

Real-Time and Trustworthy Classification of IoT Traffic Using Lightweight Deep Learning

Arunan Sivanathan, Deepak Mishra, Sushmita Ruj, Natasha Fernandes, Quan Z. Sheng, Minh Tran, Ben Luo, Daniel Coscia, Gustavo Batista and Hassan Habibi Gharakaheili

Abstract—The rapid growth of Internet-of-Things (IoT) devices in enterprise and industrial networks presents significant challenges for device behavior analysis and security. Existing machine learning models for IoT traffic classification face limitations. Shallow models, relying on manually engineered features, struggle to capture complex, nonlinear patterns and generalize across diverse environments. Deep pre-trained models, such as transformers, demand extensive preprocessing and significant computational resources, making them less practical for real-time inference in resource-constrained environments. This paper proposes a lightweight, real-time deep learning model based on convolutional neural networks (CNNs) that classifies IoT traffic using structured flow sequences, providing an efficient and reliable solution.

Our contributions are threefold: (1) We develop a novel structure of flow data sequences that represents IoT network behavior as a fixed-size matrix, capturing flow metadata, packet timing, direction, and raw payloads. This flexible structure ensures adaptability to diverse IoT environments and enables the classification of a wide variety of devices. We publicly release our structured dataset derived from real traffic traces. (2) We propose a convolutional neural network (CNN) architecture that captures both intra-flow and inter-flow patterns, providing an efficient solution for real-time IoT traffic classification. We evaluate three traffic inference strategies across four performance metrics, namely accuracy, coverage, computational cost, and traffic selectivity, demonstrating the method's effectiveness for real-time IoT traffic analysis. (3) We incorporate interpretability techniques, specifically confidence scores and Shapley values, to assess and enhance the trustworthiness of predictions. These insights refine the predictions, yielding a 4% boost in macro-averaged F1-score. They also significantly reduce high-confidence misclassifications to one-fifth when applied to real IoT traffic traces.

Index Terms—Modelling, network behavior sequence, IoT devices, deep learning.

I. INTRODUCTION

ENTERPRISE and operational networks are increasingly onboarding a specific class of devices known as Internet-of-things (IoT) devices, such as cameras, sensors, printers, and power switches. Unlike traditional information technology

A. Sivanathan, D. Mishra, and H. Habibi Gharakaheili are with the School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, NSW 2052, Australia (e-mails: {a.sivanathan, d.mishra, h.habibi}@unsw.edu.au).

S. Ruj and G. Batista are with the School of Computer Science and Engineering, University of New South Wales, Sydney, NSW 2052, Australia (e-mails: {sushmita.uj, g.batista}@unsw.edu.au).

N. Fernandes and Q.Z. Sheng are with the School of Computing, Macquarie University, Sydney, NSW 2109, Australia (e-mails: {natasha.fernandes, michael.sheng}@mq.edu.au).

M. Tran, B. Luo and D. Coscia are with Information Sciences Division, Defence Science & Technology Group, Edinburgh, SA 5111, Australia (e-mails: {minh.tran7, ben.luo, daniel.coscia1}@defence.gov.au).

(IT)¹ devices, IoT devices exhibit: (a) high heterogeneity in hardware, operating systems, and firmware; (b) limited computing resources aligned with their specialized functions; (c) reliance on open-source or third-party components to save costs, which may introduce vulnerabilities; (d) minimal support for remote management or software agents; and, (e) sensitivity to active scans, making traditional asset discovery techniques ineffective. These characteristics complicate the management and security of IoT devices, which often expand the attack surface of the networks they join.

Despite these challenges, IoT devices exhibit distinct and predictable network behavior patterns [1] due to their specialized functions. Their traffic patterns, ranging from endpoint choices [2] to timing regularities [3] and protocol usage [1], can be leveraged for passive monitoring and classification. Consequently, machine learning (ML) models have been explored to infer device types and behaviors based on observed benign network traffic. Building an accurate and up-to-date network device registry (asset mapping) is a foundational step for several network management tasks, including cybersecurity and risk analysis.

However, existing ML approaches face certain limitations. Shallow models, such as decision trees and logistic regression, are computationally efficient and cost-effective but heavily rely on manually engineered features. This dependency limits their ability to capture complex, non-linear relationships within traffic data. Furthermore, their adaptability to dynamic environments is limited [4], making it challenging for them to generalize even under minor variations in traffic patterns. On the other hand, deep pre-trained models [5]–[7], such as those based on transformers or large neural networks, can deliver strong predictive performance, but require substantial amounts of data and extensive traffic preprocessing. These demands make them less suitable for real-time environments with constrained resources, such as edge devices or embedded systems within IoT networks, where computational resources, memory, and latency are limited. As a result, trade-offs involved in current ML-based solutions and practices limit their broader applicability.

In this paper, we argue that addressing this gap requires a unified data structure and a flexible learning architecture capable of capturing diverse behavioral patterns. A versatile data structure enables the comprehensive incorporation of traffic information, from header metadata to packet timing and

¹This refers to devices, such as personal laptops, smart phones, desktop computers, and servers.

TABLE I
OBJECTIVE AND FOCUS OF PRIOR WORK ON IOT TRAFFIC ANALYSIS.

Work	Objective		Traffic Focus			
	Asset Mapping	Anomaly Detection	Telemetry Statistics	Flow Metadata	Protocol Specific Attributes	Raw Packet Payload
[1]	✓		✓	✓	✓	
[3]		✓		✓		
[8]	✓		✓			
[9]	✓	✓	✓	✓		
[10]		✓	✓	✓	✓	✓
[11]	✓					✓
[12]	✓		✓	✓		
[13]		✓		✓		
[14]	✓			✓		
[15]	✓		✓			
[16]	✓				✓	
[17]	✓	✓	✓		✓	

payload bytes, while a flexible learning architecture allows models to detect subtle patterns within the dynamic traffic data, facilitating effective adaptation when needed. It is important to note that an in-depth understanding often requires extensive data. When deploying ML models, it is important to assess whether their predictions can be trusted. Interpretability methods [18]–[20] help show how a model makes its decisions, while trustworthiness reflects the confidence that these predictions are reliable in practice. By combining interpretability techniques (Shapley values) with prediction confidence scores, we link the “*why*” (interpretability) to the “*can it be relied upon*” (trustworthiness). Note that we use the term trustworthiness in this narrow sense of prediction reliability, different from broader trustworthy learning research [21], that encompasses aspects such as causality, fairness and privacy.

This paper makes three specific contributions. Our **first** contribution (§III) develops a configurable data structure that represents network traffic in a fixed-size matrix, where each row corresponds to a flow (up to a configurable number of past flows) and includes metadata such as flow timing, transport and network headers, volume statistics, and a sequence summarizing packets by direction, timing, and raw payloads. This structure uses explicit demarcation to retain traffic semantics in an organized and well-structured manner. By analyzing a large dataset of IoT packet traces, we generate our structured traffic data, which we will publicly release, and we gain insight into behavioral patterns exhibited by IoT devices. Our **second** contribution (§IV) develops a two-dimensional convolutional neural network (2D-CNN) architecture to infer device-specific patterns from the structured traffic representation we use as input. This architecture models traffic patterns within individual flows (intra-flow) as well as across sequences of flows (inter-flow) without the need for specialized supervision regarding communication protocols. We experiment with three representative strategies for traffic inference and highlight their pros and cons based on four metrics: accuracy, coverage, computational workload, and traffic selection needs. Finally, our **third** contribution (§V) develops a method that utilizes confidence scores in conjunction with Shapley values to assess and verify the trustworthiness of predictions. When applied to real traffic traces, we demonstrate how the interpretability

insights derived from the predictions can refine the 2D-CNN model, leading to higher-quality predictions.

II. RELATED WORK

IoT Traffic Analysis: Over the past decade, researchers have increasingly focused on studying the behavior of IoT devices, particularly analyzing and modeling the network traffic these devices generate [26]. The scope of analysis has ranged from examining specific packet types (*e.g.*, TCP, HTTP, DNS) to analyzing all packets or broader traffic flows, depending on constraints such as environmental settings (*e.g.*, industrial, home automation, enterprise), available computing resources, or protocol expertise. This research has supported a wide range of use cases, including asset mapping [1], [8], cyber health monitoring [9], anomaly detection [3], and the identification of specific attacks [10]. Table I groups prior relevant studies by to their primary objectives: asset mapping, anomaly detection, or combined. Although relatively successful, inference models are often highly customized to achieve the highest prediction accuracy in specific settings, making it difficult to generalize or adapt them in diverse network environments shaped by varying device types and configurations [27]. Furthermore, their heavy reliance on human expertise for feature design can overlook subtle but distinctive patterns in traffic data that are not easily apparent to humans [27].

An effective method [17] for IoT traffic analysis is expected to balance prediction accuracy and design flexibility, allowing it to adapt in varied settings where computing resources, responsiveness needs, and device compositions differ widely. Table I also highlights each prior work focuses on which traffic attributes across four pillars: (1) telemetry statistics over entire traffic or specific flows; (2) flow metadata, comprising network and transport-layer header fields such as IP addresses and port numbers; (3) protocol-specific attributes, derived from application-layer protocols (*e.g.*, domain names, HTTP agents); and (4) raw packet payload, representing the undecoded payload bytes of packets. Approaches to managing the trade-off between network behavior inclusivity and computational cost differ: raw byte analysis [11] provides detailed insights into behavioral patterns but demands high computing resources, while flow-level analysis using traditional IPFIX records [12] or newer programmable telemetry like Open-

TABLE II
SUMMARY OF LITERATURE ON NEURAL NETWORK-BASED INFERENCE.

Work	Traffic Classification Techniques	Limitations
[22], [23] [23] [24]	Neural networks for automatic feature learning. CNN+transformer: flow data transformed into grayscale images. LSTM using packet size, timing, ports (no payload).	Black-box; low interpretability. No semantic grounding; weak robustness across environments. Omits payload; high-dim payloads are costly; encrypted data breaks assumptions.
[5] [6] [7] [25]	Large transformer with 25–100m parameters. Efficient alternative to transformers, tokenized packet metadata. Lightweight, flow-level summarization. Critique of DNNs and large transformers in traffic analysis.	Heavy computing costs; impractical at edge. Still resource heavy; lacks semantic visibility. Coarse features; loses packet/payload details; GPU hungry. Black-box nature persists.

Flow [13] offers a concise summary of flow metadata [14], though it sacrifices some finer-grained behavioral patterns in network traffic. Additionally, abstract statistical metrics, such as traffic rates or packet counts [15], are computationally efficient but may not be much predictive, as they can be influenced by network settings, characteristics, and neighboring devices.

Instead of committing to one side of the trade-off at the start of the inference process, we propose a customizable data record that enables operators to incorporate richer contextual traffic behavior wherever desirable or practical. This approach provides a balanced and generalizable alternative to individual packets, which offer detailed insights but are costly and lack broader context, and coarse-grained flow statistics, which are more scalable but have limited predictive power.

Some researchers have proposed focusing on specific subsets of network traffic (*e.g.*, HTTP, DNS) [16], a strategy that helps manage computing costs while enabling the explicit extraction of distinctive protocol-specific features (*e.g.*, domain name [1], user agent [17]) from traffic payloads to classify IoT traffic. While effective for devices that frequently use those protocols, their specialized approach is not generalizable to many devices that rarely or never use those protocols for communication [28]. In contrast, our approach allows for the selective inclusion of packet-level details (*e.g.*, packet size, arrival time, and even raw bytes from payload) alongside flow-level metadata, all while remaining agnostic to application-layer protocols. This enables operators to fine-tune inference models by incorporating costly payload data proportionally, balancing prediction quality with available computing resources—without requiring protocol-specific expertise.

Neural Network-Based Inference: Deterministic signature matching [29] often struggles to account for the dynamic behaviors of IoT devices [30]. Consequently, ML techniques, such as decision trees [1], clustering [9], and neural networks [22], [23], have become increasingly popular. Unlike traditional ML, which depends on manually selected features, deep neural networks (DNNs) [23], [24] can automatically capture both prominent and subtle features that may escape human observation, resulting in a more adaptable and flexible architecture. This adaptability allows models to be retrained or refined with minimal manual intervention, making them suitable for new environments without requiring a complete overhaul.

Recent efforts have applied neural networks to network traffic classification with varying assumptions and architectures. One study [23] reshapes the first 784 bytes of flow

data, excluding Internet Protocol (IP) and Medium Access Control (MAC) addresses, into a 28×28 grayscale image for input to a CNN and transformer stack. While visually inspired, this approach lacks semantic structuring and treats traffic like pixel data, limiting its interpretability and robustness across diverse settings. Another approach [24] uses Long Short-Term Memory (LSTM) to classify application-layer protocols using lightweight features like packet sizes, timing, and ports, but omits payloads, missing potentially critical indicators for distinguishing fine-grained device behaviors. Including payload data for LSTM models presents challenges due to the high dimensionality and variability of the raw payload content. In addition to demanding extensive preprocessing resources, encrypted data can introduce complications to the temporal modeling assumptions inherent to LSTMs.

