LoFi: Low-Cost Early Application Filter Based on Cached ML Decisions

Johann Hugon[1], Shinan Liu[2], Paul Schmitt[3], Nick Feamster[4], and Francesco Bronzino[1,5]
[1]*ENS Lyon,* [2]*University of Hong Kong,* [3]*Cal Poly,* [4]*University of Chicago,* [5]*Institut universitaire de France*

***Abstract*—Application-layer filters are crucial for network traffic analysis tasks such as telemetry, Quality of Experience (QoE) monitoring, and intrusion detection. Unlike general traffic classification, which assigns applications to flows without strict real-time requirements, filters must operate inline where balancing accuracy and performance is paramount. Traditional classification techniques based on pattern-matching (*e.g.*, port numbers, IP addresses, or TLS SNI) offer low latency but face significant accuracy challenges as traffic becomes increasingly encrypted. While Machine Learning (ML) models achieve high classification accuracy on encrypted traffic, their computational overhead limits their applicability as inline filters. This paper introduces LoFi, a hybrid approach that combines the efficiency of pattern-matching with the accuracy of ML-based classification. By selectively applying ML models only when necessary, LoFi reduces packet loss from 38.33% (ML-only baseline) to 1.17% on CAIDA traces while maintaining low computational overhead.**

I. Introduction

Traffic analysis pipelines extract diverse information from network flows to support critical operational functions including network telemetry [1], [2], [3], [4], Quality of Experience (QoE) estimation [5], [6], [7], and intrusion detection [8], [9], [10], [11]. However, as traffic grows, the complexity of these analysis tasks increases, making filtering out irrelevant traffic crucial for system efficiency and accuracy. In this context, a *filter* is a stateful mechanism that selectively admits or rejects network flows based on predetermined criteria related to their originating applications. This differs from general *traffic classification*, which assigns services or applications to flows as an analysis task without strict real-time requirements. A filter, by contrast, must operate inline with strict latency and throughput constraints, making the balance between classification accuracy and system performance paramount. Yet, modern traffic analysis often overlooks robust application-aware filtering mechanisms, an oversight that is particularly problematic as modern networks carry increasingly complex and encrypted traffic, making accurate filtering an even more fundamental component of traffic analysis.

Over time, filtering techniques have evolved from pattern-matching methods to learning-based [2], [3], [4]. Traditional static filters rely on header information such as port numbers or IP addresses, allowing for high-speed, low-overhead processing. However, these filters can be circumvented as modern services operate over HTTPS [12] and frequently change IP assignments through Content Delivery Networks (CDNs) [13]. Dynamic filtering approaches leverage DNS queries or TLS headers, such as inspecting the Server Name Indication (SNI) during TLS handshakes [14], [15], to identify applications. However, pervasive encryption [16], [17] and Encrypted Client Hello (ECH) [18] threaten to eliminate remaining plaintext indicators, making approaches reliant on them unsustainable.

In response, Machine Learning (ML) and Deep Learning (DL) models have emerged as powerful alternatives, demonstrating remarkable ability on encrypted traffic by analyzing statistical features like packet sizes and timings [10], [4], [5], or raw packet headers [19], [2]. However, these approaches introduce significant computational overhead when applied to every flow in real-time, high-speed networks [4], [20], [3], [2], [9]. Our findings show that ML-only approaches can lead to packet loss rates as high as 38.33% on representative network traces due to processing bottlenecks. Furthermore, many academic ML solutions are developed in offline environments [10], [21], [19], often lacking the optimizations and practical considerations needed for real-world deployment.

