

A Graph Representation Framework for Encrypted Network Traffic Classification[★]

Zulu Okonkwo^{a,*}, Ernest Foo^a, Zhe Hou^a, Qinyi Li^a and Zahra Jadidi^a

^aGriffith University, QLD, Australia

ARTICLE INFO

Keywords:

Network Traffic Classification
Neural Networks
Internet security
Encryption
Graphs

ABSTRACT

Network Traffic Classification (NTC) is crucial for ensuring internet security, but encryption presents significant challenges to this task. While Machine Learning (ML) and Deep Learning (DL) methods have shown promise, issues such as limited representativeness leading to sub-optimal generalizations and performance remain prevalent. These problems become more pronounced with advanced obfuscation, network security, and privacy technologies, indicating a need for improved model robustness. To address these issues, we focus on *feature extraction* and *representation* in NTC by leveraging the expressive power of graphs to represent network traffic at various granularity levels. By modeling network traffic as interconnected graphs, we can analyze both flow-level and packet-level data. Our graph representation method for encrypted NTC effectively preserves crucial information despite encryption and obfuscation. We enhance the robustness of our approach by using cosine similarity to exploit correlations between encrypted network flows and packets, defining relationships between abstract entities. This graph structure enables the creation of structural embeddings that accurately define network traffic across different encryption levels. Our end-to-end process demonstrates significant improvements where traditional NTC methods struggle, such as in Tor classification, which employs anonymization to further obfuscate traffic. Our packet-level classification approach consistently outperforms existing methods, achieving accuracies exceeding 96%.

1. Introduction


The surge in cyber-attacks has heightened interest in Network Traffic Classification (NTC), which categorizes network traffic into various classes. Encryption has complicated this task, prompting the adoption of deep learning (DL) to analyze patterns, characteristics, and contents of encrypted traffic. Before DL, NTC methods ranged from port-based classification to statistical techniques [16, 35, 26, 44]. Recent and previous research [27, 24, 33] leveraged statistical features due to the non-contextual nature of encrypted network traffic. This led to a new wave of Machine Learning based practices and now Deep Learning methods [38, 9, 39, 49, 35, 26]. Early statistical approaches relied on hand-crafted flow-based features, which posed challenges for real-world classification due to disparities between long and short flows. Recent classification strategies also use network metadata, such as packet length and inter-arrival time. Understanding the differences between network metadata and actual encrypted traffic is crucial for effective NTC.

Network Traffic Classification (NTC) is generally regarded as a classification problem in the field of ML and DL, (see Figure 1). Effective NTC requires structured data representation for ML/DL analysis. Traditional approaches [41, 42, 35, 26] have transformed network traffic into grey-scale or colored images, inspired by DL's success in image classification. While these methods yield good results, they often capture network characteristics rather than application behavior, leading to poor generalization. Given the time-series nature of network traffic, temporal features

are also vital for classification. Recurrent Neural Networks (RNNs), particularly Long Short Term Memory (LSTM) networks [21, 47], have been used to extract these features, inspired by advances in natural language processing (NLP). These methods analyze raw traffic, simplifying feature selection and showing promising results. Combining DL methods [21, 40, 1] has further improved classifier performance. However, the unstructured nature of encrypted traffic still poses challenges, especially for DL analysis of NTC. CNN-based methods [41, 42, 35, 26] often involve manual feature selection and extraction, emphasizing network behavior rather than application behavior, which results in poor generalization. LSTM approaches analyze all traffic similarly, leading to sub-optimal performance under enhanced encryption. Combining DL methods increases network complexity and the risk of overfitting. This critical challenge underscores the need for novel approaches in representing encrypted traffic. Hence, our research question emerges: *how can encrypted network traffic be represented to enhance the accuracy of detection, identification, and classification?* Addressing this question is essential for advancing the field of NTC, particularly in the face of evolving encryption techniques.

Recent works, such as [51, 11, 23] have demonstrated the effectiveness of processing both payload and header data for classification tasks. These studies reported remarkable performance, despite the inherent lack of pattern in encrypted payloads. Inspired by these results, we incorporate payload data in our experiments to evaluate its potential contribution to classification accuracy. This approach aims to determine whether the inclusion of payload data can indeed enhance classification performance, despite the challenges posed by encryption.

*Corresponding author

 zulu.okonkwo@griffithuni.edu.au (Z. Okonkwo)

ORCID(s):

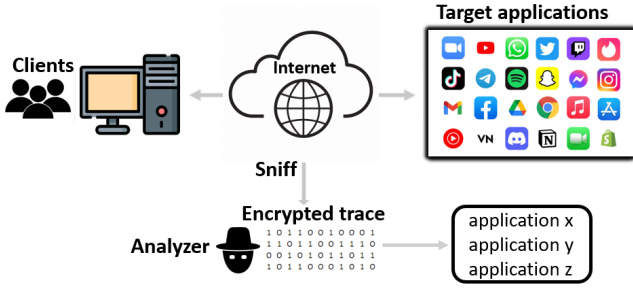


Figure 1: A Typical Encrypted Network Traffic Classification Model Task

Geometric deep learning (GDL) [2] have been effectively employed to unstructured data, enabling node, link, or entire graph prediction. By examining graph components and their topological structures, GNNs can handle complex classification tasks. The application of GNNs to network traffic classification (NTC) has shown promising advancements [26, 11, 30, 52, 51, 3]. Despite recent advancements, significant challenges persist. Just as distinct features aid in image recognition, graphs must encapsulate the expressive nature of entities before deep learning computations. Enhancing this capability requires careful design of graph structural properties, focusing on feature extraction and representation methodologies. Studies by Huoh et al. (2021) and Okonkwo et al. (2023) applied GNNs to NTC but fell short in leveraging the full potential of graph expressiveness, producing simplistic graphs with minimal topological distinctions [11, 26], resulting in path graphs with no structural distinction. This approach is inadequate for granular, multi-classification tasks in encrypted NTC. To advance the field, it is crucial to develop methods that enhance the expressive nature of graph representations, allowing for more effective deep learning computations and improved classification accuracy in encrypted NTC. This calls for innovative approaches in feature extraction and graph generation, addressing the limitations of current methodologies and harnessing the full potential of GNNs in NTC.