At the other end of the spectrum, large models such as ET-BERT [5] and YaTC [6], which rely on pre-trained transformer architectures, and NetMamba [7], which is based on a pre-trained state-space model architecture, contain 25–100 million parameters. While these models achieve strong classification accuracy, approaches like ET-BERT incur substantial overhead during preprocessing and inference, limiting their practicality for real-time or resource-constrained environments. YaTC offers a more efficient alternative, but still relies on tokenized packet-level metadata. In contrast, NetMamba is lighter and faster, but fundamentally constrained by its use of coarse flow-level summaries, and like the others, it lacks visibility into packet payloads and deeper application semantics. The authors of a recent study [31] argue that the high accuracy of representation learning approaches often stems from shortcut learning and data leakages rather than a genuine semantic understanding of traffic patterns. They also claim that simpler ML models often outperform deep-representation models. In line with this perspective, we develop a lightweight CNN-based classifier with only 200 thousand parameters, which avoids tokenization overhead, directly consumes raw traffic features, and achieves competitive accuracy while remaining practical for real-time and resource-constrained environments.

While neural networks offer high accuracy and adaptability, their complexity often results in a “black box” effect [25], where it is difficult to interpret and/or verify how specific features influence predictions. Interpretability is crucial, especially for network operators monitoring critical production environments. Although the interpretability of predictions has gained traction in fields such as healthcare [32] and finance [33], they are less common in network traffic analysis.

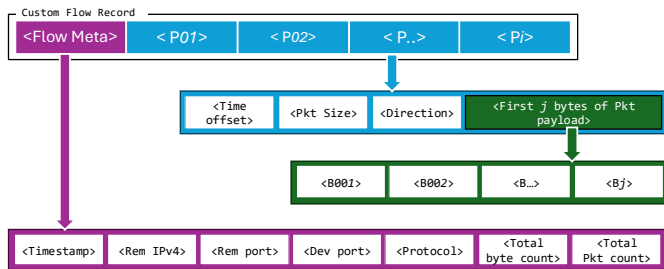


Fig. 1. Structure of custom flow records constructed from raw network traffic.

Techniques like SHAP (SHapley Additive exPlanations) [18], LIME (Local Interpretable Model-agnostic Explanations) [19], and gradient-based methods (such as Gradient Tape) [20] are widely used to interpret predictions. While gradient-based approaches offer efficient feature importance calculations, they lack robustness. LIME provides model-agnostic explanations, but can be unstable, while SHAP offers consistent local and global interpretability at a computational cost. In this work, we employ SHAP to assess model prediction reliability and build trust in its output. Table II provides a compact summary of the existing literature on neural network-based traffic classification, highlighting their core techniques and key limitations.

Our Novelty: Previous work on IoT device classification has employed machine learning and traditional approaches to infer device types and behavioral patterns. However, these methods typically rely on narrowly defined traffic features or computationally expensive representations, limiting their applicability in diverse, real-world, and resource-constrained environments [27]. In contrast, our work involves three key innovations: **(1)** A novel flow-based traffic representation that directly incorporates raw traffic signals, including flow metadata, fine-grained packet features, partial payload bytes, and temporal sequencing, without the need for deep encoders or protocol-specific feature engineering. This empirically validated representation captures semantically rich, protocol-agnostic patterns and generalizes effectively across diverse IoT devices (§III). **(2)** A lightweight CNN-based classifier ($\approx 200K$ parameters) that efficiently models intra-flow and inter-flow dynamics, allowing accurate real-time inference in embedded devices and edge devices (§IV). **(3)** A framework that combines SHAP-based interpretability with confidence scores to enhance trust in model predictions and guide iterative refinement of data representation (§V).

III. IOT NETWORK BEHAVIOR CHARACTERIZATION

This section defines the behaviors a network device may exhibit in its traffic. We then develop a generic structure to represent traffic data instances, enabling data-driven models and algorithms to make inferences from network traffic patterns.

A. Traffic Data Representing Network Behavior

Analyzing patterns in network traffic can be done at both micro and macro levels. At the micro level, examining byte values in the headers and payload of individual packets can

offer valuable insights into specific behavioral patterns. However, this method is often computationally intensive and may not capture the broader communication context. At the macro level, collecting and processing flow records is more cost-effective. A network flow is a sequence of packets sharing common properties, such as source/destination IP addresses, source/destination port numbers, and protocol (*e.g.*, TCP or UDP). Flow records may not always offer the detailed insights needed for fine-grained inferences, such as classifying a wide range of diverse devices and applications. Therefore, we aim to leverage both approaches to balance flow metadata with packet payloads. In this paper, we develop “custom” bidirectional flows to capture a comprehensive picture of behaviors. Our custom flows incorporate metadata (key header fields), statistical summaries, packet timestamps and directions, and selected payload bytes from each packet.

Standard methods like NetFlow [34], IPFIX [35], and OpenFlow [36] telemetry are commonly used to collect flow records (metadata and high-level statistics) from network traffic. A traffic flow is exchanged between two or more endpoints (including multicast or broadcast scenarios), potentially capturing partial behaviors of these endpoints. Previous research [37] has typically focused on unidirectional or bidirectional flows associated with the initiator endpoint. However, this approach can lead to issues and confusion for inference models, particularly when two network devices (or endpoints) exchange traffic and the goal is to determine the class of both devices (endpoints). For traffic exchanged locally between two devices on the monitored network (*i.e.*, device-to-device communications), we duplicate the bidirectional custom flow and adjust each version to reflect the perspective of one endpoint (noting that outgoing packets from one endpoint are counted as incoming packets for the other), ensuring a consistent representation of the communication between the two devices. However, when a network device communicates with a cloud server (outside the local network), the bidirectional custom flow is considered only once and is associated with that local device.

We identify each custom flow by a five-tuple: device IP address, remote IP address, protocol, device port, and remote port, with a lifetime of one minute. We note that very long flows, such as those (TCP-based) maintained by some IoT devices with their cloud servers, can be computationally intensive and may impact system responsiveness. We, therefore, export our custom flow records after one minute to ensure efficient real-time computation and consistent analysis of both TCP and UDP communications. Any traffic extending beyond one minute is captured by a new flow record with the same five-tuple, ensuring no data is unprocessed. Also, instead of using the terms “client” and “server”, we opt for “device” and “remote”, as distinguishing between the client and server sides in real-time can be challenging, particularly when the UDP protocol is used. While this distinction is somewhat easier in TCP communications, tracking SYN/FIN states can be complicated by the potential loss of critical packets during measurement.

We recognize that TCP headers may exhibit specific patterns [38]. However, we have chosen not to include them in our analysis to maintain generalizability. While network-

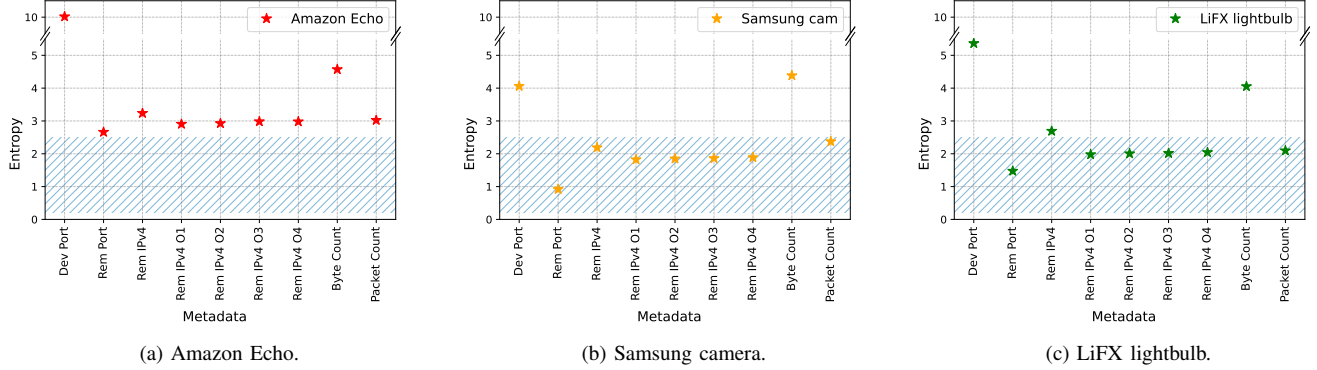


Fig. 2. Entropy measures of values for the metadata fields (device-port, remote-port, remote-IPv4, packet count, and byte count of custom flows) for three representative IoT device types in our dataset, (a) Amazon Echo, (b) Samsung camera, and (c) LiFX lightbulb, based on a week’s worth of data.

TABLE III
SUMMARY OF OUR CUSTOM FLOW DATA.

IoT Device Type	# Custom Flows
Amazon Echo	777,587
Belkin motion sensor	970,940
Belkin power switch	642,451
Dropcam	171,929
HP printer	238,831
iHome power plug	17,026
Insteon camera	1,095,843
LiFX lightbulb	190,632
NEST Protect	989
Netatmo camera	308,053
Netatmo weather	76,972
PIX-STAR photoframe	24,197
Samsung camera	620,912
Smart Things	195,540
TP-Link camera	55,959
TP-Link power plug	17,729
Triby speaker	164,076
Withings baby monitor	63,304
Withings scale	805
Withings sleep sensor	96,332
IT (android tablet)	235,311

layer payloads could potentially capture these patterns, this paper focuses on the transport-layer payload. By doing so, we allocate computing resources to uncover hidden characteristics within the application layer, which is likely to contain distinct patterns that reveal functionality, manufacturer details, or specific software modules in use.

Fig. 1 illustrates the structure of our custom flow records constructed from raw network traffic. Each custom flow includes the following metadata: the `timestamp` (in microseconds) of the first packet in the Unix timestamp format, the `remote IPv4` address, the transport layer `protocol`, the `device-side port`, the `remote-side port`, total byte count and total packet count. Note that we obfuscate local unicast IP address ranges (*e.g.*, in our settings, private address blocks: 10.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12) of the remote endpoint by replacing them with 0.0.0.0. To support generalizability, we ensure that deployment-specific information (*i.e.*, local IP addresses) and non-semantic identifiers (*i.e.*, MAC addresses) are not included in our custom flow records.

Additionally, we collect fine-grained data from packet pay-

loads, which can reveal distinctive behavioral patterns. It is important to note that the number of packets in a flow and the payload size of each packet can vary significantly. Analyzing all packets within a flow would be prohibitively expensive. Prior research [39] has shown that behavioral fingerprints often appear within the initial bytes of the first few packets of a flow. Therefore, we record specific information from the first i packets of each flow. For every packet included in our custom flow, we capture the time offset from the flow’s first-seen timestamp, the total packet size, and a flag indicating the packet direction from the device’s viewpoint (1 if the packet is sent by the device, 0 if sent by the remote endpoint). Additionally, we extract the payloads of these first i packets from each flow, and from each packet, capturing up to j bytes of the transport layer payload as a byte array. Both i and j can be configured based on specific needs, traffic mix, and available computing resources. In our dataset, we have found that 92% of all 1-minute flows contain 10 or fewer packets, and 96% of packets are 1,000 bytes or smaller. Therefore, for our analysis in this paper, we start with the choice of $i = 10$ and $j = 1,000$.

It is important to note that we do not make any assumptions about the payload data, whether encrypted, encoded, or in plaintext [40]. By including payload data bytes in our custom flows, we enable the learning model to opportunistically extract patterns (strong or weak) that indicate IoT device behaviors.

B. IoT Traffic Dataset and Behavioral Insights

Dataset: This paper uses a public dataset of full packet traces, which we released in our previous work [1]. The dataset includes 60 days of packet traces collected from 22 types of consumer IoT devices, such as cameras, lightbulbs, power plugs, motion sensors, appliances, and health monitors. We analyzed these packet captures (PCAP files) and extracted over 5.9 million custom flows. Table III summarizes the number of custom flows across the different device types. We release our dataset of custom flows for the research community to use [41]. It is important to note that the activity level varies significantly across devices. For instance, devices like the Amazon Echo, Insteon camera, and Belkin motion sensor are

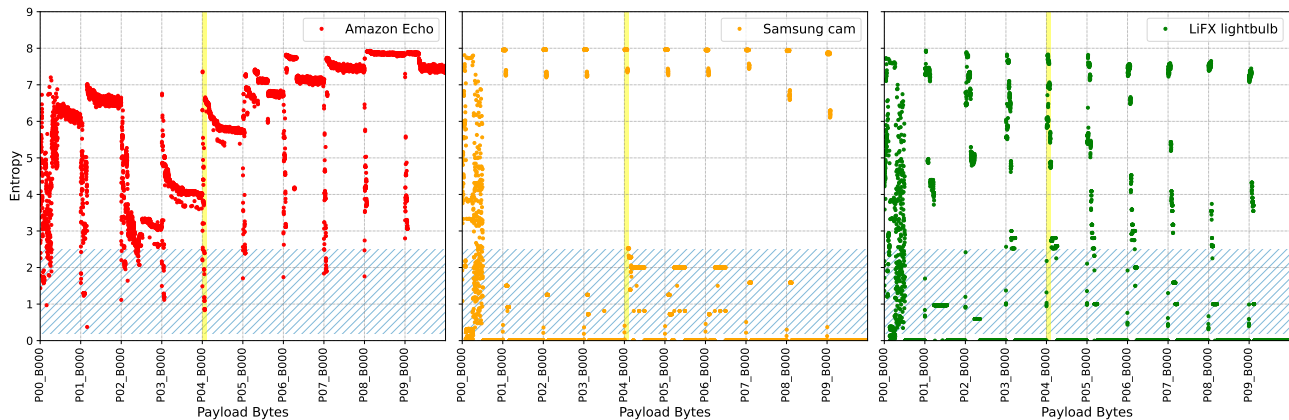


Fig. 3. Entropy measure of values for the payload fields (first 1,000 bytes of first ten packets of custom flows) for three representative IoT device types in our dataset, based on a week’s worth of data.

highly active, each contributing around one million flows. In contrast, other devices, such as the Withings scale and NEST Protect, are far less active, generating fewer than 1,000 custom flows.