Existing methods have limitations: traditional techniques are inflexible and struggle with encryption, while ML is challenging to scale in practice. This motivates the need for a new strategy. Our work introduces LoFi, a hybrid application filter that balances the efficiency of traditional pattern-matching with the flexibility of ML-based classification. The architecture relies on an Early Identification module that uses cached IP addresses to process known flows, avoiding unnecessary ML execution. This approach efficiently handles most traffic and outperforms both direct ML inference and traditional TLS SNI inspection, while unrecognized flows are sent through the ML pipeline, which updates the Decision cache. To manage the cache effectively, the architecture incorporates Least Recently Used (LRU) eviction, time-based expiration, and thread-safe inter-core sharing. This architecture achieves substantial performance gains; for instance, on CAIDA traces, Early Identification reduced packet loss from 38.33% to 1.17%. We release LoFi as open-source software to facilitate further research and practical adoption.[1]

II. Related Work

This section reviews traffic classification techniques and evaluates their applicability as inline filters, where balancing accuracy and performance is paramount (Table I). We organize these techniques into four categories: static approaches using

[1]https://github.com/ENSL-NS/LoFi

Filter	Low latency	Dynamic	Payload encryption	ECH	DNS encryption
Pattern-matching	✓		✓	✓	✓
DPI		✓		✓	✓
DNS	✓	✓	✓	✓	
TLS SNI		✓	✓		✓
ML		✓	✓	✓	✓
LoFi	✓	✓	✓	✓	✓

TABLE I: Comparison of classification techniques and their suitability for inline filtering

fixed rules, dynamic approaches leveraging runtime information, ML-based approaches using learned models, and hybrid methods combining multiple techniques.

Static approaches. Static classification relies on fixed, predefined rules such as port numbers or IP addresses, offering low latency but limited accuracy. Traditional pattern-matching techniques (*e.g.*, port numbers, IPs) are computationally efficient but struggle with modern networks where most services use HTTPS and CDNs share IPs across applications [13]. Luxemburk *et al.* [22] show IP-based classification accuracy drops from 100% to 0% within weeks, confirming the need to constantly update IP lists. Encryption further undermines traditional methods: DPI was once robust [16] but is now ineffective due to pervasive encryption [17]. Trevisan *et al.* [23] showed that only 55% of web traffic can be identified using domains and IPs, with addresses changing frequently.

Dynamic approaches. Dynamic classification extracts information at runtime, such as DNS responses or TLS handshake fields, improving accuracy but introducing latency. To address CDN IP sharing, these techniques rely on runtime-extracted information. For example, Maghsoudlou *et al.* [24] identify services by associating DNS records with NetFlow, but this is nullified by DNS encryption [25], [26], [27], [28]. Shbair *et al.* [14], [15] investigated TLS SNI-based filtering and proposed combining it with DNS verification, but highlighted weaknesses even before the introduction of TLS's Encrypted Client Hello (ECH), reinforcing the need for alternatives.

ML-based approaches. ML-based classification uses learned models to identify flows based on statistical features or packet contents, achieving high accuracy but at significant computational cost. These filters effectively handle encrypted traffic and are resilient to ECH and DNS encryption [29], [30]. Early work explored clustering [31] and ML for HTTPS identification [32]. Recent approaches like FastFlow [33] and ServeFlow [2] perform early flow classification, but their computational demands limit their applicability as inline filters: ServeFlow requires 16 cores for under 10 Gbps, FastFlow uses 8 cores for 50K flows, and both suggest GPUs. In contrast, LoFi targets just a few cores. While feature engineering research exists [4], [3], [34], existing methods rely on multi-packet statistics, introducing latency, and require ML inference on all flows, which is costly at line rate.

Hybrid approaches. Hybrid methods combine multiple classification techniques to balance efficiency and accuracy, making them promising candidates for inline filtering. The most related work is SnortML [35], which uses ML for zero-day exploit detection. Unlike SnortML, LoFi targets service identification, supports all ONNX models (vs. TensorFlow-only), and minimizes ML invocations for efficiency. Since no existing classification technique achieves the right balance between accuracy and performance for inline filtering, we introduce LoFi in the next section.