This paper addresses representation for encrypted NTC using graph-based analysis at the flow and packet levels to create interconnected graphs. We introduce two types of edges to enrich graph structure: chronological edges preserving time series data, and similarity edges facilitating communication between similar nodes while enhancing topological distinctions. Using cosine similarity, we establish secondary node relationships by creating edges when a similarity threshold is met. This promotes interaction among similar nodes, generating structural embeddings essential for graph definition. We evaluate our method on four public datasets using GNN variants, emphasizing models that retain higher-order information for effective graph classification. Our Contributions:

1. We propose novel methods to represent network traffic as graphs, constructing interconnected graphs at both

flow and packet levels by analyzing sequences of network packets and streams of bytes, respectively.

2. We design an effective GNN-based classifier for network graphs. We test various Graph Neural Network architectures, focusing on whole graph classification, defining a baseline model, and building customizations. Our experiments identify an architecture that excels in both flow and packet-level classification.
3. We extensively evaluate our methodology, conducting comparative analyses with related work and providing experimental explanations. These explanations reveal how our network graph structures enhance the classification process by preserving crucial structural information for encrypted network traffic classification.

This paper is organized as follows: Section II defines the preliminaries of network traffic and our adopted Graph Neural Network (GNN). Section III introduces our approach to defining and creating network graphs, as well as our classification model. Section IV details the dataset used and provides an in-depth experimental analysis. Section V discusses key observations from the novelty of our approach. Section VI reviews related literature, and Section VII concludes the work.

2. Preliminaries

This section introduces network properties and GNNs.

2.1. Encrypted Network Traffic Classification

In packet-switched networks, a flow is defined as a sequence of packets sharing common parameters: src/dest IP, src/dest port, and protocol (the 5-tuple). For simplicity, we reduce traffic flows to time windows. We segment traffic flows T_F into smaller subsets, referred to as flows, based on time windows of n seconds. Let T_F be a traffic flow, using a time window of n seconds we split T_F into separate fragments of flows T_{f_i} , where $i \in \{1, 2, \dots, m\}$ and $T_{f_i} \in T_F$ and we overload the set notations for lists. Mathematically,

$$T_F = \{T_{f_1}, T_{f_2}, \dots, T_{f_m}\}, \quad (1)$$

where T_{f_1}, T_{f_2} etc, refer to fragments of traffic flows returned after splitting T_F . Network packets are the basic units of data transferred over a network, combining to form larger messages and network flows. A flow T_{f_i} , can be broken down into individual packets $P_{i_1}, P_{i_2}, \dots, P_{i_m}$. Let

$$T_{f_i} = \{P_{i_1}, P_{i_2}, \dots, P_{i_m}\}, \quad (2)$$

where P_{i_m} denotes an individual packet in the flow, m represents the packet sequence and the length of the flow in this case. i is an identifier that ties the packet to a flow, like a label. Basically, P_{i_m} is the m -th packet of the i -th traffic flow. An encrypted packet is a byte stream capped at a maximum transmission unit of 1500 bytes. For packet-level classification, we analyze the byte sequence of packets.

Given a flow T_{fi} and a packet P_{i_m} from this flow, we can break down P_{i_m} into its byte stream. The byte $b_{i_m\ell}$ represents the ℓ -th byte of the m -th packet in the i -th traffic flow. Let

$$P_{i_m} = \{b_{i_m1}, b_{i_m2}, \dots, b_{i_m\ell}\}, \quad (3)$$

where ℓ represents the byte position, m is an identifier that ties the byte to its packet, and i is an identifier that ties the byte to its traffic flow in $b_{i_m\ell}$.

Encrypted traffic classification categorizes network traffic by applications, services, protocols, etc. This paper focuses on classifying applications and services (e.g., video, audio, chat, file transfer). We propose two methodologies for representing network traffic as graphs. The classification tasks involve training a model on labeled sample graphs and predicting the labels of an unseen sample set. The trained neural network maps a sample of unlabeled flow graphs T_{fx} or packet graphs P_{ix} where $x \in \{1, 2, \dots, n\}$ to labels $L_s = 0, 1, 2, \dots, N - 1$.

2.2. GNN Preliminaries

Graph Neural Networks (GNNs) perform optimizable transformations on graph attributes while preserving permutation invariance and equivariance. GNNs typically employ a message-passing mechanism where neighborhood information is aggregated to form new node representations. In the context of NTC, we focus on graph classification, aiming to predict the properties of entire graphs. A simple graph $G = (V, E)$ consists of a set nodes $V(G)$ and edges $E(G)$. Nodes are initialized with an attribute matrix representing the feature vector X_v for $v \in V$. Let \mathcal{G} be set of graphs such that $\{G_1, \dots, G_N\} \subseteq \mathcal{G}$ and \mathcal{L} be a set of labels for \mathcal{G} such that $\{l_1, \dots, l_N\} \subseteq \mathcal{L}$ the graph classification task aims to learn a representation vector h_{G_N} which is used to predict the label l_N of the entire graph G_N .

GNN layers aggregate neighborhood information to update node representations. The basic propagation rule is:

$$\mathbf{h}^{(l+1)} = \sigma(\mathbf{A}\mathbf{h}^l, \mathbf{w}^l), \quad (4)$$

where $\mathbf{h}^{(l+1)}$ is the feature representation at layer “ $(l + 1)$ ”, \mathbf{A} is the adjacency matrix defining graph connectivity, \mathbf{h}^l is the feature representation at layer l and \mathbf{w}^l is the trainable weight parameter at layer l . The non-linear function σ is used during training. Variants of GNNs modify this propagation rule 4 to enhance performance.

In this paper, we adopt the methodology proposed by Morris et al. [25], which performs message passing between sub-graph structures rather than individual nodes, enhancing the capture of structural information in graphs for encrypted NTC.