Traffic Fingerprint Insights: We now closely analyze a subset of our data to extract high-level insights from the unique fingerprints embedded in different sections of our custom flows, enabling us to differentiate between various types of IoT devices.

Let us select three representative classes of IoT devices (*i.e.*, Amazon Echo, Samsung camera, and LiFX lightbulb) and analyze a week’s worth of their traffic data. Previous research has shown that metadata fields, such as traffic activity and volume [1], server-side IP addresses [42], and transport-layer services [12], can reveal behavioral characteristics of various IoT device types and the cloud servers with which they communicate.

In Fig. 2, we present the entropy measures of specific fields in the flow metadata, including device-port, remote-port, remote-IPv4 (comparing a full 32-bit address with its four 8-bit octets), byte count, and packet count. The entropy of each data field measures the randomness of the values across all flows in our representative sample, which consists of one week of data for each type of IoT device. The entropy computation is based on the probability distribution of each symbol within a given message.

$$H(x) = - \sum_{i=1}^n P(x_i) \log_2 P(x_i) \quad (1)$$

where the message consists of n distinct symbols (values), denoted as x_i . Given a symbol x_i , the probability $P(x_i)$ is determined by the frequency of occurrences of x_i within the message, divided by the total count of symbols in the message.

Higher entropy values indicate greater randomness, making it more difficult to extract identifiable fingerprints. In contrast, lower entropy values suggest less randomness and increase the likelihood of identifying fingerprints from that specific data field. It is important to note that insufficient data, such as NaN values, result in zero entropy.

In Fig. 2, we have used blue-hatched patterns to highlight the low-entropy region (entropy values between 0.2 and 2.5)

to demonstrate how different data segments exhibit distinct identifiable fingerprints. We generally expect the device port number, a 16-bit value, to exhibit high entropy due to its inherent randomness. This is mainly because IoT devices often initiate connections to their servers. This expectation is confirmed in Amazon Echo traffic, which shows an average entropy of 10. However, the Samsung camera and LiFX lightbulb behave differently, displaying mid-range entropy values between 4 and 6 for their device port numbers. This indicates that their port number selection is more predictable, which is not ideal from a security standpoint. In protocols like DNS, such predictability can increase the risk of cache-poisoning attacks [43]. In contrast, the remote port number is expected to be more predictable, likely corresponding to specific network services on the server. This characteristic is evident in the traffic of all three representative devices, with the Samsung camera exhibiting it more prominently than the other two. Moving to the remote IPv4 address, this field in the flow metadata shows promise for revealing distinct patterns with relatively low entropy. We have observed that remote IPv4 addresses in Amazon Echo traffic appear more dynamic or random than those from the Samsung camera and the LiFX lightbulb. Additionally, analyzing the remote IPv4 address as four separate octets (O1-O4) provides slightly more information (lower entropy) than treating it as a single 32-bit number. The left octets (O1 and O2), representing the subnet, reveal better fingerprints (with slightly lower entropy) than the right octets. Finally, as expected, traffic volume—encompassing both byte count and packet count—reveals some identifiable patterns. However, the packet count proves to be more promising across the three representative devices in our dataset, as it yields a relatively lower entropy measure.

Now, let us focus on the payload sections of our custom flow record. Fig. 3 shows the entropy measurements for the first 1,000 bytes of payloads in the first ten packets of custom flows from three representative IoT devices. It can be seen that custom flows from the Samsung camera and LiFX lightbulb contain smaller packets, as indicated by the zero entropy values appearing in the tail sections of the payloads. In contrast, no zero entropy is observed in any section of the Amazon Echo packets. This suggests that the packets in Amazon Echo

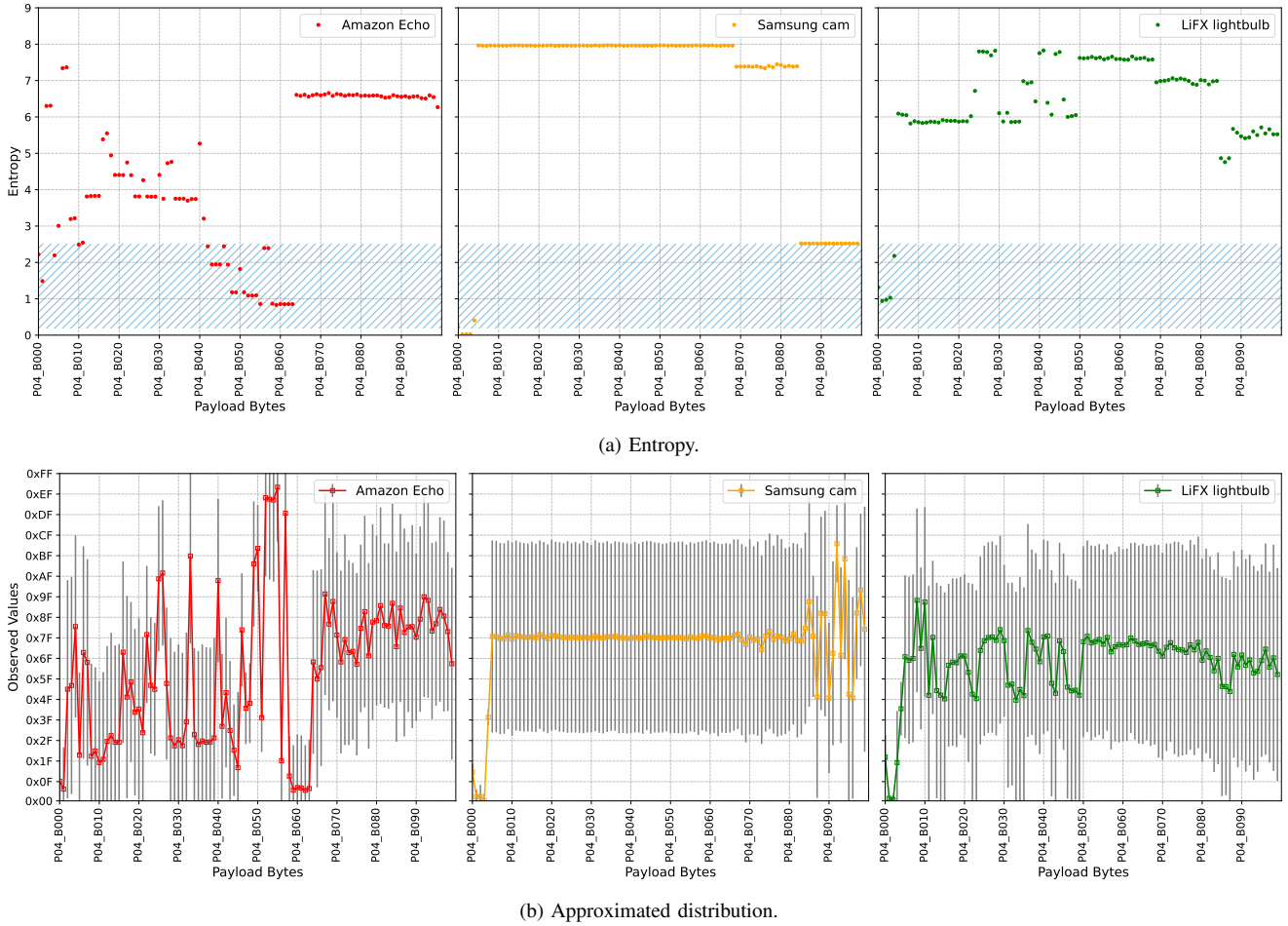


Fig. 4. Characteristics: (a) entropy measure, and (b) distribution (approximated by μ and σ) of values for the first 100 bytes of the fifth packet (*i.e.*, $P04_B00$ — $P04_B99$) in custom flows of three representative IoT device types in our dataset.

traffic are typically larger, likely containing 1,000 bytes or even more. Another observation is that the average entropy of payloads increases beyond the fourth packet as the flow of Amazon Echo progresses. This suggests that the initial packets will likely provide identifiable patterns for the Amazon Echo. In contrast, the Samsung camera and LiFX lightbulb appear more promising across all packets, as every packet from these devices contains sections with relatively low entropy measures. To gain deeper insights, we closely examine the first 100 bytes of packet $P04$, highlighted in yellow in Fig. 3.

Fig. 4b presents the value distribution of bytes (approximated by mean and standard deviation), corresponding to those shown in Fig. 4a. In Fig. 4b, the markers represent the mean of the byte values, while the error bars indicate the standard deviation. A small standard deviation suggests that the byte values are concentrated near the mean, indicating lower entropy, whereas a large standard deviation implies a more even distribution of byte values, indicating higher entropy.

In Fig. 4a, the Amazon Echo exhibits some low entropy values in the payloads of the first six packets. These low entropy byte positions are characterized by narrow byte ranges around their mean values (*e.g.*, the entropy measure, the mean, and the standard deviation of $P04_B060$ are 0.86, $0x0A$, and 41, respectively). However, the Samsung camera shows a very

low entropy of 0.03 for the first five bytes, with small standard deviation values of 6.

We note that beyond the fifth byte, the Samsung camera exhibits very high entropy up to the 60th byte, with a mean value of approximately 127 and a relatively consistent standard deviation of around 75. This indicates a high degree of randomness in this byte range, likely due to well-encrypted payloads [40]. Despite this randomness (as indicated by large entropy measures), the sequence of bytes still reveals distinct patterns. For example, the byte sequence from positions 6 to 60 is unique to the Samsung camera compared to the Amazon Echo and LiFX lightbulb.

The key takeaway is that patterns emerge in individual bytes (when entropy is low) and sequences of byte values. Fig. 4 clearly illustrates these distinct flow patterns, which sophisticated models like neural networks can leverage to differentiate traffic of various device types.

Another important pattern observed in the network traffic of IoT devices is their use of transport-layer services (*e.g.*, TCP/80, UDP/53), which they may access and/or expose. In addition, the transitions between these services can reveal unique behavioral fingerprints for IoT devices. In Fig. 5, we illustrate the transitions between network services for three representative devices during the first week of our dataset, representing an approximate state machine for the network

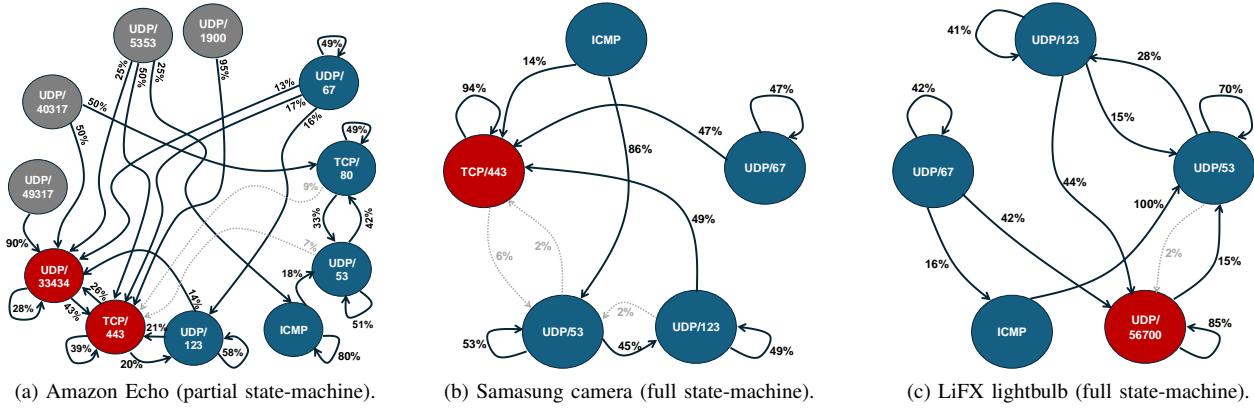


Fig. 5. An approximate state-machine of services observed in the network traffic of three representative devices, based on a week's worth of data.

behavior of each device. This figure is intended as an intuitive visualization rather than a formal model, motivating our subsequent construction of input sequences (§III-C) for the classifier model. For clarity, each service is treated as a state, and we estimate transition probabilities between consecutive services as:

$$\delta(s_i, s_j) = \frac{\text{count}(s_i \rightarrow s_j)}{\sum_{k \in S} \text{count}(s_i \rightarrow s_k)}$$

where $\text{count}(s_i \rightarrow s_j)$ denotes the number of times service s_j follows s_i . These probabilities capture the relative likelihood of transitions, but for visualization only the most frequent transitions are displayed. Low-probability transitions and rarely observed services are omitted, meaning that the outgoing probabilities shown in Fig. 5a may not sum to one, resulting in a partial representation. This approach captures the dominant behavioral patterns of the device, omitting rare transitions for visual clarity. That said, our input traffic representation later in §III-C will incorporate all observed sequences, including both less-frequent and most-frequent patterns.

