

# Synecdoche: Efficient and Accurate In-Network Traffic Classification via Direct Packet Sequential Pattern Matching

Minyuan Xiao, Yunchun Li, Yuchen Zhao, Tong Guan, Mingyuan Xia, Wei Li\*

Beihang University

{xiaominyuan, lych, ruin1732, guantong, catfish2339210, liw}@buaa.edu.cn

**Abstract**—Traffic classification on programmable data plane holds great promise for line-rate processing, with methods evolving from per-packet to flow-level analysis for higher accuracy. However, a trade-off between accuracy and efficiency persists. Statistical feature-based methods align with hardware constraints but often exhibit limited accuracy, while online deep learning methods using packet sequential features achieve superior accuracy but require substantial computational resources. This paper presents Synecdoche, the first traffic classification framework that successfully deploys packet sequential features on a programmable data plane via pattern matching, achieving both high accuracy and efficiency. Our key insight is that discriminative information concentrates in short sub-sequences—termed Key Segments—that serve as compact traffic features for efficient data plane matching. Synecdoche employs an “offline discovery, online matching” paradigm: deep learning models automatically discover Key Segment patterns offline, which are then compiled into optimized table entries for direct data plane matching. Extensive experiments demonstrate Synecdoche’s superior accuracy, improving F1-scores by up to 26.4% against statistical methods and 18.3% against online deep learning methods, while reducing latency by 13.0% and achieving 79.2% reduction in SRAM usage. The source code of Synecdoche<sup>1</sup> is publicly available to facilitate reproducibility and further research.

**Index Terms**—traffic classification, programmable data plane, pattern matching.

## I. INTRODUCTION

The convergence of 5G, IoT, and edge computing is driving an evolution towards automated, intelligent networks. These networks demand real-time traffic analysis and classification capabilities for optimal performance [1], [2]. Such capabilities are critical for cybersecurity, enabling the rapid identification and mitigation of threats, as well as for guaranteeing the stringent Quality of Service (QoS) required by latency-sensitive applications. Consequently, the ability to perform accurate, real-time traffic classification directly within the data stream has become a cornerstone for the security, reliability, and performance of these next-generation networks.

To address these real-time requirements, performing traffic classification directly on the data plane offers significant advantages by avoiding the latency overhead of control plane round-trips and centralized processing bottlenecks.

Programmable Data Planes (PDPs) [3], [4], powered by languages like P4 [5], have emerged as a promising solution. They enable in-network inference by embedding classification models directly into the switching fabric, thus achieving high-throughput, line-rate traffic classification.

Traffic classification methods have evolved from per-packet approaches [9]–[11] to flow-based methods, as flow-level analysis provides richer contextual information and higher classification accuracy [7], [8]. Within flow-based approaches, existing methods can be categorized into two main paradigms based on the types of features they employ: statistical feature-based methods and packet sequential feature-based methods. Statistical feature-based approaches extract aggregated flow statistics and typically employ decision tree-based models [7], [8], [12], which are compatible with programmable switches. However, these methods suffer from inherent accuracy limitations due to the limited expressiveness of statistical summaries [30]–[34]. In contrast, sequence feature-based approaches leverage raw packet content and ordering information, typically employing deep learning models to learn discriminative patterns [13]–[15]. However, deploying these models on the data plane presents significant challenges. Hardware constraints requiring model quantization can degrade performance [15], while online inference consumes more hardware resources and introduces extra latency [6].

This raises a compelling research question: Can we harness the high accuracy of sequential features while maintaining efficient classification on the data plane? Our key insight is that we can leverage offline analysis to discover discriminative sequential patterns. Traffic flows inherently contain critical sub-sequences—Key Segments [18], [19]—that encapsulate the essence of packet sequential features. These segments, typically spanning just a few consecutive packets, capture the distinctive communication patterns that differentiate various traffic classes. We can transform them into range-based match-action table entries, effectively distilling the discriminative power of entire packet sequences into compact, hardware-friendly matching rules.

To realize this insight, we propose Synecdoche, a novel traffic classification framework that bridges the accuracy-efficiency gap through an “offline discovery, online matching” paradigm. Synecdoche consists of two phases: First,

\*Wei Li is the Corresponding author.

<sup>1</sup><https://github.com/swampx/Synecdoche>

**the discovery of Key Segments** (Section IV), where we employ deep learning models and Grad-CAM techniques to automatically extract discriminative packet sub-sequences from training data. Second, **matching with Key Segments** (Section V), where we transform discovered segments into optimized table entries and design a matching pipeline for line-rate classification. To the best of our knowledge, this represents the first work to successfully deploy sequential feature-based classification through direct pattern matching on a programmable data plane.

Our main contributions are:

- **Automated Key Segment Discovery Technique:** We propose a pipeline using deep learning and Grad-CAM that automatically discovers highly discriminative Key Segments from traffic data.
- **Efficient Data Plane Pattern Matching Framework:** We design the first efficient data plane architecture through direct pattern matching of packet sequential features, successfully achieving both high efficiency and high accuracy on a programmable data plane.
- **Comprehensive System Implementation and Experimental Validation:** We implement Synecdoche on a Tofino switch and conduct extensive experiments on public traffic datasets, achieving F1-score improvements of up to 26.4% over statistical methods and 18.3% over online deep learning approaches, while reducing latency by 13.0% and achieving 79.2% reduction in SRAM usage compared to existing approaches.

## II. BACKGROUND AND RELATED WORKS

### A. Programmable Data Plane

The evolution towards more dynamic and intelligent networks has been significantly propelled by the advent of the programmable data plane. At the heart of this evolution is the Protocol-Independent Switch Architecture (PISA), which deviates from the rigid, fixed-function pipelines of traditional network switches. PISA provides the flexibility to define custom packet processing logic directly in hardware, enabling rapid deployment of new network services without costly hardware replacement cycles.

The PISA pipeline consists of three primary programmable blocks: 1) A Programmable Parser that identifies and extracts header fields from incoming packets according to user-defined protocol specifications; 2) A Match-Action Pipeline comprising multiple sequential stages, each equipped with memory (SRAM, TCAM) and ALUs, where packets are matched against flow tables and subjected to corresponding actions; 3) A Programmable Deparser that reconstructs packets from processed headers and payload before transmission. This architectural flexibility is programmable via domain-specific languages like P4, offering immense opportunities. However, it also imposes strict constraints to maintain line-rate processing.

Any practical data plane solution must operate within these limitations:

**Memory Limitation:** High-speed memory in each stage is scarce. TCAM for complex matches is particularly limited,

while SRAM for exact-match tables is more abundant but still orders of magnitude smaller than conventional server memory.