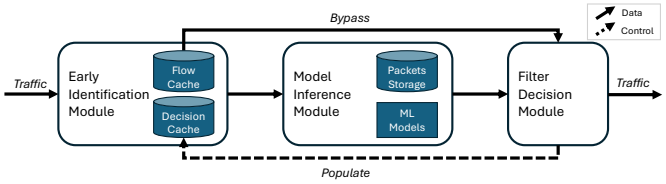


Fig. 1: Diagram of LoFi

III. LoFi

We propose LoFi, a filter design that aims to strike the right balance between the performance of pattern-matching rules and the flexibility of ML classification. The system operates in three main steps, illustrated in Figure 1. First, an early identification module takes advantage of previous decisions to classify incoming packets without invoking ML. If no match is found, the system falls back to ML classification, which handles uncertain or novel flows by analyzing statistical features. Packets are collected and grouped by flow to provide sufficient context for feature extraction. Once enough packets of the flow are gathered, features are extracted and sent to the ML model for classification. Finally, the Filter Decision Modules applies the result, updates the caches, and enables LoFi to efficiently capitalize on previous decisions to reduce unnecessary computational costs.

The rest of the section first presents how we support model integration through data subscriptions and model-agnostic design (Section III-A). We then describe how we take advantage of previous decisions to bypass ML models when possible (Section III-B). Finally, we discuss our prototype implementation and its ability to integrate other systems (Section III-C).

A. ML Inference

ML inference enables flexible classification of flows without the need for hand-written signatures or rules. However, ML models are constantly evolving, both in underlying techniques and in the data they consume. We design LoFi to support any model type and any feature type, ensuring long-term adaptability.

Model Support. Due to the rapid evolution of ML, systems tied to a single model architecture may become obsolete before publication. Moreover, many models are research prototypes that overlook practical deployment challenges, leading to high accuracy in controlled settings but poor performance in live environments. LoFi addresses this by facilitating deployment and evaluation of off-the-shelf ML models in real-world network environments. It serves as a flexible, model-agnostic module, allowing operators and researchers to deploy diverse

models easily. To ensure compatibility across models and frameworks, we enforce three key constraints:

1) **ONNX.** ONNX runtime [36] supports a wide variety of ML libraries. By adopting ONNX, LoFi becomes agnostic to the training framework, allowing developers to use their preferred tools (*e.g.*, TensorFlow [37], PyTorch [38], Scikit-learn [39]) while ensuring consistent deployment.
2) **Binary Decisions.** Each model must produce a binary decision interpreted as accepted or refused. This abstraction enables uniform handling of inference results across different architectures.
3) **Subscription.** Models must specify the type of flow data they require, referred to as a Subscription, as well as the number of packets needed for inference (see below).

By respecting these constraints, LoFi can seamlessly integrate a wide variety of ML models, supporting reproducible benchmarking and operational testing.

Traffic Subscriptions. A Subscription defines the specific type of input data that the system provides to an ML model. Each subscription represents a set of features derived from network flows, enabling the system to tailor data collection and processing according to the model's needs.

In our initial design, LoFi natively provides three Subscriptions:

- **Stats.** Statistical features inspired by Fauvel *et al.* [40], representing packet direction and size.
- **Raw bytes.** Byte-level representation where each packet is encoded as a fixed-length array of 1518 bytes, often used by DL models [41].
- **NPrint.** A generic packet representation designed for ML-based traffic analysis [19], where each header bit is encoded as 0, 1, or -1 (absent). We natively extract NPrint for IPv4/TCP/UDP.

However, LoFi is extensible: new Subscription types can be added by implementing the *Subscription* trait, which provides a uniform interface for feature extraction. Each implementation defines core functions for packet handling, feature processing, and metadata retrieval. Once the required packets are collected, the feature set is exported as a tensor and forwarded to the ML model for inference.

B. Early Identification Module

A cornerstone of LoFi's performance is the parsimonious use of ML, capitalizing on stored past decisions.

Caches. The system maintains two caches: a *flow cache* that tracks previously observed flows indexed by five-tuple (source/destination IPs, ports, and protocol), and a *Decision cache* that stores destination IP addresses associated with previous accept or reject decisions. When a packet arrives, the system first checks the flow cache; if the flow is known, it is immediately accepted or rejected bypassing the need to invoke the ML module. For new flows, the Decision cache is consulted; a match allows immediate acceptance without ML inference. Only when both caches miss does the system invoke ML classification. Upon ML decision, the flow cache is updated accordingly, and for accepted flows, the destination IP is added to the Decision cache. Manual cache updates are impractical due to dynamic IP assignments [13], so LoFi relies on ML to keep caches current, while time-based eviction removes stale entries.