To construct the state machine for a specific device, we first sort all flows associated with that device by arrival time and analyzed consecutive flows to calculate the transition rates. A transition from UDP/53 to TCP/80, as shown on the right of Fig. 5a, indicates that the Amazon Echo has a 42% probability of establishing a TCP/80 flow following a UDP/53 flow (the DNS protocol), thereby highlighting a specific sequence in flow arrival times. Transitions with relatively lower probabilities are highlighted using dotted gray arrows. Note that probabilities have been rounded to the nearest integer; as a result, transitions with very small probabilities (say, less than 0.5%) are not displayed in the diagrams. Gray circles represent services considered insignificant, as they rarely receive in-links from other states. Red circles, in contrast, highlight significant services that attract substantial attention, with many states transitioning to them. Blue circles indicate ordinary states.

As expected, the Amazon Echo is a complex device exhibiting sophisticated behaviors, frequently transitioning between a diverse set of services, as shown in Fig. 5a. For clarity, we have omitted transitions with probabilities less than 5%. It can be seen that the standard service TCP/443 and the non-standard service UDP/33434 are significant, highlighted in red,

receiving high probability in-links from several other services. It can be seen that four insignificant services (UDP/1900, UDP/5353, UDP/40317, UDP/49317) are highlighted in gray.

On the other hand, the Samsung camera and LiFX lightbulb exhibit simpler behavioral patterns. TCP/443 is the central service for the Samsung camera in its network activity, and the other four services transition to it with notable probabilities, as shown in Fig. 5b. Fig. 5c highlights that a proprietary service like UDP/56700 is a key state in the behavior of the LiFX lightbulb. Almost half the time, the lightbulb connects to its cloud server on UDP/56700 after syncing its time using UDP/123 (the NTP protocol).

Each of our custom flows captures the service utilized in each snapshot of traffic. To analyze sequences among services, we implement a strategy that leverages a set of flow records for inference. For this purpose, we maintain a buffer that stores the most recent K flows per device on the network. This buffer is updated with the arrival of each new flow record, while the oldest record is removed. The updated buffer is then exported to the inference engine with each new flow record, ensuring timely updates. Note that periodic updates would be challenging, as the inter-arrival time between flows can vary significantly over time and across different device types. It is also important to note that K can be configured to best suit the composition of network devices, depending on the available computing resources. In Section IV, we will set $K = 5$ for neural network models.

C. Feature Engineering

Neural network models require the input data to be of fixed size. Additionally, it is important to scale all input features in a consistent range, such as -1 to 1. If the input features are on different scales, it can slow down the learning process and hinder the model's ability to focus on the most important features. In this subsection, we refine our raw custom flow records to prepare them for downstream inference tasks.

Size of Input Features: The metadata fields, such as flow timestamp, remote IP, protocol, device port, remote port, total packet count and total byte count, are fixed in size and require minimal memory consumption. As a result, no transformation is needed for these features. However, as previously discussed

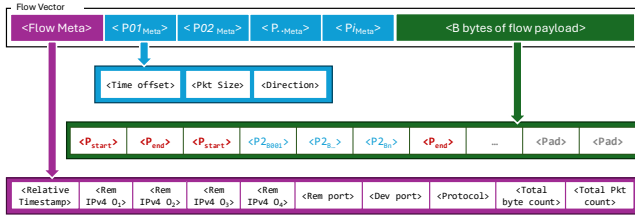


Fig. 6. Structure of a flow record prepared for inference.

(in §III-B and Fig. 2), we segment the remote IP into four octets (0-255) to reduce its complexity during classification. Therefore, we obtain ten input features from the flow metadata. The packet metadata fields offer three features: time offset, packet size, and packet direction for the first ten packets in each custom flow, resulting in 30 additional features.

The payload section of a custom flow contains substantially more data than the metadata sections. In Section III, we have explained that our custom flows are designed to opportunistically maximize the capture of behavioral information from the initial stages of communications within each minute of their lifetime, incorporating up to 1,000 bytes from each of the first ten packets. This results in 10,000 features, with each payload byte represented as an individual feature. However, our analysis reveal that 61% of packets in our dataset contain no payload (zero bytes), and the average payload size for non-empty packets is only 188.5 bytes.

Without appropriate data pruning or compaction, custom flows would include a substantial number of NaN values, which could mislead pattern recognition algorithms. Given the considerable variability in payload sizes, a fixed-size representation per packet would be suboptimal for our custom flows. To address this challenge, we implement three key changes: (a) separating metadata (time offset, packet size, and direction) from the payload data of individual packets in a flow, (b) introducing delimiters that mark the start and end of each packet’s payload (either empty or non-empty), and (c) arranging the packet metadata sequentially, followed by the corresponding sequence of packet payloads. This approach effectively manages the potential insertion of NaN values within the payload section. Additionally, we limit the total payload section of each flow to B bytes—where $B = 3,000$ in our experiment—ensuring the capture of at least three packets, which are more likely to carry valuable patterns [39]. If the total payload size, including the delimiters, is less than B , we pad the remaining bytes with a negative value of -255 to avoid any value overlapping with the actual payload bytes. This approach compacts the custom flow size by nearly 70% compared to using a fixed-size representation for each packet’s payload. The delimiters can be selected as any two distinct negative values within the range of -1 to -254 to avoid overlapping with the actual payload bytes or the padding byte. Thus, we have chosen -4 and -8 as delimiters for the start and end of the payload (<Pstart> and <Pend>) for each packet in our custom flow. Fig. 6 shows the structure of a flow record prepared for inference. This structure is a modified version of the original custom flow previously shown in Fig. 1. A custom flow record provides 3,040 in features, encompassing

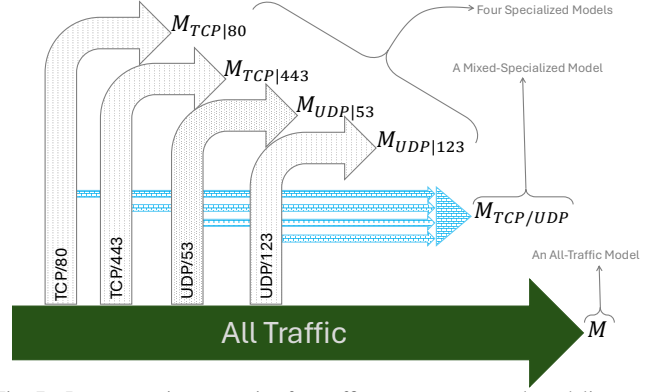


Fig. 7. Representative strategies for traffic measurement and modeling.

flow metadata, packet metadata, and payload sections.

Scaling Input Features: We normalize the values of various features to ensure that learning models can effectively focus on the complex patterns present within them. Beginning with the timestamp feature, we convert this absolute value into a relative timestamp (relative to its preceding flow in the buffer). We then divide it by $24 \times 60 \times 60 \times 10^6$ (24 hours in microseconds), scaling the value to fall within the range of 0 to 1. In our dataset, the flows of individual IoT devices are typically spaced less than 24 hours apart under normal circumstances. In rare cases where the relative timestamp exceeds a day, the scaled timestamp can be greater than 1.

We scale the remote IP octets and port numbers by dividing them by their maximum values (*i.e.*, 255 for IP octets and 65,535 for port numbers). For non-TCP/UDP flows where a port number is missing, we fill it with -1 to indicate the non-existent value. The total byte count and packet count features of a flow can vary widely in value. These features often exhibit a skewed distribution with a long tail in IoT traffic, where many small values coexist with a few very large ones. To address this, we apply a log transformation² to compress the range of values. Subsequently, we perform z-score normalization using the mean and standard deviation derived from the training data.

Packet time offsets are normalized to 60 seconds, reflecting the maximum flow expiration time. The packet size is scaled according to the most common MTU size of 1,500 bytes, optimizing model performance for typical network setups. Packet directions are not further scaled, as they only take values of 0 or 1. If a flow contains fewer packets than expected, we set the sections corresponding to the time offsets, sizes, and directions of the missing packets to -1 to indicate padding. The order of the arrival of the packet is preserved within each feature array to capture the sequence of the arrival of the packet. Finally, for the byte values in the payload section, we scale each byte by dividing by 255, resulting in values that range between -1 and $+1$. Negative values are the result of special characters in our custom flow representation: padding (NaN) is represented by -1 , while the delimiters <Pstart> and <Pend> are represented by -4 and -8 , respectively.

² $\log(x+1)$, where x represents a counting feature, is used to handle cases where the count might be zero by shifting the input to $(x+1)$.

After structuring and scaling the flow vectors, we create flow instances, comprising five successive flows per device identity, as discussed in Section III-B, to train the models in the following section. A flow instance is organized as a matrix, where rows represent individual flows and columns correspond to 3,040 features (metadata and payload).

IV. MODELING NETWORK BEHAVIORS

Network operators employ various strategies to model the traffic of their network assets and make inferences. Real-time and continuous inferences are highly desirable for cyber analysts to monitor their asset registry and ensure healthy network activities. However, achieving real-time inferences, particularly at scale, can present challenges. For instance, in addition to prediction quality (*e.g.*, accuracy of classification), factors such as computing resources, prediction frequency, coverage of asset types, and the complexity of traffic measurement play significant roles in determining the optimal inference strategies. Our objective in this paper is to predict the make and model of IoT devices connected to the network, focusing on fine-grained output classes such as Insteon camera, TP-link camera, Dropcam, HP printer, or LiFX lightbulb. Alternatively, one could pursue a coarser-grained classification, where each class represents a cluster of behaviors [44], such as device functionalities (*e.g.*, cameras, printers, lightbulbs).

In this section, we examine three specific and representative strategies. We begin with a lightweight, modular, and extensible approach, where a subset of traffic corresponding to certain network services (identified by transport-layer port numbers) is modeled by a classifier that becomes “specialized” in recognizing patterns of traffic flows for that particular network service (*e.g.*, a specialized classifier for TCP/80 traffic). These services may roughly correspond to common protocols, such as TCP/80 for HTTP. However, there is no assumption or requirement for a specific mapping of protocol to port numbers, nor for knowledge of the protocols themselves. The upper part of Fig. 7 illustrates how four representative services are filtered from the entire traffic and fed into four specialized models, depicted by gray dotted patterned arrows. Next, we explore an approach where traffic from selected services is combined, leading to a model trained on characteristics of these specific TCP and UDP services—this is illustrated by blue bricked pattern arrows in the middle part of Fig. 7. Finally, we consider a traditional approach where all traffic is modeled by a single, comprehensive model, depicted at the bottom of Fig. 7 by a solid green arrow.

As previously mentioned in Section II, among various learning algorithms and techniques, we choose to focus on neural networks for this work. Neural networks have demonstrated their superiority across various domains, including computer vision, speech processing, and natural language processing, by *automatically learning features* that would require domain expert knowledge and might be tedious and/or incomplete if engineered manually. Furthermore, neural networks tend to produce better models when large and diverse datasets are available. Importantly, neural networks can be *systematically adapted*, augmented, or pruned on demand with small amounts

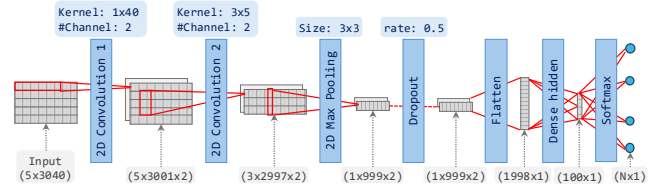


Fig. 8. Architecture of our neural network model.

of new data specific to certain classes or environments, enhancing their practical appeal.

We employ convolutional neural networks (CNNs) to model IoT network behaviors. CNNs are particularly effective for processing data that has a grid-like structure, such as images or our custom flows. In addition to their predictive performance and computational efficiency [45], CNNs offer generalizability due to their ability to learn hierarchical representations of input data. This enables them to capture complex patterns, such as the behaviors of Amazon Echo, at both local (within a flow) and global (across sequences of flows) levels. Furthermore, their translation invariance makes CNNs particularly effective at detecting identifiable patterns, regardless of their location within the input data, especially in the payload section of our custom flows.

