Jibber-Jabber?: Encoding the (Un-)Natural Language of Network Devices and Applications*

Maxwel Bar-on[1], Kiley Krosky[1], Federico Larrieu[1]
Bruhadeshwar Bezawada[2], Indrakshi Ray[1], and Indrajit Ray[1]

[1] Colorado State University, Fort Collins, CO, USA
[2] Southern Arkansas University, Magnolia, AR, USA

Abstract. Modern networks are composed of a diverse group of devices and applications, all of which speak different protocols and exhibit varied network behaviors. Understanding these communication patterns is a critical requirement for a network security analyst to enforce effective access control against malicious behavior. The heterogeneity of devices, the diverse communication patterns and the lack of detailed documentation makes it harder to detect malicious behavior.
Towards this end, we model the network communication patterns akin to natural language and design a custom transformer architecture to provide the necessary comprehension. We use natural language processing (NLP) transformers for the analysis of network traffic flows, especially for those flows exhibiting high spatial contextualization and temporal correlation. Through extensive experiments, we demonstrate that our model provides a reasonably generic understanding of network flows when applied to solve critical network problems such as application identification, device-type fingerprinting and threat identification. We tested our approach on three diverse data sets with the following results: (a) IoT device-type fingerprinting, an average recall of 97%, (b) application identification, an average recall of 99.6% and (c) threat detection, an average recall of 97%.

Keywords: Network traffic modeling, Transformers, IoT fingerprinting, Application identification, Threat Detection

1 Introduction

1.1 Motivation

Modern organizational networks have become increasingly complex with a diverse set of connected computing devices and a large number of network applications. Managing the security of such complex networks is a major challenge

* This work was partially supported by NSF under Grant No. CNS 1822118 and CNS 2226232, Award Numbers DMS 2123761, the member partners of the NSF IUCRC Center for Cyber Security Analytics and Automation – AMI, NewPush, Cyber Risk Research, NIST and ARL, and also the State of Colorado Cybersecurity Center (#SB 18-086) and NIST Grant no. 60NANB23D152.

for the network administrators as they need to be aware of the types of devices and applications present in the network. Accurate identification of the applications and devices helps to identify misbehaving devices and applications, and enforce necessary network access controls to protect the network. This task is challenging since the networks are composed of general computing devices and a significant ratio of Internet-of-Things (IoT) devices and smart devices running different types of applications and protocols.

One approach to identify the devices and applications is by generating a *"fingerprint"*, which is a meaningful representation of their communication patterns, from the network traffic traces they generate. However, the vast diversity in the networking protocols used by these devices and the complex communication patterns[3] of the applications make this a non-trivial task. We propose a solution to this problem and help network security administrators in their goal of securing their networks against internal and external threats.

1.2 Problem Statement

Our problem statement can be stated as the design of two algorithms: A_f, a fingerprinting algorithm and M, a matching algorithm described below. Without loss of generality, we use the terms *device* and *application* interchangeably.

System Definitions. Formally, let $T_t : \{\rho_1, \rho_2, \cdots, \rho_n\}$ denote a fixed size network packet traffic trace captured at a time snapshot Δ_t, ρ_i denotes the i^{th} packet in the trace, and $PROT$ is the set of protocols in the trace. A protocol element $prot \in PROT$ defines a relation R_i on the data and control fields of a packet in T_t. Therefore, $PROT$ is a collection of all such relations: $PROT : R_1 \times R_2 \times \cdots \times R_{|PROT|}$. By extension, packet ρ_i can be viewed as a subset of $PROT$ as each packet contains various protocols and their field values. A fingerprint F_s, of a device or application, is a concise representation that is generated out of a given network trace T_s captured at some random time snapshot s.

The fingerprinting problem is to design an algorithm A_f that generates a fingerprint F_s by taking as input a traffic trace $T_{\{s \in \Delta\}}$ and $PROT$ where Δ is the time duration of network trace collection. Now, if the same device generates another trace T_x, which is different from T_s, at another time-snapshot x, A_f can compute the fingerprint F_x of T_x. The fingerprint matching problem is to design a matching algorithm M that accurately compares the two fingerprints F_s and F_x, to confirm the identification of the same device.

1.3 Limitations of Prior Art

Several existing works [9,20,10] have addressed the extraction of grammar rules for network protocols and applied the results for classifying traffic, and identifying applications. Network traffic classification [2,15,16,12,27,3,23] is the task of

[3] *Jibber-Jabber* or Gibberish refers to incomprehensible speech patterns like speaking too fast or too slow and/or using strange vocabulary.

assigning sequences of packets to different categories based on their characteristics. Although the problem of network traffic classification is well studied, we focus only on solutions that proposed deep learning models and identify their shortcomings.

- *Bi-directional flow semantics and shortage of data.* Existing literature has focused on using IP flow-based semantics, especially bi-directional flows, i.e., incoming and outgoing packets of the same flow. However, from our observations and experiments, we found that many IoT devices and smart applications do not generate sufficient bi-directional data, which makes it difficult to apply many existing results in practice.
- *Related flow semantics.* We observed that a particular device or application typically generates several uni-directional flows to perform a specific task. We note that existing literature has not explored the relational semantics of such uni-directional flows, as most such flows would be considered independent and processed as such.
- *Lack of multi-class network classifiers.* We note that many existing methods perform poorly when classifying network data into multiple classes due to the constraints of the chosen deep learning model to learning the conversational features of the network traffic.

1.4 System and Threat Model

System Model. Our system consists of a group of IoT devices and smart applications running over an organizational network. The inward and outward traffic flows from these devices are subject to monitoring by the network administrator. The network administrator can perform deep analysis and enforce necessary access controls.

Threat Model. We consider the threats that cause devices and applications to exhibit significant deviation from their normally acceptable or understood behavior. For instance, if a device is compromised and is generating high amounts of traffic in contrast to its original specification, such a device needs to be detected and isolated. We also consider active threats into the network from external and internal actors who might be performing scanning, reconnaissance or malware injection via the network.

1.5 Proposed Approach

In our approach, to overcome the limitations in prior art, we model the network traffic packets as natural language data. For NLP, input samples are sequences of language "words"; whereas, for network traffic, input samples are sequences of packets. Compared to multi-variate time-series problems, such as weather forecasting, network traffic has a closer resemblance to natural language because each time step represents a single event instead of multiple interrelated events.