Cache Management. To prevent memory exhaustion under high traffic, each cache is bounded by a maximum size (rather than a resizable HashMap) and uses LRU eviction to discard least-recently-accessed entries. This effectively removes inactive flows, including incomplete connections common in real-world traffic due to widespread TCP SYN scanning and incomplete handshakes [42], [43], [44]. The flow cache benefits from consistent hashing in high-performance networking stacks, which ensures packets of the same connection are processed by the same core. However, the Decision cache requires thread-safe sharing across cores: distinct flows to the same destination IP may be processed by different cores, and without sharing, one core might invoke ML while another already has the IP cached. We implement the Decision cache as a shared structure accessible system-wide, with time-based eviction to prevent stale entries from causing misclassification (see Section IV-C). Maintaining a globally accessible, thread-safe Decision cache ensures that knowledge about known IPs is leveraged system-wide, reducing redundant computation and improving overall throughput.

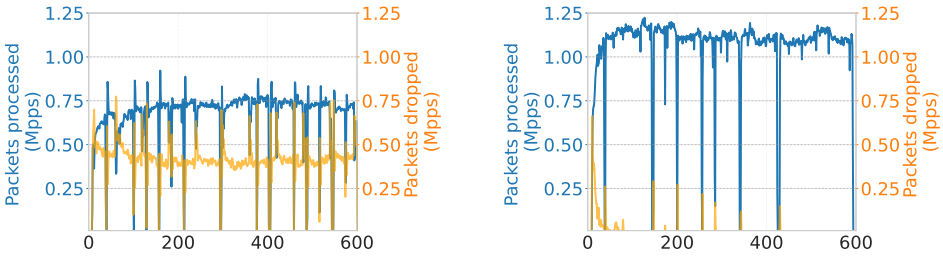
C. Prototype Implementation

We develop LoFi's prototype in Rust, leveraging its memory safety and concurrency guarantees. A key design goal was to build LoFi as a modular library that can be seamlessly integrated into larger systems requiring traffic filtering for downstream tasks such as telemetry collection, QoE monitoring, or security analysis. With LoFi, users can leverage the provided Subscription types or implement new ones to suit their needs. Cache management relies on the Moka [45] library, which partitions caches into independent segments, locking only the relevant segment during access and allowing concurrent multi-threaded operation. Finally, LoFi is agnostic to the packet delivery mechanism, allowing integration with various networking stacks. In our prototype, we use DPDK [46] for its low-latency, high-throughput performance. Finally, while multiclass classification is possible within our architecture, we leave its integration for future work implementing only binary decisions. We released the library as open source, enabling operators and researchers to deploy and evaluate diverse models.

IV. Evaluation

We evaluate LoFi on multiple scenarios using CAIDA traces [47], network traces from a tap in our campus network,[2] and purely synthetic traffic for micro benchmarks. We use T-Rex [48] to replay traffic from a server to a switch. The network is then duplicated via port mirroring from this switch.

[2]This tap was set up with campus networking and security teams to capture anonymized network traffic. The study focused solely on service performance and characteristics, not individual user behavior.



(a) Applying ML to each connection

(b) LoFi

Fig. 2: Number of processed and dropped packets (in Mpps) over 10 minutes of CAIDA traffic using 2 CPU cores

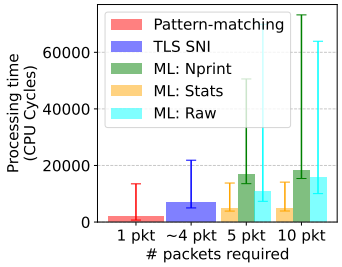


Fig. 3: Cost comparison between ML, Early Identification, and TLS-SNI (1st–99th percentiles)

This recreates a realistic environment in which an operator monitors a link from their network. Experiments are run on an AMD EPYC 7343 CPU with an Intel E810 NIC, with multithreading disabled and CPU cores isolated to prevent OS interference.