A. Model Training

To better align our assessment with real-world scenarios, we organize the training and testing sets chronologically. We utilize training and validation datasets for model training. The algorithm is trained on the training set and fine-tuned using the unseen validation set. We select instances from the first seven days (*i.e.*, 23-Sep-2016 to 29-Sep-2016, which contain 786,000 instances) for training and the following three days (*i.e.*, 30-Sep-2016 to 02-Oct-2016, containing 346,000 instances) for validation. The remaining 4.8 million instances, from 03-Oct-2016 to 22-Nov-2016, are reserved for testing. When training models, we exclude device classes that contribute fewer than 50 instances to their respective modeling strategies. Additionally, we limit the number of training instances per class to manage class biases. For the first strategy (specialized models), we set an upper limit of 10,000 instances per class. For the other two strategies, we double this threshold to 20,000 instances per class to account for the larger number and diversity of instances. If a class contributes more than these thresholds, we randomly select instances meeting the configured limits per device class. To enhance generalizability and prevent overfitting to a particular order of flow instances, we shuffle the training data instances before presenting them to the algorithm.

For this study, we design a compact CNN model architecture capable of training from the custom flows. We utilize a 7-layer [46] 2D CNN model comprising two 2D convolutional layers, a dropout layer, a max pooling layer, a flattening layer, a dense layer, and a softmax layer. Fig. 8 shows the architecture of our neural network. The input to the network is defined as a tensor of shape 5×3040 , representing a moving window of five flows, each with 40 metadata features and 3000 payload bytes. The first convolutional layer processes patterns

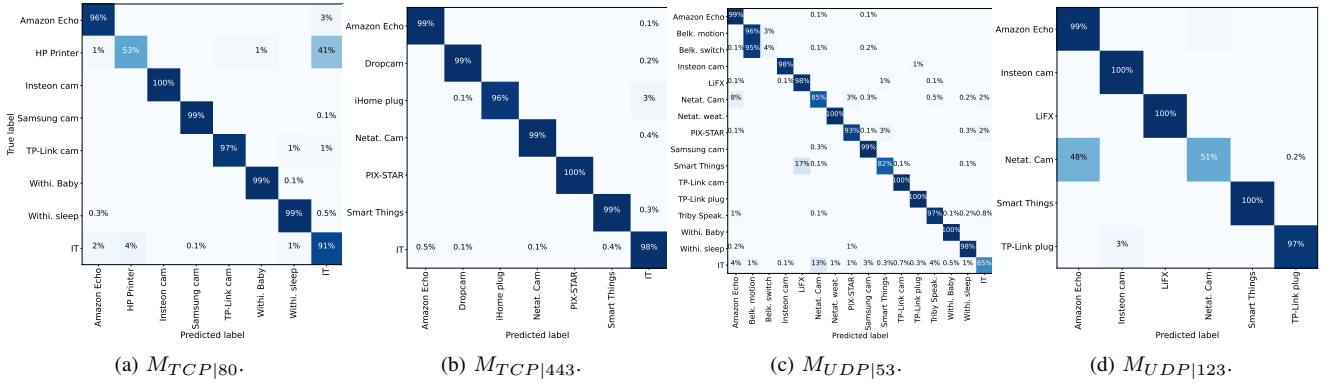


Fig. 9. Prediction quality of four specialized neural networks, each trained on traffic of a specific network service—(a) TCP/80, (b) TCP/443, (c) UDP/53, and (d) UDP/123—when tested on corresponding instances.

within individual flows. To encompass all 40 metadata features in a single convolution, we set the kernel size to 1×40 and the strides to $(1, 1)$. The second convolutional layer detects patterns across the flows, using a kernel size of 3×5 and strides of $(1, 1)$. To avoid overfitting, we include a dropout layer with a rate of 0.5. A max-pooling layer further reduces the spatial dimensions. The model proceeds with a flattening layer, which converts the 2D matrices into a 1D vector. This is followed by a fully connected dense layer consisting of 100 neurons with ReLU activation. The final (output) layer employs a dense layer with softmax activation to produce class probabilities. The output is a probability vector with a dimension equal to the number of unique device types (classes), obtained via a softmax layer, where each element corresponds to the predicted likelihood of a class.

The model is trained using sparse categorical cross-entropy loss with the Adam optimizer [47]. To assess performance, we employ the Macro F1 score, as it provides a balanced evaluation of precision and recall across all classes and is therefore more suitable than simple accuracy for our multi-class setting. We use the early stopping technique to halt training when the validation loss does not improve for a specified number of epochs, in our case, five, to prevent overfitting. This approach ensures optimal performance on unseen data by monitoring the validation loss with a minimum improvement threshold of 0.001 and restoring the model weights from the best epoch.

Our dataset is large, and loading it entirely for training or testing demands substantial memory. To address this, we have implemented a lazy loading mechanism using TensorFlow Data APIs [48], where instances are streamed to the model progressively.

B. Model Performance

We use various metrics to compare the performance of different strategies. We primarily focus on prediction quality (*i.e.*, model accuracy) and later discuss other computing and operational metrics.

Specialized Models: For our first strategy, we focus on four representative network services: two for TCP and two for UDP. For TCP, we select TCP/80 and TCP/443, which likely represent HTTP (plaintext) and HTTPS (encrypted), commonly used for client-server communications. For UDP, we focus on UDP/53, which probably represents DNS, and

UDP/123, which likely represents NTP. These UDP protocols are prevalent in IoT device traffic and are used to resolve cloud server addresses (DNS) and obtain accurate time stamps (NTP) for measurements and actions. These four services collectively contribute to 30% of our traffic dataset. We, therefore, train four specialized models: $M_{TCP|80}$, $M_{TCP|443}$, $M_{UDP|53}$, and $M_{UDP|123}$. It is important to note that not all device types in our dataset necessarily utilize all four of these protocols in their network activities. For example, only eight device types used TCP/80, while 16 used UDP/53.

Fig. 9 shows four confusion matrices, each corresponding to a specialized model. Ideally, we expect a diagonal matrix from individual models. Among the four specialized models, $M_{TCP|443}$ outperforms the other by consistently achieving high accuracy across all classes it predicts, shown in Fig. 9b. However, in the other three models, we observe that at least one class fails to achieve high accuracy (*i.e.*, greater than 90%), highlighted by light blue cells in the corresponding confusion matrices.

Quantitatively, based on the macro-averaged F_1 score, the prediction quality of these models is as follows: $M_{TCP|443}$ leads with the score of 0.99 followed by $M_{UDP|123}$ with 0.93. The other two models, $M_{TCP|80}$ and $M_{UDP|53}$, follow with scores 0.87 and 0.80, respectively. Note that while $M_{TCP|443}$ performs relatively well, it covers seven classes of IoT devices in our dataset, which is fewer than $M_{TCP|80}$ (8 classes) and $M_{UDP|53}$ (16 classes).

Analyzing more closely, we observe that the uniqueness of behavioral patterns per class varies across these four representative traffic services. For example, all four models capture Amazon Echo’s network behavior well. Also, Withings devices (baby monitor and sleep sensor) are classified relatively well by $M_{TCP|80}$ and $M_{UDP|53}$; these two devices do not utilize TCP/443 and UDP/123 for communication. However, other classes are unable to achieve such consistent performance across the four models. For example, a device like the Netatmo camera can be perfectly distinguished by analyzing its TCP/443 traffic. However, this distinction is overshadowed by the Amazon Echo when analyzing UDP/123 traffic as shown in Fig. 9d. It can be seen in Fig. 9c that the Belkin motion and the Belkin switch, both from the same manufacturer, exhibit similar patterns in UDP/53 traffic. As a result, most UDP/53 flows are predicted to be Belkin motion,

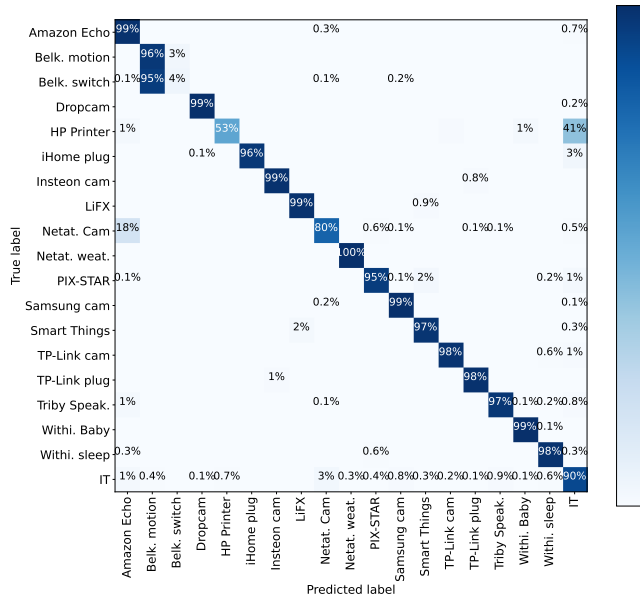


Fig. 10. Prediction quality when employing the union of four neural networks: $\{M_{TCP|80}, M_{TCP|443}, M_{UDP|53}, M_{UDP|123}\}$.

reflecting its dominance in the training data. Similarly, the prediction quality of $M_{UDP|53}$ for the IT class is not very high (65%), since IT devices in our testbed often accessed the same domain names as IoT devices. In contrast, the IT class is well distinguished by $M_{TCP|443}$, where there are clear patterns with minimal overlap with other classes.

To better assess the overall performance of this inference strategy, we construct a confusion matrix using the union of input instances and their corresponding predictions from the four specialized models: $M_{TCP|80}, M_{TCP|443}, M_{UDP|53}$, and $M_{UDP|123}$. It is important to note that the four subsets of input instances are non-overlapping, and their corresponding prediction subsets are independent. Therefore, instead of employing standard techniques, such as majority voting, for ensemble models, we utilize the union of predictions. The resulting matrix is shown in Fig. 10. This set of combined models achieves a macro-averaged F1 score of 0.82, covering 19 classes of devices. A key observation is that while some classes experience lower accuracy in individual specialized models, due to known overlaps and/or rare usage of the corresponding protocols, overall performance improves when considering the union of predictions. For example, the Netatmo camera receives an accuracy of only 51% from $M_{UDP|123}$, but with the integration (union) of models, 80% of its traffic flows are correctly classified. This is because NTP (UDP/123) constitutes a relatively small proportion of total traffic for this specific class, and therefore, it has less impact on overall inference performance.

We quantify the models' workload as a proxy for computing resource demand by measuring the average number of invocations each model receives per hour. A model is invoked whenever a corresponding flow instance arrives; hence, workload naturally varies across networks depending on the composition of connected devices and their activity patterns. In our dataset, the models $M_{TCP|80}, M_{TCP|443}$,

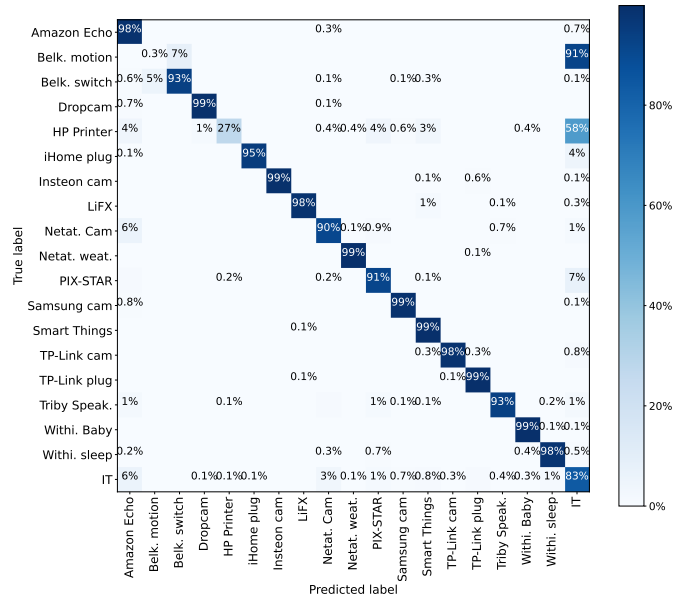


Fig. 11. Prediction quality when employing a neural network ($M_{TCP/UDP}$) trained on the mixed traffic of four services: $\{TCP/80, TCP/443, UDP/53, UDP/123\}$.

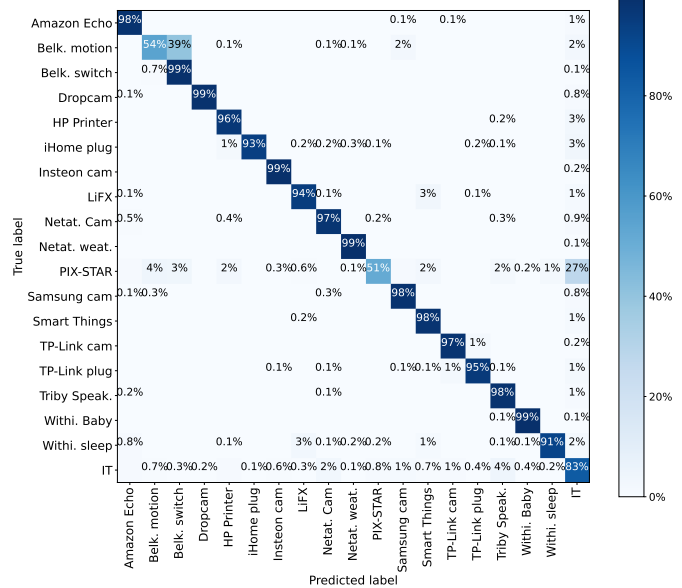


Fig. 12. Prediction quality when employing a neural network (M) trained on all traffic.