As stated earlier a graph G is a pair (V, E) , and $E \in \{(u, v) \mid (u, v) \in V^2 \ \& \ u \neq v\}$. The neighbourhood $N(v)$ of a node v in $V(G)$ is defined as $N(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. Let $V(G)^k$ denote a set of all k -elements subsets of $V(G)$ containing k nodes. For $s \in V(G)^k$ and $s = \{s_1, \dots, s_k\}$, the neighbourhood of s is defined as

$$N(s) = \{t \in V(G)^k \text{ s.t. } |s \cap t| = k - 1\}, \quad (5)$$

The local neighbourhood $N_L(s)$ consists of all elements in the set $t \in N(s)$ such that $(u, v) \in E(G)$ for unique $u \in s \setminus t$ and unique $v \in t \setminus s$. To clarify Equation 5: For a given integer k , we consider all sets of k vertices from a graph G , denoted as $V(G)^k$. Let $s \in V(G)^k$ be a set of k vertices, represented as $\{s_1, \dots, s_k\}$. The neighborhood of s , $N(s)$, is defined as the collection of sets $t \in V(G)^k$ where $|s \cap t| = k - 1$. Essentially, $N(s)$ contains sets that differ from s by exactly one vertex. The local neighborhood, $N_L(s)$ consists of all $t \in N(s)$ that has a unique edge $(u, v) \in E(G)$ such that $u \in s \setminus t$ and $v \in t \setminus s$. For a k -set t to be in the local neighborhood $N_L(s)$ there must exist:

- A unique vertex $u \in s \setminus t$ (i.e., a vertex that is in s but not in t)
- A unique vertex $v \in t \setminus s$ (i.e., a vertex that is in t but not in s)
- an edge $(u, v) \in E(G)$ (i.e., there must be an edge in the graph G connecting u and v)

Consider the graph G with vertices $V(G) = \{v_1, v_2, v_3, v_4\}$, and edges $E(G) = \{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}$.

Choosing $k = 2$ yields,

$$V(G)^k = \{(v_1, v_2), (v_1, v_3), (v_1, v_4), (v_2, v_3), (v_2, v_4), (v_3, v_4)\}$$

Selecting s from $V(G)^k$ as $s = \{v_1, v_2\}$

From Eq 5, $N(s) = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_2, v_3\}, \{v_2, v_4\}\}$

The local Neighbourhood $N_L(s) = \{v_1, v_3\}$ because it satisfies the three conditions.

The local propagation of feature vectors at layer ℓ is given by:

$$f_{k,L}^{(\ell)}(s) = \sigma \left(\mathbf{W}_1^{(\ell)} \cdot f_{k,L}^{(\ell-1)}(s) + \mathbf{W}_2^{(\ell)} \cdot \sum_{j \in N_L(s)} f_{k,L}^{(\ell-1)}(j) \right), \quad (6)$$

where $f_{k,L}^{(\ell-1)}(s)$ is the feature vector of the set of nodes s at layer $\ell - 1$, and $\sum_{j \in N_L(s)} f_{k,L}^{(\ell-1)}(j)$ represents the aggregation of the local neighborhood set of s . For simplicity, this can be denoted as:

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j, \quad (7)$$

where \mathbf{W}_1 and \mathbf{W}_2 are trainable weight parameters, \mathbf{x}_i is the feature vector of node i , $\sum_{j \in \mathcal{N}(i)}$ is the aggregation of the local neighborhoods of i , $e_{j,i}$ is the edge weight, and \mathbf{x}_j is the feature vector of the neighbors. A non-linear function like ReLU, Tanh, or Sigmoid is used during computation.

3. Our Approach

In this section, we detail our network graph generation process at both flow and packet levels, providing formal and logical explanations for our edge creation process, which enhances the structural richness of our graphs. We also introduce our classification model, which employs a higher-order Graph Neural Network (GNN) to leverage graph structure during computation. Figure 2 shows our end-to-end model.