A. Overall performance

To evaluate the overall performance of the system under realistic conditions, we present an experiment, illustrated in figure 2, using the CAIDA trace from Equinix Chicago, captured on 2016-01-21 [47]. The trace is used in its original form, with the only modification being the merging of both traffic directions to ensure complete connections are preserved. We consider this trace to represent realistic traffic, as it originates directly from an ISP network.

We first run LoFi without the Early Identification module, running the ML pipeline on each connection. This approach replicates state-of-the-art service classification [10], [21], where models are typically evaluated offline. Such non-real-time analyses often process the entire traffic dataset without properly considering system constraints or deployment limitations.

We run the experiment for ten minutes, relying on two CPU cores, and using the NPrint Subscription for the first ten packets. Restricting the number of CPU cores is necessary because we expect to add more computation beyond just filtering traffic on the same server. As Figure 2a shows, the system cannot handle the load, resulting in significant packet loss. After ten minutes, 38.33% of input traffic was dropped. Next, we run the same traffic with the Early Identification module enabled. As Figure 2b shows, we observe a brief warm-up period during which the system achieves nearly zero packet loss with occasional spikes. Downward spikes in traffic are naturally present in the traces, likely caused by capture or anonymization artifacts. These irregularities allow us to observe how LoFi behaves with imperfect, real-world traffic. After ten minutes, 11% of connections were rejected, 31% were accepted, and 57% were still in the flow cache waiting for enough packets, resulting in only 1.17% packet loss. This experiment demonstrates the efficiency benefits of LoFi compared to state-of-the-art approaches under realistic workloads.

B. ML performance

We now break down the system to compare LoFi to other approaches.

Cost estimation. We first run microbenchmarks of both Early Identification and ML Subscriptions. For this experiment, we send dummy traffic generated with T-Rex and manually target particular paths within the system, as illustrated in Figure 3. To evaluate Early Identification (in red), we generate packets targeting pre-cached IP addresses, enabling focused microbenchmarking of this component. To evaluate the ML Subscriptions (in green, yellow, and cyan), we send the required number of packets with a random five-tuple and remove them from the cache after ML processing to avoid falling back on the Early Identification path. Finally, to establish a baseline and highlight its performance characteristics, we implemented TLS SNI processing in purple. This shows as four packets, because Client Hello is the fourth packet of the connection, but can be split into multiple TCP packets. To avoid skewing the results based on the performance of the models or the network, we did not consider the inference time of the machine learning or the time between packets, as these may vary according to use cases. This is motivated by the inherent design of LoFi and recent work showing the network latency is the higher and yet incompressible cost in packet processing [2]. Thus, the evaluation focuses primarily on the costs associated with Early Identification and Subscription types.

We measure processing time by reading the TSC register via the $RDTSC$ instruction, recording CPU cycles from a packet's entry to the filter's decision. This provides a low-overhead estimate of processing cost. Figure 3 shows the median processing times and the 1st–99th percentile range for each method. Early Identification is the fastest, with a median of 1,888 CPU cycles. Stats Subscriptions are next, with similar medians for 5 and 10 packets (4,704 and 4,864 CPU cycles) due to their computational simplicity, even lower than TLS SNI (6,816 CPU cycles). Raw Subscriptions are more costly, increasing from 10,816 to 15,648 CPU cycles. NPrint has the highest cost: 16,704 CPU cycles for 5 packets and 18,304 for 10. All methods are right-skewed, with more variability in complex cases, motivating the use of the median. These results highlight the significantly lower cost of Early Identification