$M_{TCP|53}$, and $M_{TCP|123}$ process approximately 150, 430, 50, and 150 instances per hour, respectively. Among these specialized models, $M_{TCP|443}$ experiences the highest average load, handling 430 instances per hour. We therefore take this worst-case workload and define it as one unit of workload (U). **Mixed-Specialized Model:** Instead of creating separate models for each service, our second inference strategy employs a single model trained on the combined traffic flows of four services: $\{TCP/443, TCP/80, UDP/53, UDP/123\}$. The confusion matrix illustrating the performance of this mixed model is presented in Fig. 11. This approach takes advantage of the potential to identify patterns in the sequence of services used by each class that may be missed by specialized models. However, this strategy may present challenges for classes that do not utilize all four common services (or rely only on

TABLE IV
SUMMARY OF INFERENCE APPROACHES WHEN APPLIED TO OUR DATASET.

	$M_{TCP 80}$	$M_{TCP 443}$	$M_{UDP 53}$	$M_{UDP 123}$	Specialized	Mixed-Specialized	All-Traffic
F1-score (macro)	0.87	0.99	0.80	0.93	0.82	0.80	0.90
Classes covered	8	7	16	6	19	19	21
Workload hourly	150 \rightarrow 0.35U	430 \rightarrow U	50 \rightarrow 0.11U	150 \rightarrow 0.35U	430 \rightarrow U	1230 \rightarrow 3U	4140 \rightarrow 10U
Traffic selection	Yes	Yes	Yes	Yes	Yes	Yes	No

one). As a result, their unique behavioral patterns might be overshadowed by those of other active classes that utilize a wider range of services.

Compared to the specialized models, the overall prediction performance declines, achieving a macro-averaged F1 score of 0.80 while maintaining coverage across 19 device classes. It should be noted that the impact varies between different classes. For example, the accuracy for the PIX-STAR photo class decreases from 95% to 91%. In contrast, the accuracy for the Netatmo cam class improves significantly, increasing from 80% to 90% compared to our first approach. This improvement is possibly because the Netatmo camera benefits more from the patterns in the sequence of services it utilized. In terms of workload, our mixed model handles an average of 1,230 instances per hour, about three times³ the baseline capacity in our first approach.

All-Traffic Model: Finally, let us consider a model trained on all traffic flows generated by IoT devices. This model aims to capture a superset of behaviors exhibited by various device classes in network traffic. However, its performance may be affected by common, yet not necessarily distinct, services (*e.g.*, mDNS, SSDP, and NetBIOS protocols) that are shared across multiple device types.

Fig. 11 shows the confusion matrix for this model. The overall performance is higher than the previous two strategies, achieving an F1 score of 0.90 and covering 21 device classes. However, a class like the PIX-STAR photo frame shows reduced accuracy compared to previous approaches, likely due to its frequent use of the common discovery protocol (*i.e.*, NetBIOS). Lastly, this model handles an average workload of 4,140 instances per hour, approximately ten times the baseline strategy (*i.e.*, 10U, where $U = 430$ denotes the average workload of our specialized models.).

Summary: We have examined three approaches to infer from network traffic, each with pros and cons depending on the environment and objectives. Table IV summarizes their merit quantified by four metrics. Using specialized models is a relatively light solution for network operators who face constraints on how much traffic they can analyze, whether due to privacy concerns or limited computational resources. However, this approach requires careful selection of network traffic (*e.g.*, using programmable switches [49]) to ensure the appropriate subsets are fed into the respective models. The prediction quality and coverage highly depend on the protocols operators choose to focus on—in industrial and building management systems; certain protocols, such as Modbus or BACnet, may lead to an effective inference. Alternatively, a mixed model ap-

³With an average workload of 430 instances, represented by U in our specialized approach, the calculation $1,230/430$ yields approximately $3U$.

TABLE V
SUMMARY OF MODEL SIZES.

Model	Size on Disk (MB)	Memory After Loading (MB)
ET-BERT	628	694
NetMamba	22	540
Our Classifier	2.4	6

proach may be suitable when operators wish to avoid analyzing all traffic but still want to leverage sequence patterns across network services used by connected devices. Finally, the all-traffic model, though computationally expensive, is effective for classifying a wide range of heterogeneous devices when pre-selecting target services is undesirable or impractical.

C. Performance Benchmarking: Inference and Computing

We now benchmark our classifier (*i.e.*, model M trained on all traffic) against two recent deep learning neural network-based models, designed for network traffic representation and classification: ET-BERT [5] and NetMamba [7]. Both are related to our work but differ in model architecture. ET-BERT employs a heavy multi-stage preprocessing pipeline, converting raw traffic bytes into hex strings, applying vocabulary-based tokenization, and then feeding sequences into a BERT-like encoder. In contrast, NetMamba uses a Mamba-based state space model, which is more memory efficient than transformers, but Mamba blocks demand GPU, making NetMamba unsuitable for CPU-only edge devices.

We start our benchmarking by first evaluating the three models (our classifier, NetMamba, and ET-BERT) on a GPU server, and then examining their performance on edge devices.

Model Sizes: We profile the size of the models on a GPU server in two ways: (a) their size on disk (model at rest) and (b) their memory footprint immediately after loading into memory (before inference). Table V summarizes our measurements. As expected, the memory footprint after loading is larger than the disk size for all three models. A minor observation is that our classifier shows $2.5\times$ increase (from 2.4 MB to 6 MB), compared to only 3% for ET-BERT and 19% for NetMamba. More importantly, our classifier remains one order of magnitude smaller than NetMamba and two orders of magnitude smaller than ET-BERT. This drastic reduction in size makes our classifier particularly well-suited for embedded deployment in resource-constrained edge devices.

Inference Quality: For this metric, we examine the macro-F1 score of the three models when applied to our dataset collected from the traffic of IoT devices. Again, on the GPU server, we fine-tune the ET-BERT and NetMamba models with our dataset. During the testing phase, our classifier achieves a macro-F1 score of 0.90, outperforming ET-BERT (0.72) and NetMamba (0.74).

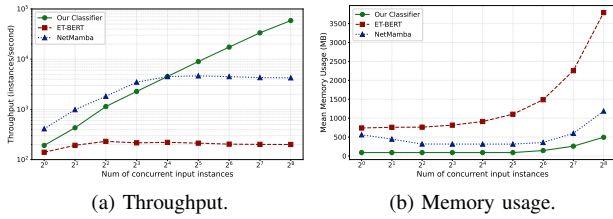


Fig. 13. Runtime footprint of our classifier vs NetMamba vs ET-BERT as a function of the number of concurrent input instances on a GPU server: (a) throughput (number of inferences per second), and (b) memory usage (MB).

By analyzing their confusion matrices, we found that NetMamba and ET-BERT demonstrate high-quality predictions in certain classes (*e.g.*, ET-BERT in Withings Sleep with 98% and NetMamba in LiFX with 99%). However, their performance is relatively poor in some other classes. For example, NetMamba struggles to distinguish between devices from the same vendor, such as Belkin motion and Belkin switch, achieving only 5% accuracy. ET-BERT, on the other hand, shows a strong performance in classifying devices with encrypted traffic (*e.g.*, IT at 98% and iHome plug at 95%), but performs poorly on devices with less encrypted traffic, such as the TP-Link plug (21%) and Samsung cam (65%). In contrast, our model achieves high accuracy relatively consistently across most classes, with fewer significant drops, resulting in a more balanced performance profile. This stability across diverse device types accounts for its higher macro F1-score than NetMamba and ET-BERT. It is important to note that the definition of a flow differs between our model and NetMamba and ET-BERT: we split flows exceeding 60 seconds into multiple flows, whereas NetMamba and ET-BERT do not enforce such timeouts and treat the entire flow as a single flow.

Runtime Footprint: For this experiment on the GPU-based server, we restrict GPU usage to one NVIDIA L4 card (24 GB memory), out of the four installed on the server. We evaluate two metrics: (a) model throughput, measured as the number of instances inferred per second, and (b) average memory usage (MB) during inference. Note that a higher throughput indicates faster inference. The experiment is repeated with the number of concurrent input instances ranging from 1 to 256, doubling at each step. Fig. 13a shows that the throughput of our classifier (solid green line) monotonically increases with the number of concurrent input instances. In contrast, ET-BERT (dashed red line) and NetMamba (dotted blue line) show a modest increase initially and then stabilize midway. At lower workloads (fewer than 16 input instances), NetMamba achieves higher throughput than our classifier. However, beyond this point, our classifier overtakes it and steadily widens the gap, ultimately achieving a throughput of 58000 (that is, 17 microseconds per inference) at the maximum workload (256 concurrent inputs), an order of magnitude higher than NetMamba. While this confirms that our classifier is highly efficient under heavy workloads, ET-BERT remains significantly slower, reaching nearly three orders of magnitude lower throughput at the maximum workload.

In terms of memory usage shown in Fig. 13b, our classifier consistently maintains a smaller footprint across all workloads.

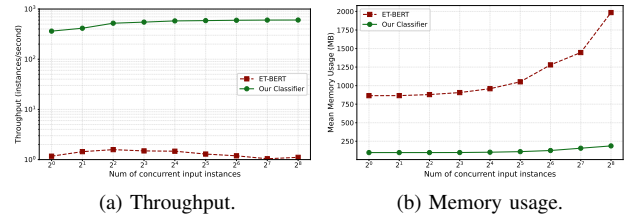


Fig. 14. Runtime footprint of our classifier vs ET-BERT as a function of the number of concurrent input instances on a CPU edge device: (a) throughput (number of inferences per second), and (b) memory usage (MB).

With a single input instance, it requires only 16% of NetMamba’s memory, increasing to 40% at 256 input instances. Compared to ET-BERT, our classifier consumes just one-tenth of its memory across all workloads. We also note that ET-BERT’s memory usage increases sharply under medium to high workloads. At the same time, NetMamba exhibits a slight decline at lower workloads, stabilizes at intermediate levels, and then increases modestly at higher workloads.

Deployability on Edge Devices: Finally, we evaluate the feasibility of real-time inference on a resource-constrained edge device: Raspberry Pi 5 with quad-core ARM Cortex-A76 @ 2.4 GHz and 16 GB RAM. Note that Raspberry Pi devices provide CPU-only computing resources, without GPU accelerators or NPU add-ons.

On the Raspberry Pi 5, we could only run ET-BERT and our classifier, and NetMamba was excluded from our benchmarking study due to its dependency on GPU resources. Fig. 14 presents the runtime footprint of our classifier versus ET-BERT on Raspberry Pi 5 (a CPU-based edge device). Starting with throughput, ET-BERT achieves its best rate of 1.8 instances per second at 4 concurrent inputs. In contrast, our classifier maintains a minimum of 350 instances per second throughout and scales to nearly 600 instances per second (meaning 1.7 ms per inference) at 256 inputs. This translates to more than 300× faster than ET-BERT. Moreover, consistent with the GPU benchmarking results, ET-BERT shows a nearly order of magnitude higher memory footprint across all workloads, as shown in Fig. 14b.

These results highlight an important distinction. While large models such as ET-BERT and NetMamba achieve reasonable accuracy, their computational and memory demands make them impractical for deployment on CPU-only edge devices. In contrast, our contribution is a lightweight CNN-based classifier that not only delivers higher accuracy but also enables real-time inference on resource-constrained hardware platforms. This demonstrates that, when combined with appropriately formatted input flows, a carefully structured CNN can achieve competitive, and in some cases superior, performance compared to much larger architectures. In the next subsection, we analyze the effect of CNN depth on inference quality, and in Section V, we provide interpretability analysis to show how the model leverages the input flow structure to reach high accuracy.

D. Ablation

We conduct an ablation study to evaluate the contribution of each layer in our CNN-based classifier for IoT traffic. The baseline architecture consists of two convolutional layers

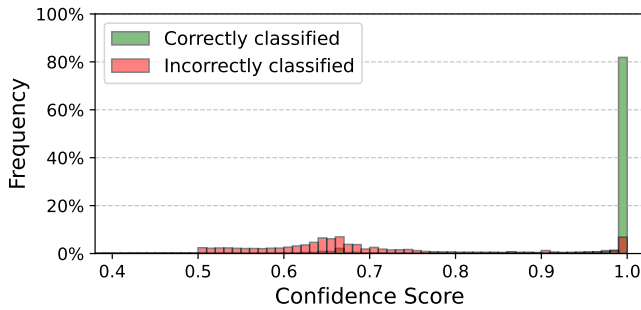


Fig. 15. Distribution of confidence score for correctly and incorrectly classified instances

TABLE VI
ABLATION STUDY RESULTS.