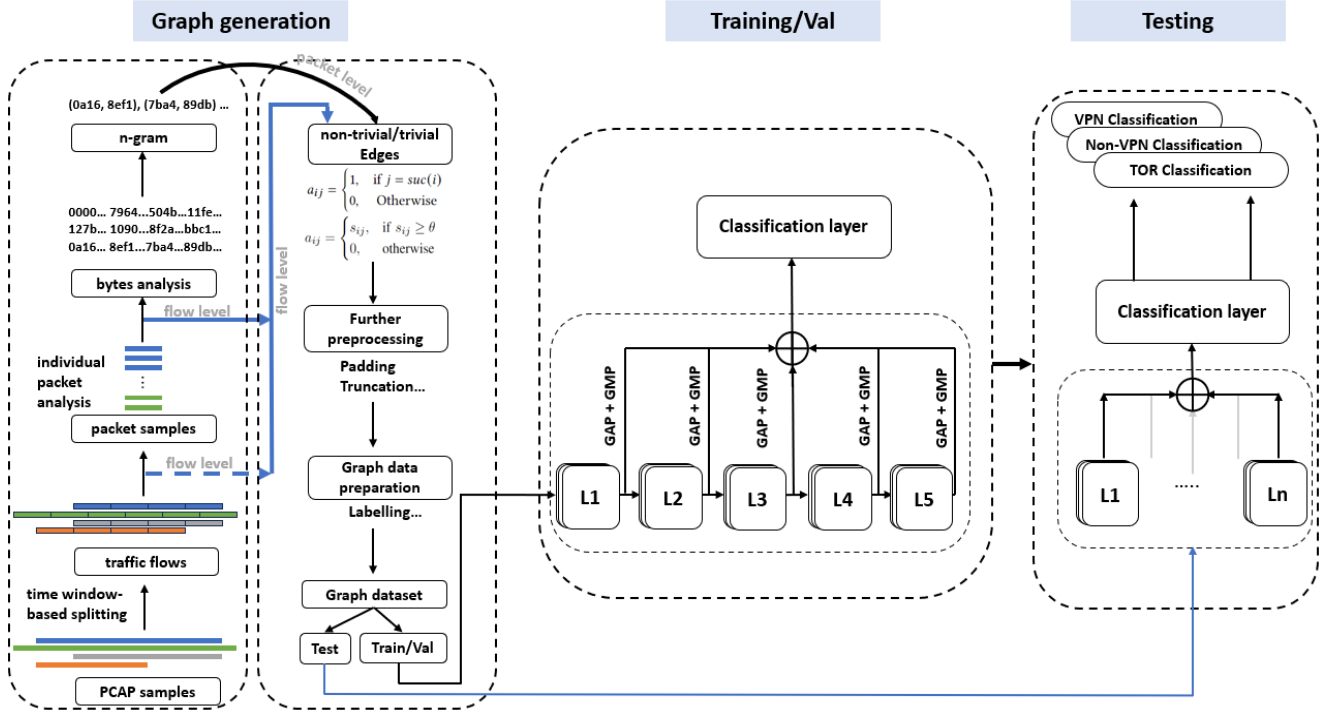


Figure 2: Model Overview. Graph generation starts with processing the PCAP samples then graphs are created based on the specified task. After splitting PCAP samples into flows, individual packets of flows are analysed. For flow classification, the edge creation process comes next. For packet-level classification, the byte stream of individual packets is analysed before edge creation. Padding is applied at the byte-level to ensure uniformity of data. Truncation is applied at the packet-level to ensure graphs are not dense. The graph data is labelled, processed and stored in memory. Training and testing is described in Section 4.2.2.

3.1. Flow-level Traffic Graph Generation

The flow-level graph generation process involves creating graphs from traffic flows T_{fx} , with nodes representing packets P_{ix} , within these flows. As discussed in Section 2.1, defining an optimal time window is crucial. Peng et al. [31] identified 5-7 packets as optimal for early traffic identification. Establishing a TLS connection requires up to six packets, so we extend this threshold to 10 for regular TLS traffic and 20 for more complex traffic, such as Tor. To determine the time window, we need at least 20 packets per window. Given P as the total number of packets and D as the total duration of the traffic capture, the average packet rate R is $\frac{P}{D}$. The time T to capture 20 packets is calculated as:

$$T \geq \frac{20}{R} = \frac{20D}{P}, \quad (8)$$

The variables P and D are obtained from the traffic capture and not calculated. To effectively parse traffic samples, we utilized "splitcap" a tool that enables the selection of a time window for splitting pcap samples. Since we exclude handshake packets and concentrate on application packets, a carefully selected time window is crucial to ensure an adequate number of packets remains post-elimination of unwanted packets. After calculation we observed that different traffic samples have varying times and the need to standardize the time became important. This varying calculated time windows illustrates the complexity and potential tedium

of manual window selection. Consequently, we propose a standardized time window of 15 seconds, which was determined after averaging all the observed time distribution. The time 15 seconds is optimal across all datasets mitigating the complexities associated with manual adjustments. From Equation 8 we determined that a 15-second window size is optimal for all flow-level experiments. To define connectivity between packets, we introduce two edge types, namely non-trivial and trivial edges.

3.1.1. Non-trivial edges

The non-trivial edges define the time series relationship between packets, therefore preserving the chronological arrangement of network packets. Creating a relationship between nodes v_1, v_2, \dots, v_n such that the edges are $\{(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)\}$. This edge type generates a network path graph drawn such that all of its vertices lie on a single straight line. While this edge type adds no topological distinction to the graphs, it remains important to the overall design as communication between preceding and succeeding nodes is preserved. Huoh et al. [11] and [26] demonstrated the importance of this edge type for a similar task. Given nodes i, j , we define the edges through the adjacency matrix a of nodes i, j as

$$a_{ij} = \begin{cases} 1, & \text{if } j = \text{suc}(i) \\ 0, & \text{Otherwise.} \end{cases} \quad (9)$$

That is, an edge is created only if node j is a successor of node i in a packet sequence. We denote an edge creation with the numeral 1 and otherwise with 0. This edge ensures the sequential order of packets in a flow is maintained during processing and analysis. The node attributes are the packet contents, converted from raw byte to decimal form, normalized, and padded to a vector length of 1500 bytes which is the maximum transmission unit (MTU). For effective graph classification, structural distinction is essential. While our non-trivial edges create connected graphs, they provide no topographical distinction.

The key to graph classification lies in identifying a single defining property for the entire graph. In ML and DL, parameter size directly correlates with performance. In GNN tasks, enhancing node communication is crucial for effective graph classification. However, this must be done systematically to prevent over-smoothing. To create graphs with rich structures we therefore propose defining a new edge type that uniquely identifies various traffic types while enhancing node communication without causing over-smoothing.

3.1.2. Trivial edges

Encrypted payloads appear meaningless until decrypted. Attempting to analyze these encrypted payloads in their raw, encrypted state invariably leads to representations that lack meaningful insight. To address this, we employ a method of transforming packet data as shown in Figure 3, transforming them into data points that can be analyzed more effectively. Each vector represents a packet's data content, and by comparing these vectors, we can infer how similar or dissimilar different packets are, despite the encryption.

	Raw Packets (hex)	Decimal representation	Normalized
Pac1	00, 00, 79, 64, 50...	0, 0, 121, 100, 80...	0, 0, 0.47, 0.39, 0.31...
Pac2	12, 7b, 10, 90, 8f...	18, 123, 16, 144, 143...	0.07, 0.48, 0.06, 0.57, 0.56...
Pac3	0a, 16, 8e, f1, 7b...	10, 22, 142, 241, 123...	0.04, 0.09, 0.57, 0.95, 0.48...