Service	Target (%)	Daily Ext. (%)	Daily Extra. MC/UK	Hourly Ext. (%)	Hourly Extra. MC/UK
google	1.547	6.534	0.264/4.724	4.868	0.196/3.126
icloud	0.332	1.352	0.039/0.980	0.950	0.022/0.597
office	0.434	1.324	0.480/0.410	0.850	0.209/0.207
hypotheses	0.169	0.182	0.006/0.007	0.171	0.001/0.001
gmail	0.119	0.327	0.036/0.172	0.166	0.003/0.043
microsoft	0.989	2.354	0.438/0.927	1.927	0.241/0.697
spotify	0.236	0.389	0.071/0.082	0.320	0.040/0.045
hubiconnect	0.166	0.169	0.000/0.003	0.167	0.000/0.000
github	0.189	0.231	0.008/0.034	0.199	0.002/0.008
adobe	0.077	0.507	0.259/0.171	0.240	0.084/0.078
threads	0.013	0.056	0.003/0.039	0.014	0.000/0.000
youtube	0.072	2.961	1.030/1.859	1.465	0.472/0.922
facebook	0.073	0.652	0.055/0.523	0.549	0.049/0.427
apple	0.515	2.736	0.972/1.249	1.384	0.452/0.417
nflx	0.015	0.045	0.002/0.028	0.016	0.000/0.001

TABLE II: Percentage of targeted vs. extracted (Ext.) and extraneous (Extra.) flows, split into misclassified (MC) and unknown (UK).

compared to both the ML path and TLS SNI, emphasizing its critical role in improving overall system performance.

C. Filter accuracy

This section evaluates the accuracy of LoFi. As a filter designed to reduce overhead for downstream analysis tasks, LoFi favors inclusion over exclusion: when certainty is not possible, it is preferable to accept additional traffic rather than risk dropping relevant flows. The key question is how much extraneous traffic this strategy introduces. We collected traces from a network tap for eight days and extracted the available SNI and DNS records. We removed internal services, extracted the ten most used services, and added five popular services: Threads, YouTube, Facebook, Apple, and Netflix.

To emulate LoFi, we identify IP destinations from target-labeled flows, then extract all flows to those IPs to measure overhead. This simulates a perfect model that flags only relevant flows, resulting in a Decision cache containing all target IPs. This oracle mechanism aims to isolate the accuracy of the system from the accuracy of the model for evaluation purposes. The first column lists the service and its flows as a percentage of all flows over eight days. The Target column represents the ideal scenario where the filter extracts solely the flows labeled as the targeted service.

The left part of Table II replicates our system behavior with a daily eviction policy. As expected, the system extracts all required flows but also an extraneous quantity. The overhead (Extraneous) is the difference between Target and Extracted flows. Extraneous flows are split into two categories: Misclassified (MC, in red), identified by SNI or DNS as belonging to a different service, and Unknown (UK, in blue), where no ground truth is available. This last category can originate either from the target service or from others. Unlike classical filters that drop unrecognized traffic [49], [1], LoFi intentionally retains it, as the cost of processing additional flows downstream is preferable to missing relevant traffic.

The overhead varies depending on the service. For example, Google has 6.534% extracted flows versus 1.547% target, with 4.724% lacking ground truth and only 0.264% misclassified to other services, indicating relatively accurate targeting despite some ambiguity. Other services, like Hypotheses (a French research platform), show minimal overhead with only 0.006% misclassified and 0.007% unknown. These cases highlight how the scale and nature of a service influence IP filtering precision. With a maximum observed overhead of 6.534%, the results remain within acceptable limits given the inherent limitations of IP filtering in CDN-driven networks. Services like Apple show a more noticeable discrepancy, with a target volume of 0.515% and an extracted volume of 2.736%, suggesting significant IP address sharing. These results confirm that IP filtering effectively reduces overall traffic volume despite inducing moderate overhead.

The right part of Table II examines noise reduction with an hourly Decision cache eviction policy, resulting in a significant drop in all extracted volumes. In particular, services like HubiConnect and Netflix see extracted flows drop to near target levels. For others—such as Gmail, Adobe, YouTube, and Apple—the overhead is reduced by half. Although the reduction is less significant for some services, measurable gains are still observed. These results indicate that shortening the Decision cache expiration effectively reduces noise by frequently invalidating entries. Overall, despite accepting some extraneous traffic, LoFi substantially reduces the volume that downstream tasks must process while ensuring relevant flows are not missed.

V. Limitations and Future Works

While having promising result LoFi's evaluation has several limitations and leaves room for future research.

Limits of IP filter. Section IV-C shows that, although IP-based filters can achieve strong performance, their coarse granularity may introduce noise. Proposed approaches could be further enhanced by enriching cached IP entries with additional contextual information, enabling finer discrimination between services that share the same IP. This presents an opportunity for future investigation.