**Operation Restrictions:** To ensure microsecond-level latency, ALUs support only minimalistic operations. Complex instructions such as multiplication, division, and floating-point arithmetic are unsupported, requiring decomposition into simple integer and bitwise operations.

These constraints collectively make direct deployment of conventional machine learning models a formidable challenge, necessitating novel approaches specifically designed for this unique computational environment.

### B. Related Work

Existing traffic classification approaches on programmable data plane can be categorized into two primary paradigms based on their processing methodology: **per-packet methods** and **flow-level methods**. Per-packet methods perform classification directly on individual packet features without storing stateful information. For instance, Leo [9] proposed a sub-tree multiplexing mechanism to implement scalable and runtime-programmable online decision tree models. Mousika [10] utilized knowledge distillation to convert complex ML models into hardware-friendly binary decision trees. However, due to the limited information available in individual packets, per-packet methods often exhibit limited accuracy [7], [8]. To address this limitation, flow-level methods have emerged, which can be further divided into two subcategories.

**Statistical Feature-Based Methods:** These approaches on programmable data plane extract aggregated flow statistics and employ decision tree-based models, as their hierarchical rule structure can be directly translated into the match-action tables native to programmable switches. Flowrest [8] fully implemented a random forest model on switches to support more complex flow-level inference. NetBeacon [7] proposed a multi-phase sequential model architecture that dynamically analyzes packets with line-speed flow-level features while addressing deployment scalability through efficient model representation and stateful storage management. SentinelX [12] introduced TreeDivider and DualTree algorithms that achieve significant space reduction and improved detection accuracy through optimized flow table representation and dual-threshold decision trees.

**Packet Sequential Feature-Based Methods:** These approaches analyze network traffic by extracting features from raw packet content and using temporal ordering information to capture sequential patterns and dependencies between packets. Such methods typically employ deep learning models to process feature sequences [25]–[29]. Existing work on programmable switches also deploys deep learning models to the data plane for sequential pattern learning. Brain-on-Switch [15] designed a data-plane-friendly RNN architecture enabling line-rate neural network-driven traffic analysis. RIDS [13] explored the co-design of RNN and programmable switches to build advanced intrusion detection systems. Quark [14] leveraged model pruning and quantization to fully offload

CNN inference to the data plane. Linc [39] design a divide-and-conquer strategy for incremental model updates on data plane.

### C. Discussion

Current traffic classification approaches face an accuracy-efficiency trade-off on the programmable data plane. Statistical feature-based methods suffer from two fundamental limitations: first, their performance is heavily reliant on hand-crafted features, which restricts their generalizability [30], [31]; second, the aggregation of raw data into statistical features causes an inevitable loss of fine-grained information, which ultimately limits their classification accuracy [32]–[34]. In contrast, packet sequential feature-based approaches face significant deployment challenges on the programmable data plane. Hardware constraints necessitate model simplification strategies such as pruning and quantization, which degrade model classification accuracy [6]. Furthermore, deploying these models directly on the data plane consumes substantial memory and computational resources, introducing extra processing latency [15]. Therefore, efficiently leveraging packet sequential features on the data plane remains a challenge.

## III. KEY SEGMENTS: DEFINITION, PROPERTIES, AND FRAMEWORK OVERVIEW

This section defines our core concept of "Key Segments" and presents an overview of our framework for their discovery and deployment on a programmable data plane.

### A. Key Segments: Definition and Core Properties

In traffic flows, specific packet patterns emerge due to the inherent structure of network protocols and application behaviors. These patterns, which we term **Key Segments**, originate from fundamental network operations such as protocol handshakes, request-response cycles, and application-specific data exchanges. As illustrated in Figure 1, different applications exhibit distinct communication patterns. For example, HTTPS may show characteristic GET request-response sequences, while DoH can demonstrate compact POST-based DNS query patterns with predictable response structures.

Formally, given a traffic sequence  $X = (x_1, \dots, x_n)$ , where each  $x_i$  represents packet features, a Key Segment is a contiguous subsequence that makes significant contributions to identifying  $X$  as belonging to a specific class. These segments capture the quintessential interaction patterns that distinguish different applications in traffic. The compactness of Key Segments makes them particularly suitable for hardware-based pattern matching implementations.

Key Segments are characterized by three core properties that make them both valuable for classification and challenging to discover:

- 1) **Variable Length:** Segments range from short 2-packet sequences to complex multi-packet sequences, reflecting diverse application logic.

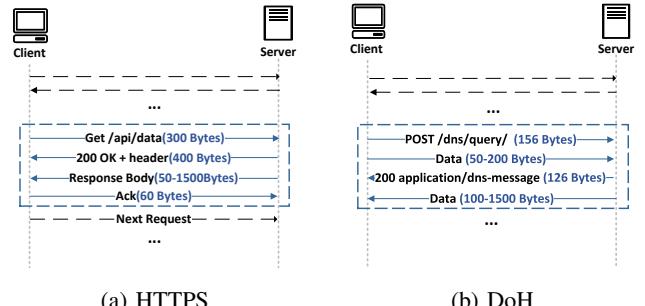


Fig. 1: Examples of Key Segments

- 2) **Positional Flexibility:** Key Segments appear at varying offsets within flows due to packet retransmissions and user behavior.
- 3) **Value Variability:** Packet sizes within segments fluctuate based on factors like content length, though some packets (e.g., ACKs) maintain stable sizes.

While these properties provide Key Segments with the expressive capability needed to effectively distinguish between different categories of traffic, they simultaneously create significant computational challenges for segment identification. The combinatorial search space across variable lengths, positions, and values leads to exponential complexity. Even with various restrictions such as limiting maximum segment lengths, the time complexity remains prohibitively high [18].

### B. Framework Overview

Figure 2 illustrates the overall architecture of our Synecdoche framework.

In the first phase, *The discovery of Key Segments*, we tackle the challenge of automatically and efficiently identifying Key Segments from raw traffic. For this, we leverage a 1D-CNN model and Grad-CAM [20] techniques to pinpoint the most influential traffic segments for each class. The segments are then refined through clustering and filtering to generate a representative set of Key Segments.

In the second phase, *matching with Key Segments on the data plane*, our framework transforms the discovered Key Segments into multi-key table entries and deploys them onto a programmable switch as a Key Segments Table. During online operation, the switch continuously updates packet feature registers for each flow and performs matching against the Key Segments Table. Once a successful match is detected, the flow is immediately classified, enabling dynamic and low-latency decision-making. Additionally, we provide a backup mechanism using a trained decision tree model to handle flows that fail to match any Key Segment within a predefined time window.

## IV. THE DISCOVERY OF KEY SEGMENTS

In this section, we elaborate on how to discover Key Segments from raw traffic.