To address the first limitation of lack of bi-directional flow data, we consider packets within a specified window of view and process all flows within this window. This ensures that our method is not limited by the requirement of sufficient

bi-directional flows from a device. To address the second limitation of relative flow semantics, we design *relative features*, which capture the relations among the various packets and flows within the same window. This ensures that we cover the case where a device uses several, seemingly unrelated, uni-directional flows for performing a certain task. Finally, to address the third limitation of multi-class classification, we choose the NLP transformer as our classification architecture because of its ability to encode relationships between words in a sentence. The transformer was originally developed for natural language translation [26], but has since been applied to other NLP tasks [6], including classification [21].

1.6 Key Contributions

Our contributions are summarized as follows:

- We describe a window based data model to create the analogy to natural language sentences and NLP transformers to solve the problem of network traffic analysis.
- We describe relative features for capturing the inter-dependencies of correlated uni-directional communications originating from a device.
- We describe a multi-class classification technique based on our approach and use it to address a few network security problems such as IoT device-type fingerprinting, application identification, and threat identification.
- We provide a comprehensive evaluation of our approach on real-world data sets.

Organization The paper is organized as follows. Section 2 outlines background and related work. Section 3 describes our proposed approach and Section 4 describes our classification architecture. Section 5 discusses the performance on various tasks. Finally, Section 6 outlines our conclusions and future work.

2 Related Work

2.1 NLP Methods for Networking Problems

Bar et al. [2] use NLP techniques to distinguish between malicious and benign network traffic, and to identify the use of VPNs (Virtual Private Networks). They use a SimCSE-based model to extract vector representations of packets, then classify the vector embeddings using SVMs and Random Forests. Meng at al. [16] propose an architecture for producing high-quality latent representations of network packets to improve the performance of traffic classification tasks. The drawback of these approaches is their use of traffic features, such as IP addresses and encrypted payloads, which results in overfitting the deep-learning model. For example, a classification model may learn to associate IP addresses of packets with a particular class; however, the IP addresses in the training data will likely differ from unseen samples as they are assigned dynamically. In our work represents each packet as a single event, similar to words in a sentence.

2.2 IoT Fingerprinting

IoT-Portrait is a transformer-based architecture, proposed by Wang et al. [27], for identifying the types of IoT devices based on sequences of traffic generated by the devices. This architecture consists of a transformer encoder followed by a linear multinomial classifier. We use a similar approach to [27]; however, we introduce more meaningful features to increase the amount of useful information contained in packet-feature vectors.

Alioghli et al. [1] address the challenge of detecting anomalies in multivariate time-series data generated by IoT devices. The authors evaluate different positional encoding (PE) mechanisms in transformer networks, including Absolute PE, Rotary PE, and two modifications of Relative PE: Representative attention and Global attention. Msadek et al. [18] explore a machine learning approach to identify IoT devices based on encrypted traffic patterns. Their approach includes a segmentation technique using an adaptive sliding window, feature extraction from packet headers, and the evaluation of multiple machine learning classifiers such as k-Nearest Neighbors (KNN), Support Vector Machine (SVM), Random Forest (RF), AdaBoost, and Extra-Trees.

The IoTTFID [8] approach focuses on identifying new IoT devices by analyzing their network traffic fingerprints. It involves extracting device traffic fingerprints, converting them into feature vectors, and updating the original model to recognize new devices. Luo et al. [14] developed a novel IoT device-type identification framework based on transformer models to handle heterogeneous IoT traffic. In the first stage, they use a transformer-based traffic diagnosis model to classify traffic into normal and abnormal types. In the second stage, they employ another transformer-based model to identify the device type from the normal traffic, and improve accuracy with a results-ensemble algorithm.

Yin et al. [30] introduce GraphIoT, a method for IoT device detection that uses graph classifiers to analyze lightweight flow information like packet length, direction, and timestamp. This data is transformed into an IoT Device Traffic Graph Representation (IoT-DTGR), which is then classified using a Graph Neural Network (GNN) that considers node and edge features and subgraph structures.

2.3 Application Identification

A study done by Yamansavascilar et al. [29] focuses on identifying individual applications, such as Facebook, Twitter, and Skype, through network traffic classification using machine learning methods. Unlike previous studies, which classified applications into broad categories such as FTP, HTTP, VoIP, Instant Messaging (IM) or Streaming (Video, Music, or Gaming), this research aims to identify specific applications. The authors used the UNB ISCX Network Traffic dataset and their internal dataset to evaluate four classification algorithms: J48, Random Forest, k-Nearest Neighbors (k-NN), and Bayes Net.

Miskovic et al. [17] propose AppPrint, a system for automatically fingerprinting mobile applications. Traditional methods typically rely on app identifiers like

User-Agent fields or analytics services, but AppPrint can determine app identities from generic HTTP traffic, even when identifiers are not included.

The APPSNIFFER framework [19] addresses the limitations of existing mobile app fingerprinting systems referenced above like FlowPrint and AppScanner, when Virtual Private Networks (VPNs) are used. It uses a two-stage classification process: the first stage distinguishes VPN traffic from normal traffic, while the second stage identifies specific mobile apps using a stacked ensemble model combining Light Gradient Boosting Machine (LightGBM) and a FastAI library-based neural network.

AppScanner is a framework designed to fingerprint and recognize Android apps solely from encrypted network traffic [24]. By running apps on physical devices, AppScanner collects network traces, pre-processes them to separate individual app flows, and applies supervised learning for classification. Taylor et al. [25] extended this to develop a technique for recognizing smartphone apps through encrypted network traffic. The main enhancement is that it introduces ambiguity detection using reinforcement learning to handle common traffic from shared libraries like ads or analytics services.

The Fine-Grained Open-World Android App Fingerprinting (FOAP) approach [11] aims to identify method-level fine-grained user actions of Android apps when the apps are unknown to the system. It involves open-world app recognition by filtering out irrelevant traffic segments and uses a novel metric called structural similarity to focus on the necessary parts.

2.4 Threat Detection

Shaukat et al. [22] evaluates the performance of three primary classifiers - deep belief network (DBN), decision tree, and support vector machine (SVM), assessed on benchmark datasets such as KDD CUP 99, NSL-KDD, Spambase and Twitter dataset. The study found that DBN performs best for detecting spam, decision trees are most effective for intrusion detection, and SVM achieves the highest accuracy for malware classification.