Model Variant	Δ Params	Epochs (<i>early-stop</i>)	Δ Train Time (<i>early-stop</i>)	Δ F1-score (<i>early-stop</i>)	Δ F1-score (<i>fixed-epoch</i>)
Baseline	—	8	—	—	—
— Conv L1	+1%	10	+25%	0%	0%
— Conv L2	0%	13	+55%	−4%	−3%
— Conv L1&L2	+1,416%	9	+5%	−40%	−44%
— MaxPool	+791%	8	+1%	−2%	+1%
— Dropout	0%	8	0%	0%	0%
— Hidden Dense	−81%	15	+87%	+1%	0%

(Conv L1 with a 1×40 kernel and Conv L2 with a 3×5 kernel), a MaxPool layer, a Dropout layer, a fully connected hidden layer, and a softmax layer. We systematically remove individual layers while maintaining the remaining architecture intact. Table VI summarizes the results of our experiments.

Each row corresponds to an experiment in which one (or two) layers are removed from the baseline. For example, “— Conv L1” indicates that the first convolutional layer of the model is removed. The first column reports the change (%) in the number of model parameters relative to the baseline model. Because convolutional and pooling layers downsample the input and leverage weight sharing, they keep the parameter count relatively low. When these layers are removed, subsequent layers must fully connect to a larger input space, causing a substantial increase in parameters.

To assess inference quality, we first employ our baseline training strategy (§IV-A), that is, allowing early stopping. It can be observed that with early stopping, the ablated models require more epochs (time) to converge than the baseline model. The most pronounced effects are observed by removing the Hidden Dense layer (requiring seven additional epochs, with 87% more training time) and Conv L2 (requiring five additional epochs, with 55% more training time). Considering the quality of the inference, the convolutional layers are found to be the most critical according to the Δ F1-score. Removing both convolutional layers leads to a substantial drop (40%) in macro-F1. Removing Conv L2 alone causes a modest decrease (4%). In contrast, removing MaxPool, Dropout, or the fully connected layer does not result in significant degradation. These findings highlight the importance of the convolutional layers, particularly Conv L2, in capturing inter-flow features

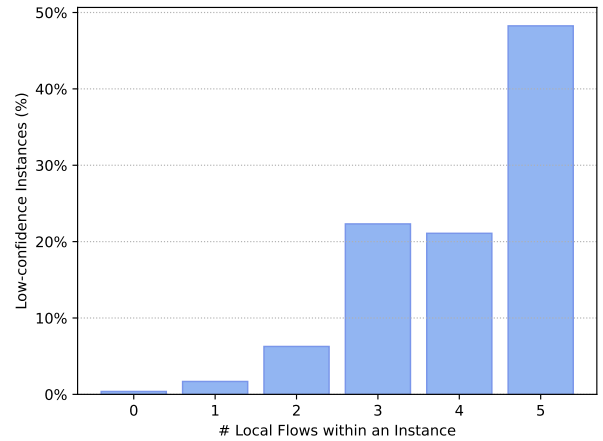


Fig. 16. Contribution of local flows to predictions with low confidence scores.

that are essential for effective IoT traffic classification. Lastly, we repeat the experiments with a fixed number of 15 epochs, chosen as the maximum from early-stopping runs. The last column in Table VI provides similar insights, again on the impact of the convolutional layers.

V. INTERPRETING THE PREDICTIONS OF MODELS

This section aims to investigate model performance more deeply and determine whether and how their predictions can be interpreted, and thereby trusted. Our analysis will only focus on the third inference approach, the all-traffic model (M), which has demonstrated relatively better predictions on our dataset.

Confident Predictions: One may use the confidence score to assess the trustworthiness of a classifier’s output, typically reflecting the probability assigned to the predicted class. Fig. 15 illustrates the distribution of confidence scores assigned by our model to correct predictions (indicated by green bars) and incorrect predictions (indicated by red bars). We observe that our model gives an average confidence score of 0.95 for correct predictions, while incorrect predictions have a lower average score of 0.71.

Let us divide the scores into low and high confidence with the threshold of 0.8⁴. We found that only 7% of all predictions are less confident. On the other hand, it is important to note that predictions with higher scores are not always correct; in fact, 1.5% of them are incorrect. More than a third of incorrectly predicted instances, over 68000, receive a relatively high confidence score (greater than 0.8).

Analyzing instances classified with low confidence (less than 0.8) revealed that most contain local flows (device-to-device communications). Fig. 16 illustrates the contribution of local flows to predictions with low confidence scores. It can be seen that more than 90% of the instances that receive low confidence scores from the model contain at least three local flows (communications between two devices on the network). Recall from Section III-A that we employed bidirectional flows for inference, and hence each local flow is associated with

⁴Determining confidence thresholds globally [1] or on a per-class basis [12] is beyond the scope of this paper. Additionally, operators may choose to handle predictions with lower confidence scores by discarding them or initiating specific investigations focused on a particular device or class.

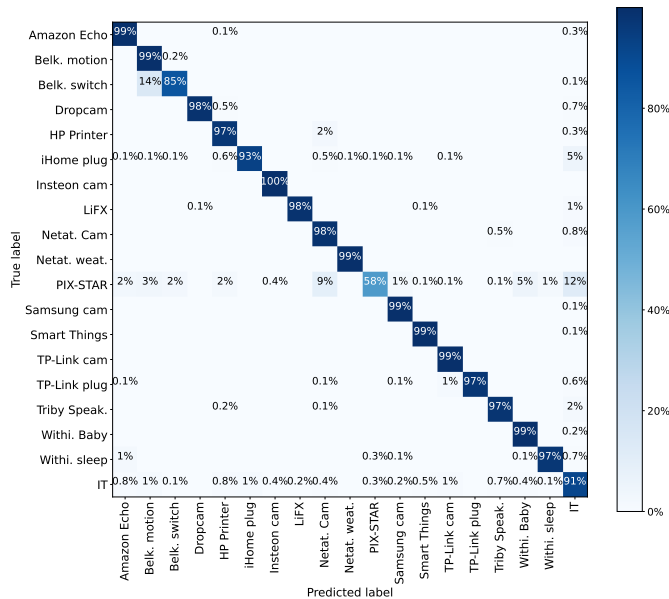


Fig. 17. Prediction quality when training the neural network (M^\uparrow) on unidirectional traffic.

both participating devices (sender and receiver). This means that our training dataset captures a local flow as two separate records, each assigned a distinct label, one for the sender and one for the receiver. Note that each record is constructed from the perspective of the corresponding endpoint, where incoming packets for one device are considered outgoing packets for the other, and vice versa. We hypothesize that local flows exhibit characteristics from both endpoints, making it more difficult for the model to confidently and accurately infer the corresponding class.

Refining Traffic Data: To reduce ambiguity, we refine our traffic data by incorporating only the traffic generated by each device within each flow. This means our flow records become unidirectional, capturing only upstream activities. By doing so, the incoming traffic from external (cloud) servers will be completely discarded from the analysis, and each local flow will be divided into two parts—each reflecting the behavioral patterns of a single device rather than two devices.

After refining our traffic data⁵ to include only unidirectional flows, we train a new model on the entire dataset, resulting in model M^\uparrow . Fig. 17 illustrates the prediction performance of this updated model. The macro-averaged F1 score shows a modest rise from 0.90 to 0.94. Notably, the accuracy of the IT class improved from 83% to 91%, while PIX-STAR photo instances saw an increase from 51% to 58%.

Furthermore, with M^\uparrow , the fraction of instances with low confidence scores decreased significantly, from 7% to 2% – a reduction by a factor of three. Additionally, the model refinement reduced the number of high-confidence misclassifications to one-fifth of the original count, decreasing from 68,000 to 14,000. Three classes, namely, the HP-Printer, IT, and Belkin switch, are the main contributors to the remaining high-confidence misclassifications. Our manual investigations

⁵Our dataset of bidirectional and unidirectional custom flows will be made publicly available [41].

revealed that those misclassified instances carry patterns deviating from those seen during the training period.

Verifying Predictions: Although refining the traffic data structure has moderately improved our classifier’s performance, network operators may still require a mechanism to better interpret or validate the predicted labels, particularly when the model exhibits high confidence. Implementing methods to interpret the model’s specific predictions could offer operators valuable insight and enhance trust in its decisions.

SHAP (SHapley Additive exPlanations) is a popular method for interpreting predictions by showing how much each feature contributes. It is based on Shapley values from game theory, which range between -1 and +1. SHAP works by testing how the prediction changes when a feature is added, removed, or altered in different combinations of features (called coalitions). To do this, it samples groups of features from the training data and compares the model output. For each prediction, SHAP then calculates how much each feature influenced the result. This allows SHAP to provide a global view of which features matter most overall and a local interpretability for individual predictions.

Due to the high computational cost of the SHAP method, our investigation focuses on sampled sets of randomly selected instances that receive high confidence scores: 1000 correctly predicted and 1000 incorrectly predicted.

Fig. 18 shows a subset of SHAP values for a correctly classified Amazon Echo instance. Recall that each instance consists of five custom flows, each with 40 metadata features and a 3,000-byte payload. For visualization purposes, we separate the metadata and divide the payload into 50-byte chunks (with rows separated by thick black lines). Each chunk contains five rows (indicated by thin gray lines) corresponding to the five custom flows. Also, this figure only shows the first 200 bytes of the payload section. Here, we have annotated each cell with red text, indicating the raw value of that feature from the original traffic data (*i.e.*, metadata, hexadecimal or ASCII characters equivalent to byte values). Note that the cell color represents the SHAP value of that specific cell (feature). Purple shades indicate a feature that positively contributed to the predicted label for this traffic instance. We manually verified that these dark purple cells correspond to low or medium entropy, as observed earlier in our analysis of raw custom flows in §III-B. However, the precise way the neural network model decodes these specific patterns remains unclear. As illustrated in Fig. 18, orange shades signify the negative contribution of a feature to the predicted label (class), reducing the probability that the model infers that class. White shades highlight neutral contributions. In this Amazon Echo example, we see that the model pays attention to patterns in unique byte values, packet direction, and packet offset times.

Fig. 19 shows the SHAP values of an incorrectly classified instance. In particular, elevated SHAP values are observed in regions denoted by padding bytes, which represent missing or empty features that were assigned a value of -1 to satisfy the fixed-size instance constraint.

To systematically employ the SHAP method, we normalize the Shapley values to a range between 0 and 1 using min-max scaling. This enables us to focus on areas that positively

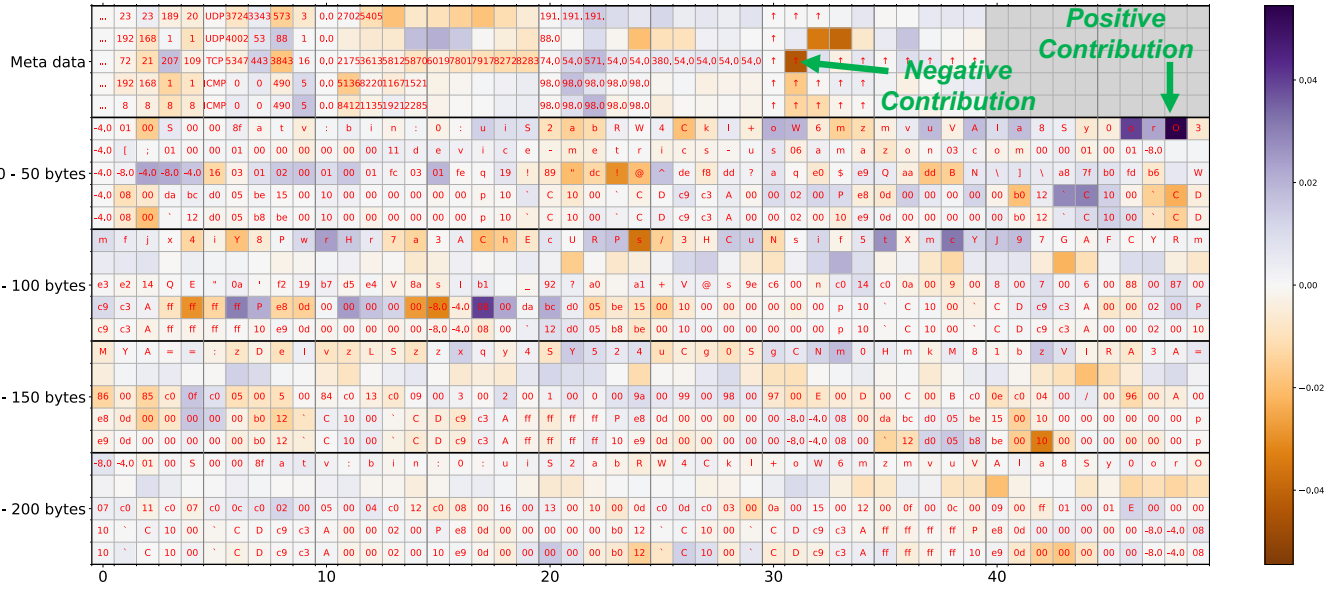


Fig. 18. Selected SHAP values for a correctly classified (unidirectional) instance of Amazon Echo—empty cells correspond to padding bytes.