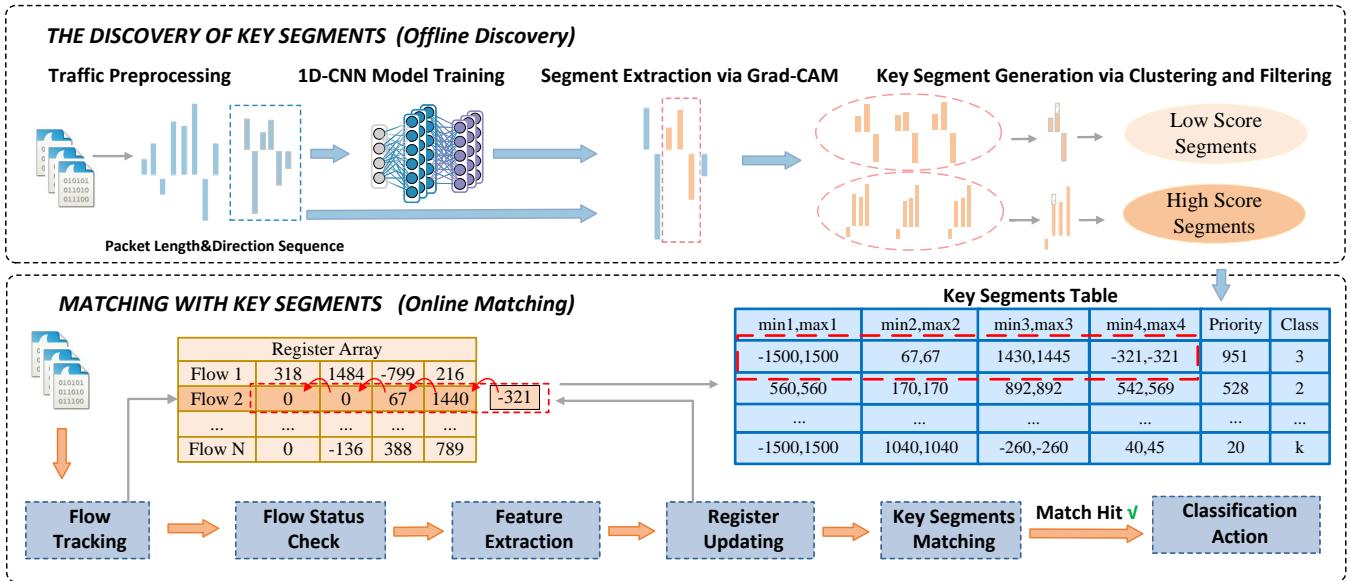


Fig. 2: The high-level architecture of our Synecdoche framework, illustrating the two-phase pipeline from offline discovery to online matching on the programmable data plane.

### A. Traffic Preprocessing

The process begins with preprocessing raw network data. We partition traffic into bidirectional flows based on the five-tuple (source/destination IP, source/destination port, protocol), then we extract the sequence of packet sizes with directional information. This feature representation has been proven effective for traffic analysis [18], [19], [25], [26]. Specifically, for each bidirectional flow, we define  $l_i$  as the length of the  $i$ -th packet, and  $d_i$  as the direction using  $\{+1, -1\}$  to represent the two directions. The combined feature is computed as:

$$x_i = l_i \times d_i \quad (1)$$

Each flow is then represented as  $\mathbf{X} = \{x_1, \dots, x_k\}$ , where  $k$  is the number of packets in the sample flow.

### B. 1D-CNN Model Training

To learn the discriminative patterns within these sequences, we designed a 1D-CNN model for traffic classification. Its architecture consists of three parts: an embedding layer, a 1D-CNN backbone, and a classifier. The embedding layer treats each packet length as a discrete categorical variable rather than a continuous numerical value, converting the signed integers into dense vectors through embedding layers. This separation of classification meaning from numerical value makes the neural network easier to train and captures richer semantic relationships between different packet sizes. The core of our model is its 1D-CNN backbone, which excels at identifying local, position-variant patterns, a perfect analogue for our Key Segments. The final classifier uses softmax loss for the multi-class classification task. We train this 1D-CNN model following standard procedures for multi-class classification tasks.

### C. Segment Extraction via Grad-CAM

With an accurately trained 1D-CNN model, we treat it as a learned knowledge base and apply model interpretability algorithms to identify which parts of an input sequence were most influential in its classification. We employ Grad-CAM [20], a gradient-based interpretability method that computes the gradient of the class score with respect to feature maps of the target convolutional layer, then uses global average pooling to obtain importance weights for generating class-discriminative localization maps.

The extraction process begins by applying Grad-CAM to compute an interpretability map for samples from each traffic category, assigning a continuous importance score to each packet position in the original input sequence. As illustrated in Figure 3, given a packet feature sequence, Grad-CAM generates corresponding importance scores that highlight the most discriminative patterns. For the aggregated importance scores  $S = \{s_1, s_2, \dots, s_n\}$ , we identify contiguous subsequences  $\{x_i, x_{i+1}, \dots, x_j\}$  from the original feature sequence where all corresponding importance scores satisfy:

$$s_k > \mu + t \cdot \sigma, \quad \forall k \in [i, j] \quad (2)$$

where  $\mu$  is the mean of all importance scores,  $\sigma$  is the standard deviation, and  $t$  is a threshold parameter.

We apply this extraction procedure to every sample in the training set to build a comprehensive pool of candidate segments. For each training sample, after identifying all contiguous subsequences that meet the threshold criterion, we apply length-based processing: subsequences shorter than the pre-defined minimum length  $L_{min}$  are discarded as insufficiently discriminative. For subsequences within the acceptable range  $[L_{min}, L_{max}]$ , we retain them entirely as candidate segments.

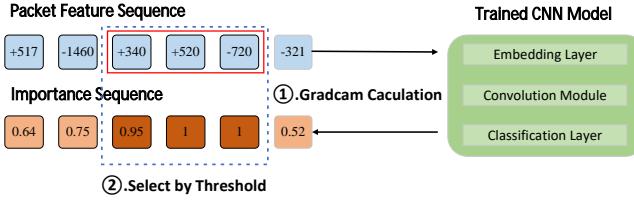


Fig. 3: An example of segment Extraction via Grad-CAM

When a subsequence exceeds the maximum length  $L_{max}$ , we use a sliding window to extract the sub-subsequence of length  $L_{max}$  that achieves the highest cumulative importance score within it.

#### D. Key Segment Generation via Clustering and Filtering

The raw candidate segments, while informative, can be numerous and contain statistical noise. To generate representative Key Segments from the raw candidate segments, we employ a two-stage generation process.