Sarhan et al. [4] presents a machine learning-based approach for insider threat detection using the CERT insider threat dataset. It utilizes Deep Feature Synthesis (DFS) that automates feature engineering, resulting in 69,738 features, and then used PCA to mitigate dimensionality issues and determine the most important characteristics in the dataset.

3 Proposed Approach

3.1 Sliding Window Network Data Model

In our data model for network traffic, the network traces are divided into overlapping sequences of packets where each sequence is a *"sentence"* and each packet is a *"word"*. Our network datasets consist of .pcap files containing network traffic collected by recording packets transmitted over the relevant network interface.

Input samples are generated by sliding a window of size w over the collected traffic and extracting 17 features, *standard* and *relative*, from each sub-sequence of w packets. For each packet ρ_t where $w < t \leq n$ is the index in a traffic trace of length n, this will yield an input sample $X \in \mathbb{R}^{w \times 17} = [\rho_{t-w}, \rho_{t-w+1}, \ldots, \rho_t]$. Each input sample is tagged with a corresponding label T_t, specifying the class of the sample. For most network datasets [5,3,13], we annotate samples using the labels of their corresponding .pcap files since each file is associated with a single class. Conversely, for the UNSW IoT dataset [23], we annotate samples using the known MAC addresses of IoT device-types connected to the UNSW network. We filter packets by their MAC address prior to applying the sliding-window approach to ensure that all packets in a window will be from the same device-type.

Our feature set includes 13 standard network features, as well as 4 relative features that are extracted through our novel relative encoding technique.

3.2 Standard Features

Standard features are extracted from each packet in a window. These features encode basic information about the protocol and payload of the packet and include a mixture of binary and continuous features as shown in Table 1. Payload Length and Payload Entropy are statistics describing the payload of a packet, while the remaining features are extracted from header fields.

Table 1: Packet Features

Feature	**Explanation**
Subnet Src	1 if source IP is in subnet else 0
Subnet Dst	1 if destination IP is in subnet else 0
Broadcast Dst	1 if destination IP is broadcast else 0
TLS	1 if packet includes TLS handshake header else 0
TCP	1 if packet includes TCP header else 0
UDP	1 if packet includes UDP header else 0
HTTP	1 if packet uses HTTP port else 0
SSL	1 if packet uses SSL port else 0
Common Src	1 if packet uses common source port else 0
Common Dst	1 if packet uses common dst port else 0
Header Length	length of transport-layer header (bytes)
Payload Length	length of payload (bytes)
Payload Entropy	information entropy of payload

3.3 Relative Features

Relative features are encoded representations of the endpoints of packets and are useful for capturing relationships among packets and flows within the window. To generate these features, we use a relative encoding scheme that maps

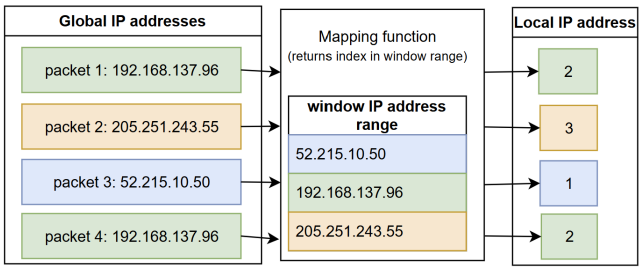

Fig. 1: Deriving relative feature values for an example endpoint-identifier (IP source address) in a window with 4 packets.

each endpoint-identifier field from a global scale to a local scale. Using a local scale preserves relationship information while reducing the dimensionality of our model and preventing overfitting. We interpret these as categorical features to avoid imposing any arbitrary ordering on the values within a local scale. Relative features include localized representations for each of the following four endpoint-identifier fields:

<*source IP address*, *destination IP address*, *source port*, *destination port*>.

These four fields act as a "flow-label" for each packet. Unlike the common approach of manually filtering packets based on flows prior to classification, flow-labels allow the model to identify relationships between packets within multi-flows. This is useful for understanding complex network behaviors that involve multiple overlapping flows.

As shown in Fig. 1, we use a mapping function to convert global endpoint-identifier values to a local scale. Let $S = \{P^{(1)}, P^{(2)}, \ldots, P^{(n)}\}$ be a window of n packets where each packet $P^{(j)}$: (*src-IP,dst-IP,src-prt,dst-prt*) is defined as a four-tuple containing the packet's endpoint-identifiers. To obtain the relative features for a packet, we use the mapping function $R(P^{(j)}, S) \rightarrow \mathbb{Z}^4$. We define the mapping function for endpoint-identifier field i of packet $P^{(j)}$ as follows:

$$R(P^{(j)}, S)_i = \left| \{k \mid k \leq P_i^{(j)} \wedge k \in \pi_i(S)\} \right| \tag{1}$$

where $P_i^{(j)}$ is the global value of the ith endpoint-identifier field for packet $P^{(j)}$ and $\pi_i(S)$ is the set of unique values for the field i found in window S. Essentially, the mapping function counts the number of unique values for field i in window S that are less than or equal to $P_i^{(j)}$.

Justification of Relative Features. Relative encoding preserves semantic properties that allow a deep-learning model to identify relationships between packets in a sequence. These relationships enhance the model's understanding of multi-packet behaviors contained in a traffic trace, which improves its decision making. Due to the size of the global scales for endpoint-identifiers, there is minimal overlap between the unique sets of values associated with different classes

of traffic in the available training data. As a result, using global values will cause the model to become reliant on endpoint-identifiers for assigning labels to traffic samples. This is undesirable because unseen samples of traffic won't necessarily use the exact same global endpoint-identifiers observed in the training data, causing the model to misclassify them. Relative features are less rigid, which prevents the model from memorizing correlations between endpoint-identifiers and traffic labels. By mapping values to a local scale, we ensure that classes in the training data cannot be uniquely identified by their endpoint-identifiers. This teaches the model to use flow-labels for finding relationships between packets.