Figure 3: Raw packets are converted from their hexadecimal representation to decimal, then normalized. All our analyses are performed on the normalized data

Upon transforming these packets, a notable pattern emerges in the distribution of non-zero vectors relative to zero vectors. Specifically, we observe that larger payloads tend to contain a higher proportion of non-zero vectors. This observation is intuitive, given that larger payloads inherently have more data and therefore more variability. In the context of information theory and entropy, a zero-vector signifies no variation or data, representing a complete absence of uncertainty and information. Our findings lead us to a significant insight: *larger encrypted payloads tend to have more information because they contain less zero-vectors than smaller payloads*. We therefore streamline our task to analyse how correlated these non-zero vectors are.

In the pursuit of analyzing the correlation among vectors, especially within the domain of encrypted data, it becomes imperative to recognize that traditional correlation measures

often fall short. Random variables representing encrypted data typically yield correlation results indicative of no meaningful relationship. Thus, conventional statistical measures of correlation are not suitable for our purpose. We redefine the concept of correlation to examine the similarity of transformed network packets by analyzing how encryption perturbs different similarities between packets in a flow. For traffic classification tasks such as fingerprinting, properties that remain consistent over time are usually preferred for classification due to their stability and uniqueness. We therefore aim to define a similarity metric that is robust enough to withstand variations introduced by encryption.

In our study, we randomly selected 120 flows each from four applications in the Ieee-im dataset [5]. Each flow is processed by converting packets to vectors (see Figure 3), removing handshake packets, retaining only application data packets and then removing packet headers. As stated in Equation 3, a packet is made up of a string of bytes. A byte can represent a number between 0 - 255 in decimal. This conversion is applied to every packet resulting in a string of integers. We then normalise the values to fall between zero and one and use this as our vectorized payload representation. To analyse the correlation between the vectors we select five distinct similarity metric: Cosine Similarity, Euclidean Distance, Jaccard Index, Manhattan Distance, and Pearson Correlation Coefficient. These metrics were carefully selected due to their extensive application in natural language processing (NLP) tasks, such as clustering, and classification. For each flow, we compute the similarity of packets, averaging the upper triangle of the resulting similarity matrix. To ensure a fair comparison, values are normalized to fall between 0 and 1. From Figure 4, it is evident that cosine similarity is consistently the least perturbed across all experiments. To substantiate this observation, we calculated the standard deviation (SD) of the similarity values, which quantifies the amount of variation or dispersion. A lower SD indicates values closer to the mean, suggesting less variation or perturbation, while a higher SD indicates more spread-out values, suggesting greater variation. Cosine similarity exhibited the lowest SD across all plots: 0.0906 for plot 4a, 0.1031 for plot 4b, 0.0569 for plot 4c, and 0.0464 for plot 4d.

Our analysis thus pivots towards examining the geometric properties of vectors. Specifically, we focus on non-zero vectors and evaluate their correlation by examining the cosine of the angle between them. This method, provides a more nuanced understanding of the relationships among vectors in high-dimensional spaces. By measuring the cosine of the angle, we can effectively quantify the directional alignment of the vectors, offering insights into their intrinsic geometric relationships. This approach aligns well with the broader objective of uncovering structures within encrypted data.

To avoid the pitfalls of generating mathematically complete graphs with indistinguishable structure, we define a similarity threshold for edge creation. By establishing this threshold, we ensure that only sufficiently similar nodes are connected, thereby creating edges that facilitate meaningful

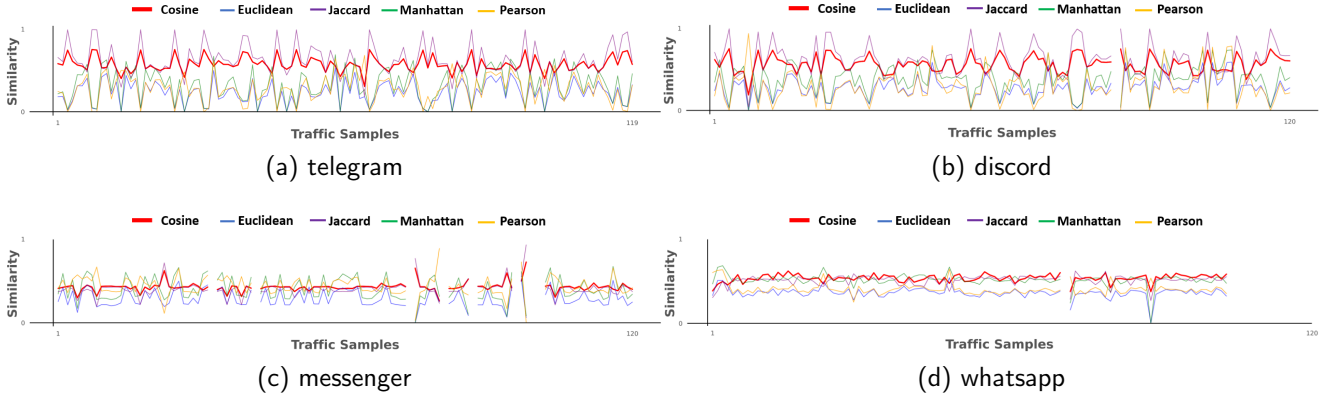


Figure 4: Comparison of Similarity Metrics Across Encrypted Traffic Flows: For each application, 120 random traffic flows are analyzed to understand how encryption impacts flow similarity. The graphs display the behavior of five different similarity metrics: Cosine, Euclidean, Jaccard, Manhattan, and Pearson. Across all applications, Cosine similarity exhibits the least perturbation. This is followed by Jaccard, Euclidean, Pearson, and Manhattan similarities, in descending order of stability. The consistency of Cosine similarity is evident, with the lowest standard deviation values, making it our go to choice.

communication while managing computational efficiency. These edges are weighted by their similarity values and are termed “trivial edges”. The designation “trivial” reflects their artificial introduction to enhance structural distinction and communication within the graph, which is crucial for entire graph classification. We rely on descriptive statistics, specifically *percentile*, a high percentile value ≥ 85 is chosen to establish this threshold. Percentiles help to adaptively set the threshold based on the distribution of the similarity scores rather than choosing an arbitrary value.