First, candidate segments are grouped by their originating sample class. Within each class-based group, we pad all segments to the maximum length  $L_{max}$  to capture segments of variable lengths as mentioned in our Key Segment properties. We then apply the DBSCAN clustering algorithm to identify dense regions of similar segments within each class. For each resulting cluster, we generate a single, generalized Key Segment as a **range-based template** directly suitable for data plane matching. For each position  $i$  within the segments of a cluster, we compute the minimum and maximum observed packet feature values:

$$\text{Key Segment} = [(x_1^{min}, x_1^{max}), \dots, (x_{L_{max}}^{min}, x_{L_{max}}^{max})] \quad (3)$$

where  $x_i^{min}$  and  $x_i^{max}$  represent the minimum and maximum feature values at position  $i$  across all segments in the cluster.

Second, since some of the segments might match multiple classes and lack sufficient discriminative power, we introduce a scoring mechanism to select only the most representative segments for each class. For each generated segment, we evaluate its discriminative power using a held-out validation dataset containing samples from all classes. For each segment  $s$  and its target class, we calculate  $C_{in}(s)$  as the fraction of in-class samples that match segment  $s$ . We also compute  $C_{out}(c, s)$  as the fraction of samples in each other class  $c$  that match segment  $s$ , and define:

$$C_{out}^*(s) = \max_{c \neq \text{target class}} C_{out}(c, s) \quad (4)$$

The discriminative score for segment  $s$  is then computed as:

$$\text{Score}(s) = \frac{C_{in}(s)}{C_{out}^*(s) + \epsilon} \quad (5)$$

where  $\epsilon$  is a small constant to prevent division by zero. This ratio-based scoring mechanism favors segments that achieve

high coverage within their target class while maintaining low coverage across other classes. Only segments with scores exceeding a predefined threshold  $S$  are retained for deployment. This ensures that only high-fidelity Key Segments with strong discriminative power are passed to the deployment stage.

## V. MATCHING WITH KEY SEGMENTS

In this section, we explain how to deploy discovered Key Segments on the data plane and how to use them in online classification.

### A. Key Segment Table Generation

The first step in deployment is to translate the Key Segments generated by our offline discovery process into match-action table entries consumable by the P4 data plane.

Each discovered Key Segment is compiled into a single P4 table entry, with each position within the segment becoming a key field that leverages the TCAM's range matching capabilities on P4-programmable switches. Specifically, for a Key Segment  $[(x_1^{min}, x_1^{max}), (x_2^{min}, x_2^{max}), \dots, (x_{L_{max}}^{min}, x_{L_{max}}^{max})]$ , we create  $L_{max}$  key fields. Each key field  $i$  is set to the range  $[x_i^{min}, x_i^{max}]$  for meaningful positions, while positions that were padded during clustering are set to match the universal set, effectively creating wildcard matches. The priority of each table entry is set to the discriminative Score computed during the filtering phase, ensuring that more discriminative segments are matched first. The action associated with each entry writes the corresponding class ID of the Key Segment into the packet's metadata.

Additionally, we provide an SRAM version that decomposes each range-containing segment into multiple exact-match entries using Cartesian product expansion, enabling precise matching on hardware with limited range matching support. In the SRAM version, since universal set matching is not feasible, we create separate tables for each possible segment length.

### B. Packet Processing Pipeline

The core of our system is the P4 packet processing pipeline that performs stateful matching at line rate. This pipeline design enables dynamic, early classification. A flow can be identified as soon as a matching Key Segment appears. The journey of each packet is as follows:

1) **Flow Identification:** Upon ingress, the packet is parsed to extract its 5-tuple. To handle bidirectional flows, we apply symmetric hashing to the 5-tuple, ensuring that packets from both directions of the same connection map to the same flow ID. This flow ID serves as an index for all subsequent stateful operations.

2) **Flow Status Check:** We query a flow classification table with its 5-tuple to determine whether this flow has already been classified. If the flow is found in the classification table with an assigned class label, the remaining steps are bypassed to conserve switch resources.

3) **Feature Extraction:** For each packet, we extract both length and direction information. The packet length is obtained

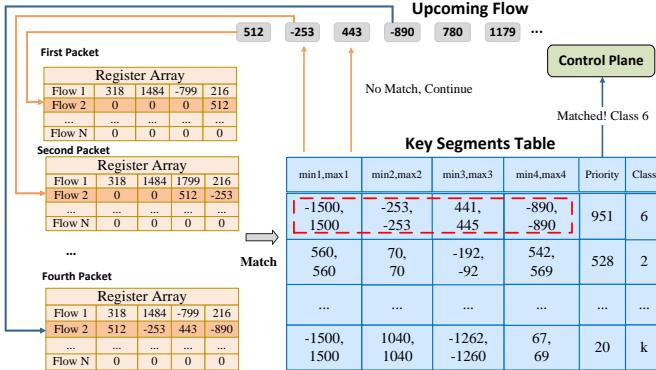


Fig. 4: An example of register update and Key Segment table matching

directly, while the direction is determined by comparing the current packet’s source IP with the first packet’s source IP in the flow. The combined feature of length and direction is computed using the same addition-based approach as described in the preprocessing phase.

**4) Stateful Register Update:** We maintain a large array of stateful registers on the switch, where each flow is mapped to its own dedicated register block of size  $L_{max}$  via its flow ID. This register block functions as a sliding window, holding the feature values of the last  $L_{max}$  packets observed for that flow. When a new packet arrives, the contents of the corresponding register block are shifted one position to the left, discarding the oldest packet feature, and the combined feature of the current packet is inserted into the newest position. As illustrated in Figure 4, flow 2’s register array starts as  $[0, 0, 0, 0]$ , evolves to  $[0, 0, 0, 512]$  after the first packet, then to  $[0, 0, 512, -253]$  after the second packet, and so on.

**5) Key Segment Matching:** After the register is updated, its entire  $L_{max}$ -element content is used as the lookup key for the main Key Segment Table. If no match occurs, the process repeats when the next packet in the flow arrives. As shown in Figure 4, for example, the first three packets of flow 2 result in register contents that do not match any table entries. However, when the fourth packet arrives and the register content becomes  $[512, -253, 443, -890]$ , it matches the first rule of the Key Segments Table, triggering classification and assigning the flow to Class 6.

**6) Classification Action:** If the lookup yields a match, the system uploads summary information to the control plane, including the flow’s 5-tuple and the identified class. The control plane then installs the corresponding flow classification entry into the data plane’s flow classification table.

#### C. Backup Classification Mechanism

To ensure that all flows are eventually classified and to handle traffic that may not contain any of the discovered Key Segments, we incorporate a robust backup mechanism. The backup mechanism employs a pre-trained Decision Tree (DT) implemented on the data plane. This DT is trained offline using the same packet features (the first  $L_{max}$  packets’ length and

direction) as Key Segments matching, requiring no additional storage space.