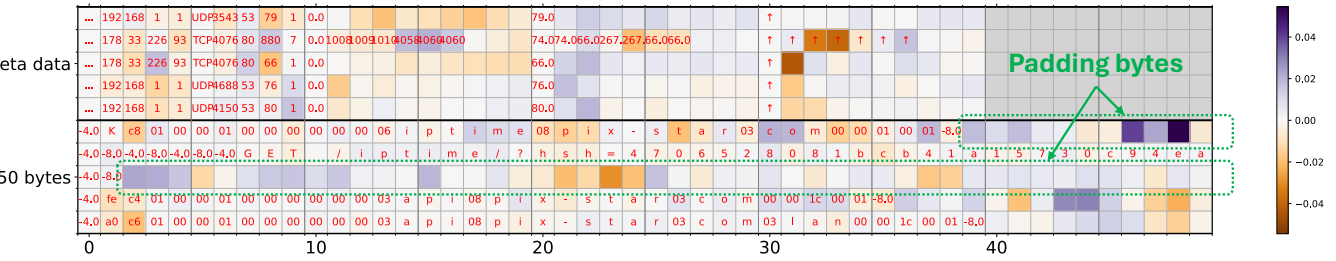


Fig. 19. Selected SHAP values for an incorrectly classified (unidirectional) instance of PixSTAR photo with great attention to padding bytes (empty cells).

contribute to the predicted label where the normalized Shapley values exceed 0.5. Next, we calculate the fraction of positive contributing features within the padding region of an instance (*i.e.*, “padding fraction”), which ideally should be nearly zero. We found that incorrect predictions exhibit larger padding fractions (an average of 0.09) than correct predictions (an average of 0.01). This suggests that the model pays more attention to padding areas when making incorrect predictions than correct ones. Applying a threshold of 0.01 to the padding fraction will help eliminate 12% of incorrect predictions with high confidence while impacting less than one percent of correct predictions. We emphasize that optimizing threshold tuning is beyond the scope of this paper. Various strategies can be used to determine the most effective thresholds for evaluating the reliability of predictions made by machine learning models. This subsection demonstrated a practical approach to using explanatory methods like SHAP to assist operators in effectively applying machine learning models for managing their networks and assets.

VI. CONCLUSION

Mapping and managing the growing number of IoT devices in enterprise and industrial networks requires efficient and generalizable traffic classification methods. This paper presented a lightweight deep learning approach for real-time inference of IoT behavior from structured network flows. We first introduced a configurable data structure that captures

flow metadata, timing, direction, and selected payload bytes in a fixed-size matrix, enabling real-time analysis without protocol-specific preprocessing. Using this, we released a large dataset of real IoT traffic representation and analyzed device behavior patterns. Next, we developed a compact CNN (200K parameters) capable of decoding intra-flow behavioral patterns and inter-flow sequences with high accuracy and low computational cost. We evaluated three inference strategies across four performance metrics, namely accuracy, coverage, computational workload, and traffic selection, highlighting trade-offs for practical use. Finally, we applied interpretability techniques using confidence scores and Shapley values to validate the trustworthiness of the predictions and help refine the model.

REFERENCES

- [1] A. Sivanathan, H. Habibi Gharakheili, F. Loi, A. Radford, C. Wijanayake, A. Vishwanath, and V. Sivaraman, “Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics,” *IEEE TMC*, vol. 18, no. 8, pp. 1745–1759, Aug 2018.
- [2] E. Lear, R. Droms, and D. Romascanu, “Manufacturer Usage Description Specification,” RFC 8520, Mar. 2019. [Online]. Available: <https://www.rfc-editor.org/info/rfc8520>
- [3] A. Hamza *et al.*, “Detecting Volumetric Attacks on LoT Devices via SDN-Based Monitoring of MUD Activity,” in *Proc. ACM SOSR*, San Jose, CA, USA, Apr 2019.
- [4] E. Maali, O. Alrawi, and J. McCann, “Evaluating Machine Learning-Based IoT Device Identification Models for Security Applications,” in *Proc. NDSS*, San Diego, California, USA, Feb 2025.

- [5] X. Lin *et al.*, “ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification,” in *Proc. ACM WWW*, Virtual Event, Lyon, France, Apr 2022.
- [6] R. Zhao *et al.*, “Yet Another Traffic Classifier: A Masked Autoencoder Based Traffic Transformer with Multi-Level Flow Representation,” in *Proc. AAAI*, Washington, USA, Feb 2023.
- [7] T. Wang *et al.*, “Netmamba: Efficient Network Traffic Classification Via Pre-Training Unidirectional Mamba,” in *Proc. IEEE ICNP*, Charleroi, Belgium, Oct 2024.
- [8] S. Marchal *et al.*, “AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication,” *IEEE JSAC*, vol. 37, no. 6, pp. 1402–1412, 2019.
- [9] A. Sivanathan *et al.*, “Detecting Behavioral Change of IoT Devices Using Clustering-Based Network Traffic Modeling,” *IEEE IoTJ*, vol. 7, no. 8, pp. 7295–7309, Mar 2020.
- [10] M. A. Ferrag *et al.*, “Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study,” *Journal of Information Security and Applications*, vol. 50, p. 102419, 2020.
- [11] J. Greis, A. Yushchenko, D. Vogel, M. Meier, and V. Steinhage, “Automated Identification of Vulnerable Devices in Networks using Traffic Data and Deep Learning,” *International Journal of Information Privacy, Security and Integrity*, vol. 5, no. 1, pp. 1–17, 2021.
- [12] A. Pashamokhtari *et al.*, “Combining Stochastic and Deterministic Modeling of IPFIX Records to Infer Connected IoT Devices in Residential ISP Networks,” *IEEE IoTJ*, vol. 10, no. 6, pp. 5128–5145, Nov 2022.
- [13] N. Sathesh, M. Rathnamma, G. Rajeshkumar, P. V. Sagar, P. Dadheech, S. Dogiwal, P. Velayutham, and S. Sengan, “Flow-based Anomaly Intrusion Detection using Machine Learning Model with Software-Defined Networking for OpenFlow Network,” *Microprocessors and Microsystems*, vol. 79, p. 103285, 2020.
- [14] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, “IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT,” in *Proc. IEEE ICDCS*, 2017, pp. 2177–2184.
- [15] A. Sivanathan *et al.*, “Managing IoT Cyber-Security Using Programmable Telemetry and Machine Learning,” *IEEE TNSM*, vol. 17, no. 1, pp. 60–74, Feb 2020.
- [16] N. Aphorpe, D. Reisman, and N. Feamster, “A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic,” 2017. [Online]. Available: <https://arxiv.org/abs/1705.06805>
- [17] A. Pashamokhtari *et al.*, “Efficient IoT Traffic Inference: From Multi-View Classification to Progressive Monitoring,” *ACM TIOT*, vol. 5, no. 1, pp. 1–30, Dec 2023.
- [18] S. M. Lundberg and S.-I. Lee, “A Unified Approach to Interpreting Model Predictions,” in *Proc. NIPS*, Long Beach, California, USA, Dec 2017.
- [19] M. T. Ribeiro, S. Singh, and C. Guestrin, ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier,” in *Proc. ACM SIGKDD*, San Francisco, California, USA, Aug 2016, p. 1135–1144.
- [20] R. R. Selvaraju *et al.*, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization,” in *Proc. IEEE ICCV*, Venice, Italy, Oct 2017.
- [21] D. Kowald *et al.*, “Establishing and Evaluating Trustworthy AI: Overview and Research Challenges,” *Frontiers in Big Data*, vol. 7, p. 1467222, 2024.
- [22] L. Fan *et al.*, “An IoT Device Identification Method based on Semi-supervised Learning,” in *Proc. CNSM*, Izmir, Turkey, Nov 2020.
- [23] S. Dong, L. Shu, Q. Xia, J. Kamruzzaman, Y. Xia, and T. Peng, “Device Identification Method for Internet of Things Based on Spatial-Temporal Feature Residuals,” *IEEE TSC*, pp. 1–16, 2024.
- [24] L.-M. and others, “Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things,” *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.
- [25] N. Burkart and M. F. Huber, “A Survey on the Explainability of Supervised Machine Learning,” *JAIR*, vol. 70, p. 245–317, May 2021.
- [26] H. Jmila *et al.*, “A Survey of Smart Home IoT Device Classification Using Machine Learning-Based Network Traffic Analysis,” *IEEE Access*, vol. 10, pp. 97 117–97 141, 2022.
- [27] M. Safi *et al.*, “A Survey on IoT Profiling, Fingerprinting, and Identification,” *ACM TIOT*, vol. 3, no. 4, pp. 1–39, Sep 2022.
- [28] V. Kumar and K. Paul, “Device Fingerprinting for Cyber-Physical Systems: A Survey,” *ACM CSUR*, vol. 55, no. 14s, pp. 1–41, Jul 2023.
- [29] A. Hamza *et al.*, “Verifying and Monitoring IoTs Network Behavior Using MUD Profiles,” *IEEE TDSC*, vol. 19, no. 1, pp. 1–18, May 2020.
- [30] W. He *et al.*, “Can Allowlists Capture the Variability of Home IoT Device Network Behavior?” in *Proc. IEEE EuroS&P*, Vienna, Austria, Jul 2024.
- [31] Y. Zhao, G. Dettori, M. Boffa, L. Vassio, and M. Mellia, “The sweet danger of sugar: Debunking representation learning for encrypted traffic classification,” in *Proceedings of the ACM SIGCOMM 2025 Conference*, ser. SIGCOMM '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 296–310. [Online]. Available: <https://doi.org/10.1145/3718958.3750498>
- [32] A. Henelius, K. Puolamäki, H. Boström, L. Asker, and P. Papapetrou, “A peek into the black box: exploring classifiers by randomization,” *Data Min. Knowl. Discov.*, vol. 28, no. 5–6, p. 1503–1529, Sep. 2014.
- [33] A. A. Freitas, “Comprehensible Classification Models: A Position Paper,” *SIGKDD Explor. Newsl.*, vol. 15, no. 1, p. 1–10, Mar 2014.
- [34] R. Hofstede *et al.*, “Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX,” *IEEE COMST*, vol. 16, no. 4, pp. 2037–2064, May 2014.
- [35] P. Aitken, B. Claise, and B. Trammell, “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information,” RFC 7011, Sep. 2013. [Online]. Available: <https://www.rfc-editor.org/info/rfc7011>
- [36] N. McKeown *et al.*, “OpenFlow: Enabling Innovation in Campus Networks,” *SIGCOMM CCR*, vol. 38, no. 2, p. 69–74, Mar 2008.
- [37] E. Valdez, D. Pendarakis, and H. Jamjoom, “How to Discover IoT Devices When Network Traffic Is Encrypted,” in *Proc. IEEE ICIOT*, Milan, Italy, Jul 2019.
- [38] B. Bezawada *et al.*, “Behavioral Fingerprinting of IoT Devices,” in *Proc. Workshop ASHES*, Toronto, Canada, Jan 2018.
- [39] F. Yin, L. Yang, Y. Wang, and J. Dai, “IoT ETEL: End-to-End IoT Device Identification Method,” in *Proc. IEEE DSC*, Aizuwakamatsu, Fukushima, Japan, Feb 2021.
- [40] A. Pashamokhtari, A. Sivanathan, A. Hamza, and H. Habibi Gharakheili, “PicP-MUD: Profiling Information Content of Payloads in MUD Flows for IoT Devices,” in *Proc. IEEE WoWMoM*, Belfast, United Kingdom, Jun 2022.
- [41] A. Sivanathan, “GitHub—customFlow,” <https://github.com/ArunanSivanathan/customFlow>, [Accessed 04-11-2024].
- [42] H. Guo and J. Heidemann, “IP-Based IoT Device Detection,” in *Proc. ACM IoT S&P*, Budapest, Hungary, Aug 2018.
- [43] (2016) Cache poisoning attack. [Online]. Available: <https://www.sciencedirect.com/topics/computer-science/cache-poisoning-attack>
- [44] Y. Meidan *et al.*, “ProfilIoT: a Machine Learning Approach for IoT Device Identification Based on Network Traffic Analysis,” in *Proc. SAC*, Marrakech, Morocco, Apr 2017.
- [45] O. I. Abiodun *et al.*, “State-Of-The-Art in Artificial Neural Network Applications: A Survey,” *Heliyon*, vol. 4, no. 11, p. e00938, Nov 2018.
- [46] A. Shrestha and A. Mahmood, “Review of Deep Learning Algorithms and Architectures,” *IEEE Access*, vol. 7, pp. 53 040–53 065, Apr 2019.
- [47] S. Mehta *et al.*, “CNN Based Traffic Sign Classification using Adam Optimizer,” in *Proc. ICCS*, Madurai, India, Apr 2019.
- [48] “tf.data: Build TensorFlow input pipelines,” <https://www.tensorflow.org/guide/data>, [Accessed 03-11-2024].
- [49] J. Ahmed *et al.*, “Automatic Detection of DGA-Enabled Malware Using SDN and Traffic Behavioral Modeling,” *IEEE TNSE*, vol. 9, no. 4, pp. 2922–2939, May 2022.