Given a set of node feature vectors, compute the pairwise cosine similarity scores. Let $S = \{s_{ij} \mid i, j \in V, i \neq j\}$ represent the set of all cosine similarity scores between node pairs, where V is the set of nodes. Select a high percentile value to determine the threshold θ . Formally, the selected threshold θ is:

$$\theta = \begin{cases} \theta_{90}, & \text{if } \theta_{85} = \theta_{90} = \theta_{95} \\ \max(\theta_{85}, \theta_{90}, \theta_{95}), & \text{otherwise} \end{cases}$$

Here we compare the calculated percentile values θ_{85} , θ_{90} , and θ_{95} for their *representativeness*. If these values are equal, we select the median percentile value. If not, we choose the largest value among them. Using the threshold θ , we define the adjacency matrix a as follows:

$$a_{ij} = \begin{cases} s_{ij}, & \text{if } s_{ij} \geq \theta \\ 0, & \text{otherwise} \end{cases} \quad (10)$$

Here, a_{ij} represents the edge weight between nodes i and j . The edge weight a_{ij} is assigned the cosine similarity score s_{ij} if the score is greater than or equal to the threshold θ ; otherwise, no connection edge is created between the nodes. Our calculated values produced thresholds of 0.7 for vpn, nonvpn, ieee-im and uisc datasets. The calculated threshold for Tor is 0.95. Figure 5 and 6 shows flow level graphs for some selected applications of the Ieee-im dataset with some similar structures highlighted.

The Algorithms, define our graph creation process for the flow traffic graphs. For brevity we assume the pcap files are past the initial Pre-processing as described in Section 4.2.1 which involves: splitting into traffic flows, deletion of unwanted flows and packets, IP address and port masking, Ethernet header removal and TCP/UDP header padding. Hence, We have a folder of the initially processed traffic flows.

Algorithm 1 Flow-level features

Input: Folder containing processed pcap flows

Output: 2D array X of dimension $n \times 1502$

```

1: Initialize an empty 2D array  $X$  of dim  $n \times 1502$ 
2: for file in Input do
3:   for packet in file do  $\triangleright$  Extract packets from flow
4:     Convert packet to raw bytes
5:     Convert bytes to decimal values
6:     Normalize to  $[0, 1]$ 
7:     if len(packet) < 1500 then
8:       Pad to length 1500
9:     end if
10:    Extract packet_label and flow_number
11:    Append packet_data, label, and flow_number to
     $X$ 
12:   end for
13: end for
14: return  $X$ 

```

Algorithm 1 operates on the folder containing the split traffic flows then returns as 2D array of dimension $n \times 1502$. Every row represents a packet with its data in columns 1 – 1500. The last two columns “packet_label” and “flow_number” indexed as 1501 and 1502 represents the class label and the traffic flow each packet belongs to respectively.

Algorithm 2 returns the threshold for initializing the trivial edges by operating on a modified version of array X . The array X' represents the packet payload vector with

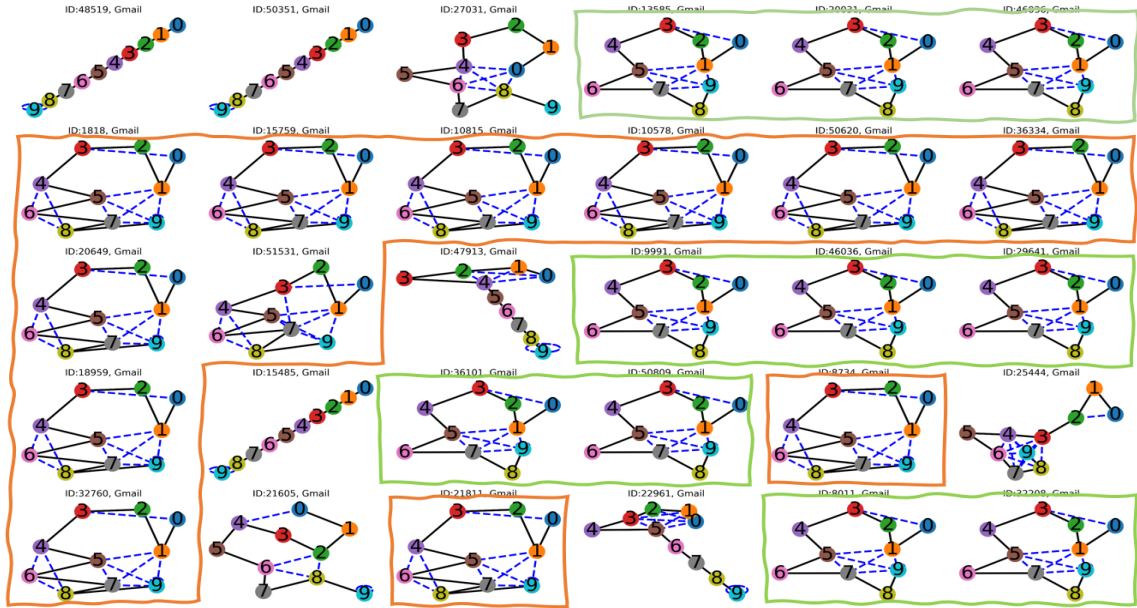


Figure 5: The figure presents a series of flow-level graphs generated using our method for the Gmail application of the IEEE-im dataset [5]. Each graph represents the structural patterns of network flows within Gmail application. The two edge types, non-trivial edges are represented as solid black lines and trivial edges are represented as dashed blue lines. The two highlighted formations (in colours orange and green) indicate multiple underlying structures unique to the application class. Below we display more applications with distinct structures.