The decision tree classification is performed when the  $L_{max}$  packet arrives. The classification result is then stored and applied as the final flow label under two triggering conditions: (1) when the flow’s packet count exceeds a predefined threshold  $pkt_{max}$ , or (2) when the time interval between consecutive packets in a flow exceeds  $time_{max}$ . In both cases, if no Key Segment has been matched yet, the system retrieves and uses the stored decision tree classification result as the final flow label.

## VI. EVALUATION

In this section, we present a comprehensive evaluation of Synecdoche. Specifically, we aim to answer the following key research questions:

**(RQ1) Accuracy:** Does Synecdoche’s direct matching of Key Segments achieve classification accuracy comparable to sophisticated online deep learning methods while surpassing traditional statistical feature-based approaches?

**(RQ2) Efficiency:** Can Synecdoche deliver superior performance in terms of latency and hardware resource utilization compared to existing data plane inference methods?

#### A. Experimental Setup

**Environment.** Our experiments are conducted in a two-part environment reflecting our framework’s offline and online phases.

- Offline Training Environment:** The discovery of Key Segments, including the training of the CNN model and the subsequent refinement process, is performed on a server running Linux Ubuntu 22.04. The server is equipped with an Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz and an NVIDIA RTX 4090 GPU. The software stack consists of Python 3.8, TensorFlow 2.6 for the deep learning framework, and the nfstream [35] for traffic processing.
- Online Data Plane Environment:** The line-rate classification engine is deployed and evaluated on a Ruijie RG-F9500-32CQ switch, which is equipped with an Intel Tofino ASIC. The data plane logic is implemented in P4<sub>16</sub>.

**Datasets.** We evaluate our method on the following tasks:

- IoT Cybersecurity:** We use two datasets to evaluate attack detection capabilities on IoT devices: (1) **Bot-IoT** [21] dataset containing normal and attack traffic from IoT devices with 4 classes (DoS, DDoS, Scan, Theft), and (2) **ToN-IoT** [22] dataset providing heterogeneous IoT network attack scenarios with 10 classes covering normal traffic and various attacks.
- Application Classification:** We evaluate on two encrypted traffic datasets with varying class complexities: (1) **CipherSpectrum** [23] dataset includes network traffic encrypted with modern TLS 1.3 cipher suites for application domains, which we evaluate at four different scales: 5, 10, 20, and 42 (all) website classes to analyze

scalability. (2) **VisQUIC** [24] dataset containing QUIC-encrypted traffic from 16 different websites, which we evaluate at three scales: 5, 10, and 16 (all) website classes. CipherSpectrum-10 and VisQUIC-10 refer to the 10-class versions of each dataset.

For each dataset, we use a standard 80%/10%/10% split for training, validating, and testing, respectively.

**Comparison Methods.** To comprehensively evaluate our approach, we compare Synecdoche against both data plane methods and offline baselines:

- **Data Plane Methods:**

**NetBeacon** [7]: State-of-the-art statistical feature-based approach using decision trees on programmable switches, representing traditional ML-based P4 inference. **Brain-on-Switch** [15]: A representative framework for deploying quantized neural networks onto the data plane.

- **Offline Baselines(Upper Bound Analysis):**

**Random Forest**: Offline statistical method using the same features as NetBeacon but without hardware constraints (unlimited tree depth and estimators). **FS-Net** [31]: State-of-the-art offline sequence feature-based deep learning method, representing the upper bound of accuracy for sequence-based approaches.

To ensure fair comparison, we configure the input features and hyperparameters of comparing methods according to the settings specified in their respective official implementations [36]–[38]. Notably, regarding the IMIS system in BoS, while we recognize its contribution in improving classification performance through data-control plane interaction, our evaluation focuses on the classification capability of the data plane itself, without incorporating the performance gains from such interaction. This methodology ensures fairness, as Synecdoche can also be integrated into systems like IMIS.

**Evaluation Metrics.** We assess performance using two categories of metrics:

- **Accuracy Metrics:** We use standard classification metrics, including overall Accuracy (Acc.) and the macro-averaged F1-Score (F1), which provides a balanced measure of precision and recall.
- **Performance Metrics:** Per-packet Latency (ns), and critical on-chip resource consumption (SRAM and TCAM as percentage of total available resources on Tofino ASIC).

## B. Hyperparameter Settings and Method Analysis

In this subsection, we provide an in-depth analysis of our method’s key components and validate the effectiveness of our Key Segment discovery approach through systematic parameter analysis and early classification validation.

Our framework involves several key parameters that affect both Key Segment matching accuracy and data plane efficiency. Through systematic parameter tuning, we maintain consistent parameter settings across most datasets. Table I summarizes these key parameters. However, the score threshold  $S$  requires dataset-specific optimization, as it controls the critical trade-off between first-stage matching rate and overall accuracy.

TABLE I: Key Hyperparameter Settings for Synecdoche

Parameter	Optimal Value
Min/Max Segment Length ( $L_{\min}, L_{\max}$ )	2, 4
CNN Embedding Dimension	128
CNN Kernel Size / Feature Channel	3, 128
Extraction Threshold ( $t$ )	0.5
Backup Mechanism Threshold (pkt <sub>max</sub> , time <sub>max</sub> )	30, 256ms

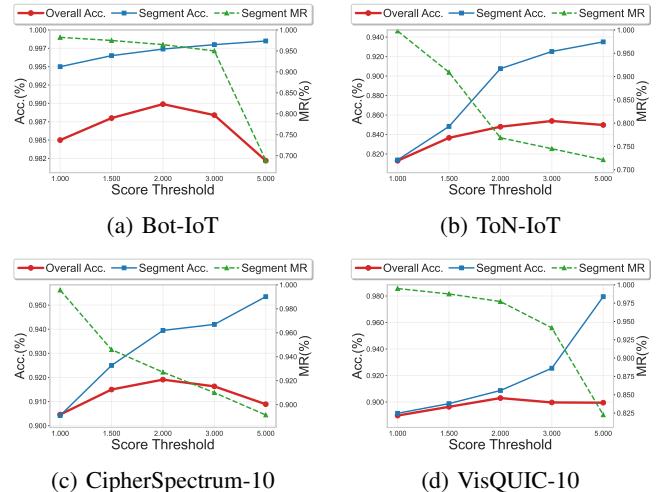


Fig. 5: Score threshold selection across datasets.