Illustrative Example. To illustrate how relative features can reveal relationships, we use an example window S of five packets ($|S| = 5$) with the following endpoint-identifiers and relative features:

- $R\left(P^{(1)} : (192.168.137.96,\ 205.251.243.55,\ 42000,\ 443),\ S\right) = [1, 2, 3, 2]$
- $R\left(P^{(2)} : (205.251.243.55,\ 192.168.137.96,\ 443,\ 42000),\ S\right) = [2, 1, 1, 3]$
- $R\left(P^{(3)} : (192.168.137.96,\ 205.251.243.55,\ 45000,\ 137),\ S\right) = [1, 2, 4, 1]$
- $R\left(P^{(4)} : (192.168.137.96,\ 205.251.243.55,\ 42000,\ 443),\ S\right) = [1, 2, 3, 2]$
- $R\left(P^{(5)} : (192.168.137.96,\ 205.255.4.123,\ 40000,\ 137),\ S\right) = [1, 3, 2, 1]$

Based on these features, we can see that $P^{(1)}$ and $P^{(4)}$ belong to the same unidirectional network flow, $P^{(2)}$ is the only incoming packet, $P^{(3)}$ is communicating with the same host machine as $P^{(1)}$ and $P^{(4)}$ but is part of a separate flow, and $P^{(5)}$ is using the same type of service as $P^{(3)}$ (audio/video streaming) but on a different host machine. We can estimate the relationships between packets in this example by calculating the attention, as described in the transformer architecture in [26], between indicator-variable representations of relative features for each packet, a common machine-learning technique for representing categorical values. We note that, we only use indicator-variables for illustrating how the attention mechanism identifies relationships between packets. In our architecture, we implement relative features as lookup-table indices instead to reduce storage and computational overhead.

Let $\delta(P^{(j)}, S) \in \{0, 1\}^{4*|S|}$ be a vector containing the indicator-variable representations for each relative feature of a packet $P^{(j)} \in S$ defined as:

$$\delta(P^{(j)}, S) = \left[\mathbb{I}\left(R(P^{(j)}, S)_1\right) \| \ldots \| \mathbb{I}\left(R(P^{(j)}, S)_4\right)\right] \tag{2}$$

where $\mathbb{R}^m \| \mathbb{R}^n \to \mathbb{R}^{m+n}$ is the element-wise concatenation of vectors and $\mathbb{I}(v) \to \{0, 1\}^{|S|}$ returns the indicator-variable representation for an endpoint-identifier value v as follows:

$$\mathbb{I}(v)_i = \begin{cases} 1 & \text{if } i = v \\ 0 & \text{else} \end{cases} \tag{3}$$

Using these representations, we calculate an attention matrix M for S using the attention operation [26] as follows:

$$M_{i,j} = \frac{e^{\left(\sum_k^{4|S|} \delta(P^{(i)},S)_k * \delta(P^{(j)},S)_k\right)}}{\sum_h^{|S|} e^{\left(\sum_k^{4|S|} \delta(P^{(i)},S)_k * \delta(P^{(h)},S)_k\right)}} \tag{4}$$

We show the attention matrix M for this example in Table 2. This matrix shows that the attention scores between packets is higher when the packets are related. $P^{(2)}$ has a high attention score with itself because it is the only incoming packet. The attention values along the diagonal are smaller for outgoing packets because there are more of them in the window. The attention values between $P^{(1)}$ and $P^{(4)}$ are relatively high, reflecting the strong relationship between packets in the same uni-directional flow.

	$P^{(1)}$	$P^{(2)}$	$P^{(3)}$	$P^{(4)}$	$P^{(5)}$
$P^{(1)}$	0.4538	0.0083	0.0614	0.4538	0.0226
$P^{(2)}$	0.0171	0.9317	0.0171	0.0171	0.0171
$P^{(3)}$	0.0950	0.0129	0.7021	0.0950	0.0950
$P^{(4)}$	0.4538	0.0083	0.0614	0.4538	0.0226
$P^{(5)}$	0.0397	0.0146	0.1080	0.0397	0.7979

Table 2: Attention matrix.

4 Transformer Architecture for Traffic Classification

Our architecture consists of three modules: an embedding layer, a Transformer encoder, and a fully-connected neural network classifier as shown in Fig. 2.

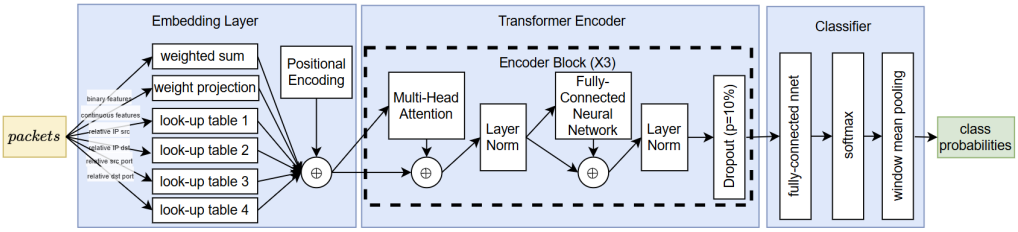

Fig. 2: **Classification Architecture.**

4.1 Embedding Layer

The purpose of the embedding layer is to produce representations of input samples in a lower-dimensional continuous vector space [26] that encodes desired semantic properties. This layer is responsible for handling the heterogeneous packet features, allowing the encoder and classifier to operate on homogeneous representations.

Given a packet feature vector $X \in \mathbb{R}^{17}$, where $X_{1 \leq j \leq 10} \in \{0,1\}^{10}$ are binary standard features, $X_{11 \leq j \leq 13} \in \mathbb{R}^3$ are continuous standard features, and $X_{14 \leq j \leq 17} \in \mathbb{Z}^4$ are relative features, this layer will produce a vector $E \in \mathbb{R}^d$, where d is the embedding dimensionality of the model.

The embedding layer is configured to handle each subspace of X according to the type of features in that region. The layer produces a d-dimensional embedding vector for each region, which are combined through element-wise addition.