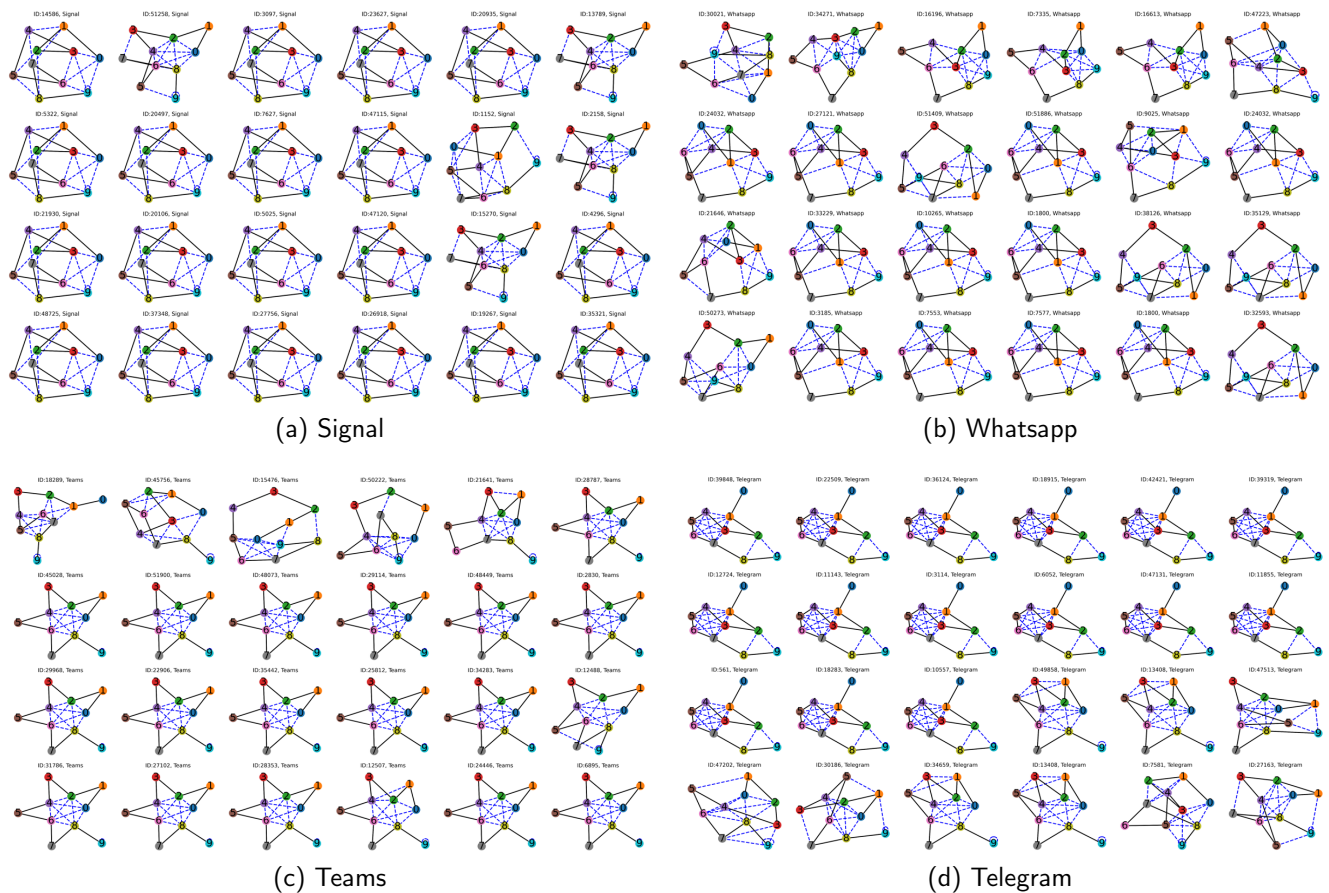


Figure 6: The graph structures represent unique flow-level graphs for four application classes of the IEEE-im dataset. The images are intentionally kept small to focus on the distinctive structural patterns of each class.

dimension $n \times 1500$, where each row represents a packet. Line 1 drops the columns added by line 11 of Algorithm 1 to ensure that only the actual vectorized payloads are considered. Line 2 computes the cosine similarity S of all the rows in X' . This results in a $n \times n$ matrix that is symmetric (i.e., the upper and lower triangular elements are the same, and the diagonal elements are 1 because they represent the similarity of a vector with itself). Line 3 extracts all the similarity values focusing only on the upper triangle (since both upper and lower are the same). Line 4, 5, 6 calculates the 85th, 90th and 95th percentiles, ensuring that the values are rounded up to two decimal places. Line 7, 8, 9 and 10 checks the conditions for threshold selection as stated in Section 3.1.2. If the values are equal then the middle value which is the 90th percentile, is chosen. If this is not the case, the maximum percentile which is the 95th is chosen. Line 11 ends the conditional statement and line 12 returns the threshold value.

Algorithm 3 computes trivial edges and their corresponding weights based on cosine similarity. Given an input array X of dimensions $n \times 1502$, the algorithm produces a 2D array Y of dimensions $m \times 3$ with columns representing the source node, destination node, and weight. Initially, the last two columns of X are dropped to obtain X' , and a similarity threshold θ is computed. The outer loop iterates over the packets in steps of 10, the middle loop iterates over the next 9 packets, and the inner loop compares each pair of packets within this window. Cosine similarity is calculated for each pair, and if the similarity exceeds the threshold, the edge is recorded in Y .