Figure 5 demonstrates the threshold selection trade-off across our evaluation datasets. As the score threshold increases, segment accuracy (Segment Acc.) consistently improves because only class-specific segments are retained, filtering out segments that might match samples from other categories. However, this improvement comes at the cost of reduced segment matching rate (Segment MR), meaning more samples must rely on the less accurate Backup Classification Mechanism for classification. Consequently, the overall accuracy (Overall Acc.) of Synecdoche reaches its optimal performance at specific threshold values: a score of 2.0 for three datasets (Bot-IoT, ToN-IoT, and CipherSpectrum-10) and 3.0 for VisQUIC-10. At these optimal thresholds, segment-based classification successfully handles 95%, 85%, 92%, and 90% of samples for Bot-IoT, ToN-IoT, CipherSpectrum-10, and VisQUIC-10, respectively.

Figure 6 demonstrates that Synecdoche’s Key Segment matching positions closely align with FS-Net’s accuracy convergence points. On ToN-IoT, 50% of samples are classified by packet 6, where FS-Net begins stabilizing, while 95% are classified by packet 12 near full convergence. This alignment confirms that Key Segments capture the same discriminative information used by deep learning models. Beyond validation, classification at these critical positions enables significant computational savings through early stopping while maintaining accuracy and reducing latency. Furthermore, different datasets show varying Key Segment positions: Bot-IoT (95% by packet 10), ToN-IoT (packet 12), CipherSpectrum-10 (packet 22), and VisQUIC-10 (packet 20). This variation

TABLE II: Classification Accuracy Comparison Across Different Traffic Types

Method	Bot-IoT		ToN-IoT		CipherSpectrum-10		VisQUIC-10	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
<i>Data Plane Methods</i>								
Synecdoche	<b>0.997</b>	<b>0.959</b>	<b>0.852</b>	<b>0.793</b>	<b>0.919</b>	<b>0.915</b>	<b>0.903</b>	<b>0.897</b>
NetBeacon	0.993	0.895	0.644	0.645	0.655	0.681	0.771	0.732
Brain-on-Switch	0.995	0.960	0.813	0.708	0.838	0.835	0.739	0.714
<i>Offline Baselines</i>								
Random Forest	0.989	0.919	0.837	0.824	0.822	0.826	0.775	0.737
FS-Net	0.997	0.941	0.892	0.863	0.997	0.997	0.956	0.956

reflects the inherent characteristics of each dataset—attack detection datasets require fewer packets due to distinct malicious signatures, while application classification needs more packets to distinguish similar behaviors.

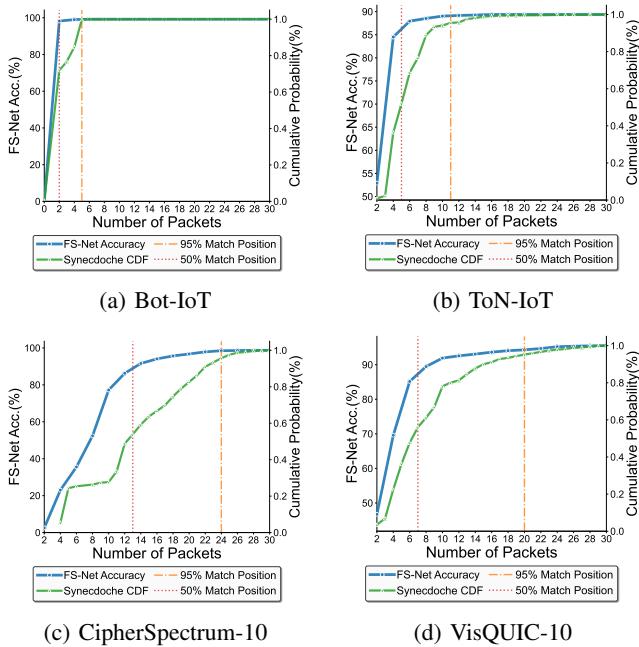


Fig. 6: Validation of Key Segment positioning through comparison with FS-Net accuracy convergence across different datasets.

### C. Accuracy Comparison (RQ1)

To answer RQ1, we evaluate the classification accuracy of Syncedcoche compared to existing data plane methods and offline baselines across different traffic classification scenarios and class complexities.

Table II demonstrates that our method significantly outperforms existing data plane approaches across most datasets. Compared to the statistical feature-based NetBeacon, we observe substantial F1-score improvements: 22.9% relative improvement on ToN-IoT (from 0.645 to 0.793) and 34.4% on CipherSpectrum-10 (from 0.681 to 0.915). Against the sequential feature-based Brain-on-Switch (BoS), our method demonstrates superiority on three datasets, with F1-score improvements of 8.5% on ToN-IoT, 8.0% on CipherSpectrum-

10, and 18.3% on VisQUIC-10, while achieving comparable performance on Bot-IoT.

The results also validate that packet sequential features methods possess higher accuracy ceilings than statistical approaches. The offline comparison shows FS-Net consistently outperforming Random Forest, while among online methods, Brain-on-Switch generally surpasses NetBeacon, confirming that packet sequential features provide superior discriminative power for traffic classification tasks.

Furthermore, our method demonstrates superior performance in application classification scenarios for two main reasons. First, application-layer traffic exhibits more distinctive segment characteristics due to standardized communication patterns and fixed interface calls, making segment matching particularly effective. Second, as shown in Figure 6, critical discriminative information for application classification typically resides in later packet positions within flows, while competing methods like NetBeacon (which makes decisions at packets 2, 4, 8, 16, etc.) and BoS (using multiple fixed sliding windows) often trigger premature classification decisions before sufficient contextual information becomes available. Our approach only initiates classification upon Key Segments detection, ensuring adequate information for accurate classification.

To evaluate how our method’s accuracy scales with increasing classification complexity, we conduct experiments across different numbers of classes for both application classification datasets. As demonstrated in 7, our Syncedcoche method consistently outperforms both baseline approaches across all scenarios. On CipherSpectrum, our method maintains accuracy of 97.3%, 91.9%, 83.8%, and 73.6% for 5, 10, 20, and 42 classes, respectively. In contrast, NetBeacon degrades dramatically from 80.6% to 28.7%, and BoS drops from 97.4% to 33.3%. Similarly, on VisQUIC, our approach achieves 99.5%, 90.3%, and 88.7% accuracy for 5, 10, and 16 classes, compared to NetBeacon’s decline from 95.3% to 58.5% and BoS’s deterioration from 99.2% to 53%. These results demonstrate the remarkable resilience of our method under increasing classification complexity, proving its strong potential for real-world multi-class application classification tasks.

**Answer to RQ1:** Syncedcoche’s direct matching of Key Segments not only achieves but surpasses sophisticated online deep learning methods like Brain-on-Switch, with F1-score

improvements of 8.5% on ToN-IoT and 18.3% on VisQUIC-10, while significantly outperforming statistical approaches like NetBeacon. Additionally, Synecdoche demonstrates superior scalability under increasing classification complexity.