For binary features ($X_{1 \leq j \leq 10}$), the embedding layer uses a learnable weight matrix $\mathbb{W}^{(b)} \in \mathbb{R}^{10 \times d}$ and produces an embedding vector $E^{(b)} \in \mathbb{R}^d$ as follows:

$$E_i^{(b)} = \sum_{j=1}^{10} \begin{cases} \mathbb{W}_{j,i}^{(b)} & \text{if } X_j = 1 \\ 0 & \text{if } X_j = 0 \end{cases} \tag{5}$$

For continuous features ($X_{11 \leq j \leq 13}$), the embedding layer uses a learnable weight matrix $\mathbb{W}^{(c)} \in \mathbb{R}^{3 \times d}$ and produces an embedding vector $E^{(c)} \in \mathbb{R}^d$ as:

$$E_i^{(c)} = \sum_{j=11}^{13} \mathbb{W}_{j,i}^{(c)} * X_j \tag{6}$$

Finally, the categorical relative features ($X_{14 \leq j \leq 17}$) are handled using a set of 4 lookup-tables implemented as a tensor $\mathbb{W}^{(r)} \in \mathbb{R}^{4 \times w \times d}$ with one weight matrix per relative feature where the number of rows is equivalent to the number of categories for the corresponding feature and the number of columns is equal to d. Finally, the categorical relative features ($X_{14 \leq j \leq 17}$) are handled using a set of 4 lookup-tables implemented as a tensor $\mathbb{W}^{(r)} \in \mathbb{R}^{4 \times w \times d}$, where each slice $\mathbb{W}_i^{(r)}$ is a weight matrix corresponding to a relative feature i. The number of rows in $\mathbb{W}_i^{(r)}$ is equivalent to the number of values in the local range for i and the number of columns is equal to d. Lookup-tables are used by returning the row corresponding to a relative feature value. The selected row vectors from each table are combined through element-wise addition. We set the number of rows to the window size (w) for all lookup-tables since this is the maximum possible number of unique values for a particular endpoint-identifier that may be observed in a window of w packets. Therefore, each relative feature will be in the range $[1, w]$. For the relative features, the embedding layer will produce a vector $E^{(r)} \in \mathbb{R}^d$ as follows:

$$E_i^{(r)} = \sum_{j=14}^{17} \mathbb{W}_{j-13, X_j, i}^{(r)} \tag{7}$$

Putting it all together, the output of the embedding layer is defined as:

$$E = E^{(r)} \oplus E^{(c)} \oplus E^{(b)} \oplus PE_t \tag{8}$$

where $\oplus$ denotes the element-wise addition of two vectors, $PE \in \mathbb{R}^{w \times d}$ is a fixed positional encoding [26], and t is the index of packet X in its associated window.

Lookup-tables allow us to store categorical variables as integers instead of one-hot-vectors while maintaining orthogonal separation of categories. Let v be a relative feature value with corresponding lookup-table $\mathbb{W} \in \mathbb{R}^{w \times d}$. The indicator-variable representation $\mathbb{I}(v) \in \{0,1\}^d$ of v is defined according to Equation 3. Our lookup-table approach will represent v as the row-vector $\mathbb{W}_v \in \mathbb{R}^d$, which is equivalent to taking the dot-product of $\mathbb{I}(v)$ and $\mathbb{W}$:

$$\mathbb{W}_{v,i} = \sum_{j}^{w} \mathbb{W}_{j,i} \mathbb{I}(v)_i \tag{9}$$

Therefore, this representation preserves the following orthogonality property of the categorical variables: $\sum_j^w \mathbb{I}(a)_j \mathbb{I}(b)_j = 0$ if $a \neq b$ where a and b are examples of a relative feature with w categories.

We note that, although $d > 17$, we consider the embedding space to be lower-dimensional because each relative feature carries the same amount of information as a $[w]$-dimensional one-hot vector. The true feature dimensionality is $13 + 4w$; consequently, the total number of learnable parameters in the embedding layer is $d(13 + 4w)$, which represents the number of parameters used to map packets from a $[13 + 4w]$-dimensional heterogeneous representation to a $[d]$-dimensional continuous representation. Given an input window $X \in \mathbb{R}^{w \times 17}$, this layer will produce an embedding sequence $E \in \mathbb{R}^{w \times d}$, which is input to the Transformer encoder module. Please see Appendix 1 for details of encoding and Appendix 2 for details of the classifier.

4.2 Training

We jointly train all components of our architecture to minimize the negative-log-likelihood loss between the classifier's predictions and the true labels of our training samples. Let C be the set of classes, $X \in \mathbb{R}^{n \times w \times 17}$ be a batch of n samples with labels $T \in \{0,1\}^{n \times |C|}$ defined as:

$$T_{i,j} = \begin{cases} 1 & \text{if } class(X_i) = C_j \\ 0 & otherwise \end{cases} \tag{10}$$

Now let $f(X;\theta) \rightarrow P$ be a model parameterized by θ, that, when given X, will produce probabilities $P \in [0,1]^{n \times |C|}$ over C. We define our loss function $\mathcal{L}(P,T)$ as:

$$\mathcal{L}(P,T) = \frac{-1}{n|C|} \sum_{i}^{n} \sum_{j}^{|C|} T_{i,j} \ln(P_{i,j}) \tag{11}$$

Using this, we train our model by approximating the optimization problem:

$$\theta \approx \min_{\theta} (\mathcal{L}(f(X;\theta), T)) : \forall (X, T) \in D_{tr} \tag{12}$$

where $D_{tr} \subset D$ is the set of all window-label pairs in the training data and D is the set of collected data.

We train our model using *Adam* on the training data with an initial learning rate of 0.01 for a maximum of $10,000$ epochs and evaluate $\mathcal{L}$ on a validation set ($D_{val} \subset D$) after every epoch, saving the model's parameters if the loss is lower than the previous lowest validation loss. If the validation loss does not improve for 15 epochs, we decay the learning rate by a factor of $3\times$ and halt the training if the learning rate decays below the threshold 0.0001. After training, we evaluate the model on the test set ($D_{te} \subset D$) using the parameters that achieved the lowest validation loss. Our data-partitioning process ensures that these subsets of D will have no overlap ($D_{val} \cap D_{tr} \cap D_{te} = \{\emptyset\}$) to ensure a fair evaluation of trained models and prevent overfitting through validation-set guided training.

5 Performance Evaluation

We evaluate our approach for network traffic classification on three tasks:

1. IoT fingerprinting: identifying the type of an unknown IoT device
2. Application identification: determining the identity of an application
3. Threat detection: determining whether a process is malicious or benign

For each task, we evaluate our approach by training and testing our model using 5-fold cross validation and report the average classification metrics over all folds. We use a 7:1:2 (train : validation : test) ratio for data partitioning.

5.1 Task and Architecture Details

Task	**IoT**	**Application**	**Threat**
Data Size(MB)	0.67	1.38	1.38
# samples	302,451	191,979	191,979
# classes	19	9	2

Table 3: Sizes of datasets for each task.