Modern traces. In this work, we use a combination of ISP trace, campus TAP data, and synthetic traffic, offering a broad picture for evaluation, enabling us to explore a new approach of a parsimonious usage of ML in networking. Nevertheless, exploring more recent traces and higher-speed settings, while falling outside the direct scope of this paper, can be a great opportunity for future works.

VI. Conclusion

We introduced LoFi, a hybrid application filter that balances the efficiency of pattern-matching with the flexibility of ML-based classification. By caching ML decisions in an Early Identification module, LoFi drastically reduces ML invocations, lowering packet loss from 38.33% to 1.17% on CAIDA traces. As a filter designed to reduce overhead for downstream analysis tasks, LoFi favors inclusion over exclusion, accepting moderate extraneous traffic rather than risking missed flows. Our evaluation shows that this trade-off remains within acceptable limits, with a maximum overhead of 6.534% under daily cache

eviction. We release LoFi as open source project to support reproducibility and practical adoption.

VII. Acknowledgement

This work has been supported by grants from the Agence Nationale de la Recherche (project no. ANR-21-CE94-0001 [MINT]), the National Science Foundation (grant nos. CNS-2334996 and CNS-2319603), and the France and Chicago Collaborating in the Sciences program.

References

[1] G. Wan, F. Gong *et al.*, "Retina: analyzing 100gbe traffic on commodity hardware," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022.

[2] S. Liu, T. Shaowang *et al.*, "Serveflow: A fast-slow model architecture for network traffic analysis," 2024. [Online]. Available: https://arxiv.org/abs/2402.03694

[3] X. Jiang, S. Liu *et al.*, "Ac-dc: Adaptive ensemble classification for network traffic identification," 2023. [Online]. Available: https://arxiv.org/abs/2302.11718

[4] G. Wan, S. Liu *et al.*, "Cato: End-to-end optimization of ml-based traffic analysis pipelines," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025.

[5] T. Sharma, T. Mangla *et al.*, "Estimating webrtc video qoe metrics without using application headers," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, 2023.

[6] T. Mangla, E. Halepovic *et al.*, "emimic: Estimating http-based video qoe metrics from encrypted network traffic," in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, 2018.

[7] J. Jiang, V. Sekar *et al.*, "Cfa: A practical prediction system for video qoe optimization," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016.

[8] A. Khraisat, I. Gondal *et al.*, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, 2019.

[9] Q. Zhang, A. Imran *et al.*, "Caravan: practical online learning of in-network ml models with labeling agents," in *Proceedings of the 3rd Workshop on Practical Adoption Challenges of ML for Systems*, 2024.

[10] S. Liu, T. Mangla *et al.*, "Amir: Active multimodal interaction recognition from video and network traffic in connected environments," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2023.

[11] R. Gupta, S. Liu *et al.*, "Generative active adaptation for drifting and imbalanced network intrusion detection," *arXiv preprint arXiv:2503.03022*, 2025.

[12] A. P. Felt, R. Barnes *et al.*, "Measuring https adoption on the web," in *26th USENIX security symposium (USENIX security 17)*, 2017.

[13] R. Guo, W. Li *et al.*, "Cdn judo: Breaking the cdn dos protection with itself." in *NDSS*, 2020.

[14] W. M. Shbair, T. Cholez *et al.*, "Efficiently bypassing sni-based https filtering," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[15] ——, "Improving sni-based https security monitoring," in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2016.

[16] T. Bujlow, V. Carela-Español *et al.*, "Independent comparison of popular dpi tools for traffic classification," *Computer Networks*, 2015.

[17] D. Naylor, A. Finamore *et al.*, "The cost of the "s" in https," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014.

[18] C. Huitema, "Issues and Requirements for Server Name Identification (SNI) Encryption in TLS," RFC 8744, Jul. 2020. [Online]. Available: https://www.rfc-editor.org/info/rfc8744

[19] J. Holland, P. Schmitt *et al.*, "New directions in automated traffic analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[20] S. U. Jafri, S. Rao *et al.*, "Leo: Online ml-based traffic classification at multi-terabit line rate," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024.