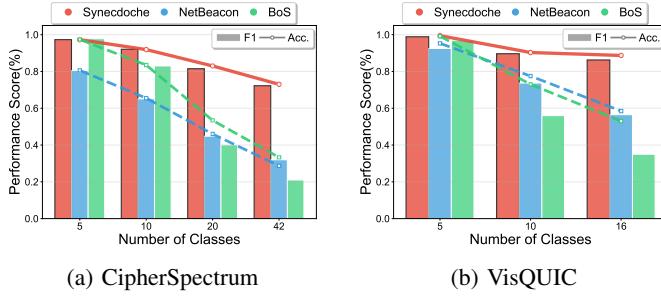


Fig. 7: Comparison of classification performance with increasing number of classes

TABLE III: Rules Complexity among Datasets

Metric	UNSW\_ToN\_IoT	Cipher\_Spectrum
Number of Classes	10	42
Range Match Rules (TCAM)	359	1485
Rules per Class	35.90	35.53
Exact Match Rules (SRAM)	1102	7223
Avg. Splits per Rule	3.07	4.86

#### D. Performance and Efficiency Analysis (RQ2)

To answer RQ2, we evaluate Synecdoche’s efficiency in terms of latency and hardware resource utilization compared to existing data plane inference methods. Table III summarizes the rule statistics among datasets. With an average of 30+ rules per class, classification is achieved effectively. After splitting range matches into exact matches, no exponential explosion occurs—each range rule expands to 3-4 exact match rules on average. We allocate 2048 TCAM and 8192 SRAM entries accordingly, with all subsequent comparisons using this configuration.

Table IV presents comprehensive performance metrics. In terms of per-flow storage requirements, our approach maintains the feature information of the most recent  $L_{max}$  packets, including: a 32-bit direction register storing the source IP of the first packet for subsequent direction determination, a 32-bit register for the latest packet timestamp, an 8-bit counter for the current packet count, and  $L_{max} \times 16$  bits for packet features. When  $L_{max} = 4$ , each flow requires 136 bits of storage., significantly outperforming both baselines. NetBeacon stores different statistical features totaling 272 bits per flow, while BoS requires 288 bits to maintain sliding window hidden states (window size=8) plus per-class cumulative scores for classification decisions. Both Synecdoche variants demonstrate superior resource efficiency in terms of SRAM utilization. The SRAM version achieves exceptional memory efficiency with only 4.03% SRAM usage, representing a remarkable 79.2% reduction compared to NetBeacon (19.38%) and an 85.8% reduction compared to BoS (28.33%).

TABLE IV: Performance Comparison of Different Methods

Method	Bits/ Flow	SRAM (%)	TCAM (%)	Stages	Latency (ns)
NetBeacon	272	19.38	31.25	12	470
BoS	288	28.33	6.94	24	937
Synecdoche (TCAM)	136	1.47	20.4	11	420
Synecdoche (SRAM)	136	4.03	1.09	8	416

Considering that switch hardware limits both the ingress and egress pipelines to 12 stages each, our Synecdoche variants demonstrate exceptional efficiency by completing the entire classification pipeline within a single pipeline direction. The SRAM version requires only 8 stages while the TCAM version uses 11 stages, both operating entirely within the ingress pipeline. NetBeacon also operates within the single-direction constraint using 12 stages. In contrast, BoS requires 24 total stages that must span both the ingress (12 stages) and egress (12 stages) pipelines. Both Synecdoche implementations achieve superior latency characteristics. The SRAM version delivers a per-packet processing time of 416 ns and the TCAM version 420 ns, which is 13.0% faster than NetBeacon (470 ns) and 125.2% faster than BoS (937 ns). This demonstrates the efficiency benefits of our segment-matching-based approach over complex statistical computation and deep learning inference.

**Answer to RQ2:** Synecdoche demonstrates superior performance in both latency and hardware resource utilization. It achieves 13.0% faster processing than NetBeacon and 125.2% faster than BoS, while reducing SRAM usage by 79.2% compared to NetBeacon and 85.8% compared to BoS, demonstrating exceptional memory efficiency.

## VII. CONCLUSION

This paper presents Synecdoche, a novel traffic classification framework that successfully bridges the accuracy-efficiency gap on programmable data plane through direct packet sequential pattern matching. By leveraging an “offline discovery, online matching” paradigm that automatically extracts Key Segments using deep learning models and deploys them as optimized table entries, Synecdoche significantly outperforms existing statistical and deep learning approaches while substantially reducing hardware resource consumption and processing latency. This work represents a fundamental advance in programmable data plane traffic classification, demonstrating that intelligent pattern extraction and hardware-optimized deployment strategies can overcome traditional accuracy-efficiency trade-offs and pave the way for sophisticated packet sequence analysis at line rate in emerging network applications. Our future work will continue to explore the impact of network fluctuations on Key Segment effectiveness and investigate mechanisms for efficient rule updates in deployed systems.

## ACKNOWLEDGMENT

This work was supported by The Cybersecurity Professional Committee of the Chinese Association of Educational Technology(NO. 2023CAET1002)