In Table 3, we show the size of the datasets used for each task. For IoT fingerprinting, our dataset consists of 302,451 total windows from 19 unique IoT device-types, combining data from 3 sources: Colorado State University [3] (2

device-types), Canadian Institute of Cybersecurity [5] (5 device-types), and University of New South Wales [23] (12 device-types). we only include device-types with at least 10,000 packets from each source. For the Application Identification task, we use 191,979 windows from 9 different applications extracted from the public USTC-TFC dataset [13]. In this dataset, 6 applications are benign while 3 are compromised by malware. We also use the USTC-TFC dataset for the threat detection task; however, we treat this as a binary classification problem, rather than a multinomial classification problem. For threat detection, we label samples from the 6 benign applications as 'benign' and samples from the 3 compromised applications as 'malicious'.

	Task	**IoT**	**Application**	**Threat**
Hyperparameter	Window Size	32	100	100
	Heads	4	4	4
	attention dim	16	32	16
	embedding dim	40	40	40
	$\alpha^{(h)}$ Dropout	10%	0.0%	10.0%

Table 4: Architecture Hyperparameter configurations for each task

We explore a variety of different hyperparameter values to discover the optimal configurations of our architecture for each task. In Table 4, we show the window size, number of heads (H), attention dimensionality (u), embedding dimensionality (d), and dropout rate applied to the outputs of each head during training. We also apply dropout with a rate of 10% to the output of each encoder block for all tasks.

5.2 Performance

		IoT		**Application**		**Threat**	
	Features	all	standard	all	standard	all	standard
Metric	Recall	97.11	96.0	99.64	97.84	97.32	96.77
	Precision	97.13	96.06	99.38	97.71	96.93	96.14
	Accuracy	97.07	95.95	99.6	97.85	96.74	96.07

Table 5: Average Recall, Precision, Accuracy on each task with all features vs. standard features

In this section, we evaluate the performance of our approach on our selected network classification tasks. For each task, we evaluate our architecture with and without relative features to outline the benefits of including relative features for identifying relationships between packets.

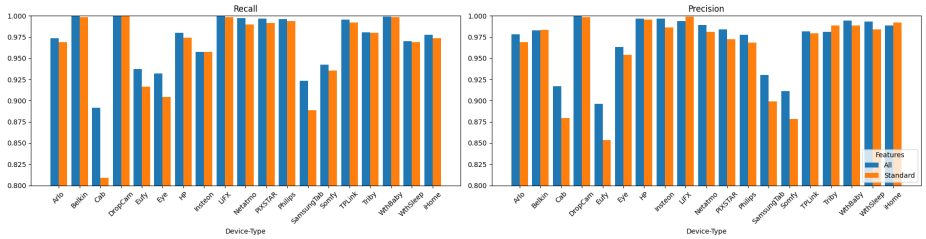

Fig. 3: Performance on IoT fingerprinting task. The blue bars show the performance with all 13 features and the orange bars with standard features.

IoT Fingerprinting Results. Our architecture achieves a strong performance on the IoT fingerprinting task. As shown in Table 6, we achieve an average device identification rate (recall) over 97%, which demonstrates that our model is able to correctly identify the majority of IoT traffic samples. If a fingerprinting model is used for security profiling, it is important to maintain a high recall score to ensure correct identification of insecure device-types on a network. Meanwhile, we achieve a precision of 0.9713 which shows that the model's predictions are reliable. If a fingerprinting model is used for access control, it is important to maintain a high precision score because, otherwise, it may impose incorrect access-control policies on device-types. Without relative features, the average precision is reduced by 1.1% while the recall is reduced by 1.14%. The performance improvement from using relative features is particularly evident in complex device-types such as "Cab" and "Eufy" as shown in Figure 3. With relative features, the model is able to understand the complex network behaviors of these device-types. These results demonstrate the potential of relative features for improving the performance and reliability of IoT fingerprinting models.

Application Identification. Our model performs well for application identification for the nine (9) applications considered with an average recall of 99.6%. These results also show the importance of relative features as the difference in performance is over 5% for some applications. Since applications have more complex communication patterns, it stands to reason that relative features have a higher bearing on the results.

Threat Detection. Our approach performs well for threat detection as shown in Table 6. Our model is capable of identifying threats with over 99% recall. For

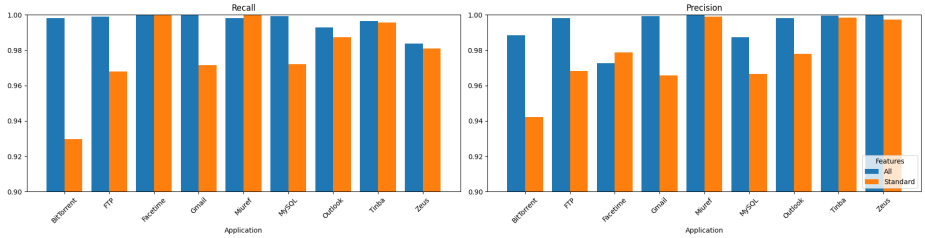

Fig. 4: Performance on the Application Identification task. The blue bars show the performance with all 13 features and the orange bars with standard features.

threat detection, the inclusion of relative features does show improvement on average, increasing F1 scores for both classes. We note that, a detailed empirical or experimental comparison to existing approaches is not provided in the interest of space.

		Benign		**Malicious**	
	Features	all	standard	all	standard
Metric	Recall	0.95	0.94	0.996	0.995
	Precision	0.998	0.997	0.941	0.926
	F1	0.973	0.968	0.968	0.959

Table 6: Performance on the Threat Detection task with all features vs. only standard features.

6 Conclusion and Future Work

In this work, we focused on the problem of network traffic classification using NLP style data modeling and techniques. Leveraging the NLP data model, we described relative encoding of packets found within a window of observation, enabling us to capture flow semantics more effectively. We used a transformer architecture to experiment on three diverse applications: IoT device-type fingerprinting, application identification and threat detection, and achieved high performance across various metrics of importance. Relative features resulted in significant improvement in results, between $2-6\%$ across several classes of traffic. We were able to demonstrate that relative features capture some general characteristics of network traffic flows. In the future, we envisage that our data model and features will result in more applications within the network traffic analysis domain. Development of different types of transformer models, for diverse data sets, may also be an active area for exploration.