[21] J. Piet, D. Nwoji *et al.*, "Ggfast: Automating generation of flexible network traffic classifiers," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023.

[22] J. Luxemburk, K. Hynek *et al.*, "Encrypted traffic classification: the quic case," in *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*, 2023.

[23] M. Trevisan, I. Drago *et al.*, "Towards web service classification using addresses and dns," in *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2016.

[24] A. Maghsoudlou, O. Gasser *et al.*, "Flowdns: correlating netflow and dns streams at scale," in *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies*, 2022.

[25] P. E. Hoffman and P. McManus, "DNS Queries over HTTPS (DoH)," RFC 8484, Oct. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8484

[26] Z. Hu, L. Zhu *et al.*, "Specification for DNS over Transport Layer Security (TLS)," RFC 7858, May 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7858

[27] S. Dickinson, D. K. Gillmor *et al.*, "Usage Profiles for DNS over TLS and DNS over DTLS," RFC 8310, Mar. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8310

[28] C. Huitema, S. Dickinson *et al.*, "DNS over Dedicated QUIC Connections," RFC 9250, May 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9250

[29] R. Boutaba, M. A. Salahuddin *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, 2018.

[30] E. Papadogiannaki and S. Ioannidis, "A survey on encrypted network traffic analysis applications, techniques, and countermeasures," *ACM Computing Surveys*, vol. 54, no. 6, 2021.

[31] L. Bernaille, R. Teixeira *et al.*, "Early application identification," in *International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2006.

[32] W. M. Shbair, T. Cholez *et al.*, "Early identification of services in https traffic," 2020. [Online]. Available: https://arxiv.org/abs/2008.08350

[33] R. J. Babaria, M. Lyu *et al.*, "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*, vol. 9, no. 2, pp. 1–27, 2025.

[34] F. Bronzino, P. Schmitt *et al.*, "Traffic refinery: Cost-aware data representation for machine learning on network traffic," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2021.

[35] "Snort - Network Intrusion Detection Prevention System — snort.org," https://snort.org/, 2024, [Accessed 01-06-2025].

[36] O. R. developers, "Onnx runtime," https://onnxruntime.ai/, 2021, version: 1.22.0.

[37] T. Developers, "Tensorflow," *Zenodo*, 2022.

[38] S. Imambi, K. B. Prakash *et al.*, "Pytorch," *Programming with TensorFlow: solution for edge computing applications*, 2021.

[39] O. Kramer and O. Kramer, "Scikit-learn," *Machine learning for evolution strategies*, 2016.

[40] K. Fauvel, F. Chen *et al.*, "A lightweight, efficient and explainable-by-design convolutional neural network for internet traffic classification," 2023. [Online]. Available: https://arxiv.org/abs/2202.05535

[41] T. Shapira and Y. Shavitt, "Flowpic: A generic representation for encrypted traffic classification and applications identification," in *IEEE Transactions on Network and Service Management*, 2021.

[42] Z. Durumeric, E. Wustrow *et al.*, "Zmap: Fast internet-wide scanning and its security applications," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

[43] Z. Durumeric, D. Adrian *et al.*, "Ten years of zmap," in *Proceedings of the 2024 ACM on Internet Measurement Conference*, 2024.

[44] P. Jurkiewicz, G. Rzym *et al.*, "Flow length and size distributions in campus internet traffic," *Computer Communications*, 2021.

[45] "Moka, a fast and concurrent cache library inspired by java caffeine," https://crates.io/crates/moka, 2021.

[46] "Data plane development kit." https://www.dpdk.org/, 2023.

[47] CAIDA, "CAIDA anonymized internet traces 2016 dataset," 2016, access restricted to approved researchers. [Online]. Available: https://www.caida.org/data/passive/passive_2016_dataset.xml

[48] "Trex, realistic traffic generator," https://trex-tgn.cisco.com, 2023.

[49] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture," in *USENIX Winter Conference*, 1993.