## REFERENCES

- [1] H. Tahaei, F. Afifi, A. Asemi, F. Zaki, and N. B. Anuar, "The rise of traffic classification in IoT networks: A survey," *Journal of Network and Computer Applications*, vol. 154, p. 102538, Mar. 2020.
- [2] M. Shen, K. Ye, X. Liu, and L. Zhu, "Machine Learning-Powered Encrypted Network Traffic Analysis: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 1, pp. 791–824, 2023.
- [3] O. Michel, R. Bifulco, G. Retv'ari, and S. Schmid, "The programmable 'data plane': Abstractions, architectures, algorithms, and applications," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–36, 2021.
- [4] W.-X. Liu, C. Liang, Y. Cui, J. Cai, and J.-M. Luo, "Programmable data plane intelligence: Advances, opportunities, and challenges," *IEEE Network*, vol. 37, no. 5, pp. 122–128, 2023.
- [5] P. Bosshart, D. Daly, G. Gibb and M. Izzard, "P4: programming protocol-independent packet processors," *SIGCOMM*, vol. 44, no. 3, pp. 87–95, July 2014.
- [6] Y. Zhang, S. Yao, Y. Feng, K. Chen, and T. Li, "Pegasus: A Universal Framework for Scalable Deep Learning Inference on the Dataplane," Jun. 06, 2025, arXiv: arXiv:2506.05779. doi: 10.48550/arXiv.2506.05779.
- [7] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium(USENIX Security 23)*, 2023, pp. 6203–6220.
- [8] A. T.-J. Akem, M. Guicciardo, and M. Fiore, "Flowrest: Practical flow-level inference in programmable switches with random forests," in *IEEE INFOCOM 2023*.
- [9] S. U. Jafri, S. Rao, V. Shrivastav, and M. Tawarmalani, "Leo: Online ML-based traffic classification at Multi-Terabit line rate," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024, pp. 1573–1591.
- [10] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *IEEE INFOCOM 2022*, pp. 1938–1947.
- [11] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, and R. Bensoussane, "IIsy: Hybrid In-Network Classification Using Programmable Switches," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, 16 Feb. 2024, pp. 2555–2570.
- [12] Z. Zhang, Z. Luan, Q. Li, Z. Qi, and K. Li "SentinelX: A Lightweight Malicious Traffic Detection System Based on Programmable Switches," in *IEEE INFOCOM 2025*, London, United Kingdom, 2025.
- [13] Z. Zhao, Z. Li, Z. Song, F. Zhang and B. Chen, "RIDS: Towards Advanced IDS via RNN Model and Programmable Switches Co-Designed Approaches," in *IEEE INFOCOM 2024*, Vancouver, BC, Canada, 2024, pp. 591–600.
- [14] M. Zhang, L. Cui, X. Zhang, F. P. Tso and Z. Zhen "Quark: Implementing Convolutional Neural Networks Entirely on Programmable Data Plane," in *IEEE INFOCOM 2025*, London, United Kingdom, 2025.
- [15] J. Yan, H. Xu, Z. Liu, Q. Li, K. Xu, M. Xu, and J. Wu, "Brain-on-Switch: Towards advanced intelligent network data plane via NN-Driven traffic analysis at line-speed," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2024, pp. 419–440.
- [16] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications," in *Proceedings 2021 Network and Distributed System Security Symposium(NDSS)*, Virtual: Internet Society, 2021.
- [17] J. Lin, Q. Li, G. Xie, Y. Jiang, and Z. Yuan, "In-Forest: Distributed In-Network Classification with Ensemble Models," in *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, Reykjavik, Iceland: IEEE, Oct. 2023, pp. 1–12.
- [18] J. Piet, D. Nwoji, and V. Paxson, "Ggfast: Automating generation of flexible network traffic classifiers," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 850–866.
- [19] K. Fauvel, F. Chen, and D. Rossi, "A Lightweight, Efficient and Explainable-by-Design Convolutional Neural Network for Internet Traffic Classification," in *Proc. ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2023, pp. 4013–4023.
- [20] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *International Conference on Computer Vision (ICCV)*, 2017, pp. 618–626.
- [21] N. Koroniots, N. Moustafa, E. Sitnikova, and B. Turnbull "Towards the Development of Realistic Botnet Dataset in the Internet of Things for Network Forensic Analytics: Bot-IoT Dataset," *Future Generation Computer Systems*, vol. 100, Nov. 2019, pp. 779–796.
- [22] N. Moustafa, "A new distributed architecture for evaluating ai-based security systems at the edge: Network ton\_iot datasets," *Sustainable Cities and Society*, vol. 72, p. 102994, Sep. 2021.
- [23] N. Wickramasinghe, A. Shaghaghi, G. Tsudik, and S. Jha, "SoK: Decoding the Enigma of Encrypted Network Traffic Classifiers," in *2022 IEEE Symposium on Security and Privacy (SP)*, 12 May 2025, pp. 1825–1843.
- [24] B. Gahtan, R. J. Shahla, A. M. Bronstein, and R. Cohen, "Exploring QUIC Dynamics: A Large-Scale Dataset for Encrypted Traffic Analysis," May 24, 2025, arXiv: arXiv:2410.03728. doi: 10.48550/arXiv.2410.03728.
- [25] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, "Accurate Decentralized Application Identification via Encrypted Traffic Analysis Using Graph Neural Networks," *IEEE Transactions on Information Forensics and Security*, vol. 16, 2021, pp. 2367–2380.
- [26] S.-J. Xu, G.-G. Geng, X.-B. Jin, D.-J. Liu, and J. Weng, "Seeing Traffic Paths: Encrypted Traffic Classification With Path Signature Features," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 2166–2181, 2022.
- [27] Y. Wang, H. He, Y. Lai, and A. X. Liu, "A Two-Phase Approach to Fast and Accurate Classification of Encrypted Traffic," *IEEE/ACM Transactions on Networking*, vol. 31, no. 3, pp. 1071–1086, June 2023.
- [28] X. Yun, Y. Wang, Y. Zhang, C. Zhao, and Z. Zhao, "Encrypted TLS Traffic Classification on Cloud Platforms," *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 164–177, Feb. 2023.
- [29] Z. Song, Z. Zhao, F. Zhang, G. Xiong and G. Cheng, "I<sup>2</sup> RNN: An Incremental and Interpretable Recurrent Neural Network for Encrypted Traffic Classification," *IEEE Transactions on Dependable and Secure Computing*, Dependable and Secure Comput., pp. 1–14, 2024.
- [30] Rezaei, Shahbaz, and Xin Liu. "Deep Learning for Encrypted Traffic Classification: An Overview," *IEEE Communications Magazine*, vol. 57, no. 5, May 2019, pp. 76–81.
- [31] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "FS-Net: A Flow Sequence Network For Encrypted Traffic Classification," in *IEEE INFOCOM 2019*, Paris, France: IEEE, Apr. 2019, pp. 1171–1179.
- [32] W. Zheng, C. Gou, L. Yan, and S. Mo, "Learning to Classify: A Flow-Based Relation Network for Encrypted Traffic Classification," in *Proc. Web Conf. 2020*, Apr. 2020, pp. 13–22.
- [33] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.
- [34] I. Akbari, M. A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, "Traffic classification in an increasingly encrypted web," *Communications of the ACM*, vol. 65, 2022.
- [35] Z. Aouini and A. Pekar, "Nfstream: A flexible network data analysis framework," *Computer Networks*, vol. 204, p. 108719, Feb. 2022.
- [36] C. Liu et al., "FS-Net," <https://github.com/WSPITH/FS-Net>.
- [37] G. Zhou et al., "Netbeacon," <https://github.com/IDP-code/NetBeacon>.
- [38] J. Yan et al., "Brain-on-Switch," <https://github.com/InspiringGroup-Lab/Brain-on-Switch>.
- [39] Y. Han, Q. Li, G. Xie, Gareth Tysopn and Y. Jiang, "Linc: Enabling Low-Resource in-Network Classification and Incremental Model Update" in *IEEE 32nd International Conference on Network Protocols (ICNP)*, Charleroi, Belgium, 2024, pp. 1–12.