Appendix 1 Transformer Encoder

We use a Transformer encoder similar to the encoder described in the machine translation architecture described by Vaswani *et al.* [26]. This module receives the output of the embedding layer and applies a series of alternating attention and fully-connected neural network sub-layers, surrounded by residual connections and layer normalizations. Each of these sub-layers is contained in an encoder block and the encoder component includes 3 of these blocks. The attention sub-layers capture temporal relationships between packets, fully-connected sub-layers prevent embeddings in a window from converging [7], residual connections increase the strength of gradient signals to deeper layers and normalization [28] ensures the stability of gradients during training. We use the superscript notation $E^{(n)}$ to refer to embedding sequences where the superscript (n) denotes the position of the embedding in the stack of encoder blocks. For example, the input to the first block is the output of the embedding layer $E^{(1)}$, while the output of the first block and input to the second block is $E^{(2)}$.

In each block, the attention sub-layer has learnable weights $\mathbb{W}^{(Q)}, \mathbb{W}^{(K)}, \mathbb{W}^{(V)} \in \mathbb{R}^{d\times u}$, $\mathbb{W}^{(m)} \in \mathbb{R}^{u\times d}$ and biases $\mathbb{B}^{(Q)}, \mathbb{B}^{(K)}, \mathbb{B}^{(V)} \in \mathbb{R}^{u}$, $\mathbb{B}^{(m)} \in \mathbb{R}^{d}$ where u is the total dimensionality of all attention heads. Let $E^{(n)} \in \mathbb{W}^{w\times d}$ be the input to encoder block n. The attention sub-layer first applies three affine transformations (Query, Key, Value) to the input, which are then split into H $[\frac{u}{H}]$-dimensional subspaces where H is the number of attention heads. Using multiple heads allows the attention layer to identify multiple separate relationship patterns for each packet which improves the modeling capabilities of the transformer. Let $Q^{(h)}, K^{(h)}, V^{(h)} \in \mathbb{R}^{w\times \frac{u}{H}}$ be the Query, Key, and Value transformations respectively obtained using the attention sub-layer's learnable parameters as follows:

$$\begin{aligned} Q^{(h)} &= E^{(n)}\mathbb{W}^{(Q)}_{*,(\frac{hu}{H}<i<\frac{(h+1)u}{H})} + \mathbb{B}^{(Q)} \\ K^{(h)} &= E^{(n)}\mathbb{W}^{(K)}_{*,(\frac{hu}{H}<i<\frac{(h+1)u}{H})} + \mathbb{B}^{(K)} \\ V^{(h)} &= E^{(n)}\mathbb{W}^{(V)}_{*,(\frac{hu}{H}<i<\frac{(h+1)u}{H})} + \mathbb{B}^{(V)} \end{aligned} \tag{13}$$

where $M_{*,(\frac{hu}{H}<i<\frac{(h+1)u}{H})}$ is an indexing operation that returns all rows of a matrix M and columns $\frac{hu}{H}$ through $\frac{(h+1)u}{H}$. The attention operation is then applied over each head as follows:

$$\alpha^{(h)} = softmax(\frac{Q^{(h)}{K^{(h)}}^{T}}{\sqrt{\frac{u}{H}}})V^{(h)} \tag{14}$$

The attention values for all heads are concatenated and combined through a fourth affine transformation as follows:

$$Z = [\alpha^{(1)}\|\alpha^{(2)}\|\ldots\|\alpha^{(H)}]\mathbb{W}^{(m)} + \mathbb{B}^{(m)} \tag{15}$$

Finally, the combined value is added to the input and normalized to produce the output of the attention sub-layer $A \in \mathbb{R}^{w\times d}$ as:

$$A = LayerNorm(Z \oplus E^{(n)}) \tag{16}$$

where $LayerNorm$ is defined as:

$$LayerNorm(X)_{i,j} = \frac{X_{i,j} - \mu(X_{*,j})}{\sigma(X_{*,j})} \tag{17}$$

and $\mu(X_{*,j})$ and $\sigma(X_{*,j})$ return the mean and standard deviation of the jth feature of X respectively.

The fully-connected layer of each block has learnable weights $\mathbb{W}^{(h)}, \mathbb{W}^{(o)} \in \mathbb{R}^{d \times d}$ and biases $\mathbb{B}^{(h)}, \mathbb{B}^{(o)} \in \mathbb{R}^{d}$. This sub-layer receives the output of the attention sub-layer and produces the value $Z \in \mathbb{R}^{w \times d}$ defined as:

$$Z = max(0, A\mathbb{W}^{(h)} + \mathbb{B}^{(h)})\mathbb{W}^{(o)} + \mathbb{B}^{(o)} \tag{18}$$

This is followed by an additional residual connection and normalization to produce the output of the nth encoder block $E^{(n+1)} \in \mathbb{R}^{w \times d}$ as follows:

$$E^{(n+1)} = LayerNorm(Z \oplus A) \tag{19}$$

This becomes the input to the subsequent encoder block $n+1$ unless n is the final block in the stack ($n = 3$), in which case, it becomes the input to the classifier module $E^{(4)}$.

Appendix 2 Classifier

The classifier module is a fully-connected neural network that receives the output of the Transformer encoder ($E^{(4)}$) and predicts the corresponding class. For each element in an input sequence, it produces a probability distribution over the set of all classes C. We then take the average over the elements of this sequence and use the average probability distribution to identify the class.

This component uses its neural network with learnable weights $\mathbb{W}^{(h)} \in \mathbb{R}^{d \times 80}$,$\mathbb{W}^{(o)} \in \mathbb{R}^{80 \times |C|}$ and biases $\mathbb{B}^{(h)} \in \mathbb{R}^{80}$, $\mathbb{B}^{(o)} \in \mathbb{R}^{|C|}$ to produce a latent value $Z \in \mathbb{R}^{w \times |C|}$ as follows:

$$Z = max(0, E^{(4)}\mathbb{W}^{(h)} + \mathbb{B}^{(h)})\mathbb{W}^{(o)} + \mathbb{B}^{(o)} \tag{20}$$

Using this latent value, the classifier will produce an average probability distribution $P \in [0, 1]^{|C|}$ as follows:

$$P_j = \frac{1}{w} \sum_{i}^{w} \frac{e^{Z_{i,j}}}{\sum_{k}^{|C|} e^{Z_{i,k}}} \tag{21}$$

During inference, the classifier will return the most likely class as the member of C with the highest corresponding value in P: $C_{argmax(P)}$. During training, the classifier returns P which we use to evaluate the loss over the network.

