on the highest $\frac{FI_i}{Bits_i}$ ratios using the heuristic introduced in Section 3.1, but do not support flexible adjustments to feature requirements as JITI does. The results show that JITI consistently outperforms all five lightweight PCIPs, achieving significantly lower TTD and superior inference accuracy.

Classifier	Features	F1-Score	TTD (seconds)
JITI (LightGBM, $\mu = 0$)	Dynamically Chosen by Adaptive Scheduler	0.85	0.23
JITI (LightGBM, $\mu = 0.85$)		0.80	0.21
LightGBM 1	ipv4-dfbit	0.32	0.63
LightGBM 2	ipv4-dfbit&tcp-fin	0.42	0.88
LightGBM 3	ipv4-dfbit&tcp-fin&ipv4-ttl	0.7	01.26
LightGBM 4	ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff	0.77	1.33
LightGBM 5	ipv4-dfbit&tcp-fin&ipv4-ttl&tcp-doff&tcp-ackf	0.77	1.48

Table 6. JITI implementations with LightGBM surpasses all simple LightGBM classifiers with small fixed feature requirements in terms of TTD and classification performance..

Received June 2025; accepted September 2025