References

1. Alioghli, A.A., Yıldırım Okay, F.: Enhancing multivariate time-series anomaly detection with positional encoding mechanisms in transformers. Journal of Supercomputing 81, 282 (2025)
2. Bar, R., Hajaj, C.: Simcse for encrypted traffic detection and zero-day attack detection. IEEE Access 10, 56952–56960 (2022)
3. Bar-on, M., Bezawada, B., Ray, I., Ray, I.: A small world–privacy preserving iot device-type fingerprinting with small datasets. In: Mosbah, M., Sèdes, F., Tawbi, N., Ahmed, T., Boulahia-Cuppens, N., Garcia-Alfaro, J. (eds.) Foundations and Practice of Security. pp. 104–122. Springer Nature Switzerland, Cham (2024)
4. Bin Sarhan, B., Altwaijry, N.: Insider threat detection using machine learning approach. Applied Sciences 13(1) (2023)
5. Dadkhah, S., Mahdikhani, H., Danso, P.K., Zohourian, A., Truong, K.A., Ghorbani, A.A.: Towards the development of a realistic multidimensional IoT profiling dataset. Submitted to the 19th Annual International Conference on Privacy, Security & Trust (PST 2022) (August 22–24 2022)
6. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers). pp. 4171–4186 (2019)
7. Dong, Y., Cordonnier, J.B., Loukas, A.: Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In: International conference on machine learning. pp. 2793–2803. PMLR (2021)
8. Hao, Q., Rong, Z.: Iottfid: An incremental iot device identification model based on traffic fingerprint. IEEE Access 11, 58679–58691 (2023)
9. Jero, S., Pacheco, M.L., Goldwasser, D., Nita-Rotaru, C.: Leveraging textual specifications for grammar-based fuzzing of network protocols. Proceedings of the AAAI Conference on Artificial Intelligence 33(01), 9478–9483 (Jul 2019)
10. Li, H., Zhang, L., Zhao, D., Xu, L., Li, X., Yang, S., Han, X.: A novel network protocol syntax extracting method for grammar-based fuzzing. Applied Sciences 14(6) (2024)
11. Li, J., Zhou, H., Wu, S., Luo, X., Wang, T., Zhan, X., Ma, X.: FOAP: Fine-Grained Open-World android app fingerprinting. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 1579–1596. USENIX Association, Boston, MA (Aug 2022)
12. Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., Lloret, J.: Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access 5, 18042–18050 (2017)
13. Lu., D.: Ustc-tfc2016. `https://github.com/yungshenglu/USTC-TFC2016` (2016)
14. Luo, Y., Chen, X., Ge, N., Feng, W., Lu, J.: Transformer-based device-type identification in heterogeneous iot traffic. IEEE Internet of Things Journal 10(6), 5050–5062 (2023)
15. Meng, X., Lin, C., Wang, Y., Zhang, Y.: Netgpt: Generative pretrained transformer for network traffic (2023)
16. Meng, X., Wang, Y., Ma, R., Luo, H., Li, X., Zhang, Y.: Packet representation learning for traffic classification. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. p. 3546–3554. KDD '22, Association for Computing Machinery, New York, NY, USA (2022)

17. Miskovic, S., Lee, G.M., Liao, Y., Baldi, M.: Appprint: Automatic fingerprinting of mobile applications in network traffic. pp. 57–69 (03 2015)
18. Msadek, N., Soua, R., Engel, T.: Iot device fingerprinting: Machine learning based encrypted traffic analysis. In: 2019 IEEE Wireless Communications and Networking Conference (WCNC). pp. 1–8 (2019)
19. Oh, S., Lee, M., Lee, H., Bertino, E., Kim, H.: Appsniffer: Towards robust mobile app fingerprinting against vpn. In: Proceedings of the ACM Web Conference 2023. p. 2318–2328. WWW '23, Association for Computing Machinery, New York, NY, USA (2023)
20. Pacheco, M.L., Hippel, M.v., Weintraub, B., Goldwasser, D., Nita-Rotaru, C.: Automated attack synthesis by extracting finite state machines from protocol specification documents. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 51–68 (2022)
21. Rodrawangpai, B., Daungjaiboon, W.: Improving text classification with transformers and layer normalization. Machine Learning with Applications 10, 100403 (2022)
22. Shaukat, K., Luo, S., Chen, S., Liu, D.: Cyber threat detection using machine learning techniques: A performance evaluation perspective. In: 2020 International Conference on Cyber Warfare and Security (ICCWS). pp. 1–6 (2020)
23. Sivanathan, A., Gharakheili, H.H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A., Sivaraman, V.: Classifying iot devices in smart environments using network traffic characteristics. IEEE Transactions on Mobile Computing 18(8), 1745–1759 (Aug 2019)
24. Taylor, V.F., Spolaor, R., Conti, M., Martinovic, I.: Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In: 2016 IEEE European Symposium on Security and Privacy (EuroSP). pp. 439–454 (2016)
25. Taylor, V.F., Spolaor, R., Conti, M., Martinovic, I.: Robust smartphone app identification via encrypted network traffic analysis. IEEE Transactions on Information Forensics and Security 13(1), 63–78 (2018)
26. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. p. 6000–6010. NIPS'17, Curran Associates Inc., Red Hook, NY, USA (2017)
27. Wang, J., Zhong, J., Li, J.: Iot-portrait: Automatically identifying iot devices via transformer with incremental learning. Future Internet 15(3) (2023)
28. Xu, J., Sun, X., Zhang, Z., Zhao, G., Lin, J.: Understanding and improving layer normalization. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
29. Yamansavascilar, B., Guvensan, M.A., Yavuz, A.G., Karsligil, M.E.: Application identification via network traffic classification. In: 2017 International Conference on Computing, Networking and Communications (ICNC). pp. 843–848 (2017)
30. Yin, Y., Xie, K., He, S., Li, Y., Wen, J., Diao, Z., Zhang, D., Xie, G.: Graphiot: Lightweight iot device detection based on graph classifiers and incremental learning. IEEE Transactions on Services Computing 17(6), 3758–3772 (2024)