Algorithm 4 computes non-trivial edges and their corresponding weights. Given an input array X of dimensions $n \times 1502$, the algorithm produces a 2D array Y of dimensions $m \times 3$ with columns representing the source node, destination node, and weight. Initially, the last two columns of X are dropped to obtain X' . The outer loop iterates over the packets in steps of 10. For each packet i , the middle loop compares

Algorithm 2 Similarity threshold

Input: Array X of dimension $n \times 1502$
Output: Threshold value θ

```

1:  $X' \leftarrow$  drop last two columns of  $X$ 
2: Compute cosine similarity matrix  $S$  from  $X'$ 
3: Extract upper triangular part of  $S$  excluding the diagonal
   as similarity_scores
4:  $\theta_{85} \leftarrow$  round(percentile(similarity_scores, 85))
5:  $\theta_{90} \leftarrow$  round(percentile(similarity_scores, 90))
6:  $\theta_{95} \leftarrow$  round(percentile(similarity_scores, 95))  $\triangleright$  in
   line 4,5 and 6, the values are rounded up to two decimal
   places.
7: if  $\theta_{85} = \theta_{90} = \theta_{95}$  then
8:    $\theta \leftarrow \theta_{90}$ 
9: else
10:   $\theta \leftarrow \max(\theta_{85}, \theta_{90}, \theta_{95})$ 
11: end if
12: return  $\theta$ 

```

Algorithm 3 Trivial edges

Input: Array X of dimension $n \times 1502$
Output: 2D array Y of dimension $m \times 3$ with headers: $src_node, dest_node, weight$

```

1:  $X' \leftarrow$  drop last two columns of  $X$ 
2:  $\theta \leftarrow$  Similarity threshold( $X'$ )
3: Initialize an empty 2D array  $Y$  of dim  $m \times 3$ 
4: for  $i = 1$  to  $n$  by 10 do
5:   for  $j = i$  to  $i+9$  do
6:     for  $k = j+1$  to  $i+9$  do
7:        $s_{ij} \leftarrow$  cosine_similarity( $X'[i], X'[j]$ )
8:       if  $s_{ij} \geq \theta$  then
9:         Append  $[i, j, s_{ij}]$  to  $Y$ 
10:      end if
11:    end for
12:  end for
13: end for
14: return  $Y$ 

```

Algorithm 4 Non-trivial edges

Input: Data array X of dimension $n \times 1502$
Output: 2D array Y of dimension $m \times 3$ with headers: $src_node, dest_node, weight$

```

1:  $X' \leftarrow$  drop last two columns of  $X$ 
2: Initialize an empty 2D array  $Y$  of dim  $m \times 3$ 
3: for  $i = 1$  to  $n$  by 10 do
4:   for  $j = i$  to  $i+9$  do
5:      $k \leftarrow j + 1$ 
6:     if  $k \leq i + 10$  then
7:       Append  $[j, k, 1]$  to  $Y$ 
8:     end if
9:   end for
10: end for
11: return  $Y$ 

```

it with the next packet within the group of 10, ensuring that the destination index k does not exceed the boundary of the current group. The resulting pairs and their associated weights are then appended to Y .

Algorithm 5 prepares the final files used to create traffic flow graphs from a folder containing pcap files, producing four arrays: V (node embeddings), E (edges with source node, destination node, and weight), N (node index and flow number), and F (flow number and label). The algorithm begins by initializing all packets into a 2D array X . It then extracts node embeddings V by dropping the last two columns of X . Trivial and non-trivial edges with weights are computed from V , resulting in the array E . The arrays N and F are initialized and populated with node indices, flow numbers, and labels by iterating through the rows of X and unique flow numbers. Finally, the algorithm returns the four arrays, which are used to create the traffic flow graphs.

3.2. Packet-level Traffic Graph Generation

The packet-level traffic graph generation aims to generate graphs from individual network packets P_{ix}, \dots . We utilise

Algorithm 5 Flow-Level graph files

Input: Folder containing pcap files
Output: Four arrays: V , E , N , F
 V ($m \times 1500$) \triangleright node embeddings
 E ($k \times 3$) \triangleright src_node, dest_node, weight
 N ($m \times 2$) \triangleright node_index, flow_number
 F ($f \times 2$) \triangleright flow_number, label

- 1: $X \leftarrow$ Flow-level features(Input)
- 2: $V \leftarrow$ drop last two columns of X
- 3: $E \leftarrow$ Trivial edges(V)
- 4: $E \leftarrow$ Non-trivial edges(V)
- 5: $N \leftarrow$ Initialize an empty 2D array of dim $m \times 2$
- 6: $F \leftarrow$ Initialize an empty 2D array of dim $f \times 2$
- 7: **for** each row in X **do**
- 8: Append (node_index, flow_number) to N
- 9: **end for**
- 10: **for** each unique flow_number in X **do**
- 11: Extract corresponding label
- 12: Append (flow_number, label) to F
- 13: **end for**
- 14: **return** V , E , N , F

the bytes sequence $b_{i_m} \dots$ to model nodes. Having exploited vectors directional orientation by similarity calculation for our flow level graphs in Section 3.1, we extend the same concept to create packet level graphs, but this time to bit sequences in a packet at a time. Sengunta et al. [34] used *bit sequence* based features to exploit the differences in the randomness of data generated by different applications. By

Algorithm 6 Flow to packet-level features

Input: Folder containing processed pcap flows
Output: 2D array Y of dimension $10n \times 152$

- 1: Initialize an empty 2D array Y of dimensions $10n \times 152$
- 2: $X \leftarrow$ Flow-level features(Input)
- 3: $X' \leftarrow$ drop last two columns of X
- 4: Initialize $row_index \leftarrow 0$
- 5: **for** each row in X' **do do**
- 6: **for** $i = 0$ to 9 **do do**
- 7: Extract columns $i \times 150$ to $(i + 1) \times 150 - 1$ from the row and assign to $Y[row_index, 0 : 149]$
- 8: Assign packet label from X to $Y[row_index, 150]$
- 9: Assign flow number from X to $Y[row_index, 151]$
- 10: $row_index \leftarrow row_index + 1$
- 11: **end for**
- 12: **end for**
- 13: **return** Y

extracting features based on the correlation and divergence of the Fast Fourier Transforms (FFT) of consecutive packets, they were able to distinguish Android-based apps from one another. They posited that the difference was attributed to the dissimilarities in encryption implemented by different applications. He et al. [10] treated the payload of network

packets as a language-like string for introducing NLP processing. To initialize tokenization, they took bit sequences of the raw payload as pairs of bytes for basic character units to generate bi-gram strings for their NLP model. This bit and byte level information mining by both methods [34, 10] shows that encrypted bits of data when processed can be used for classification tasks. The bit-byte sequence analysis of network traffic information is analogous to the n-gram method used frequently in NLP, where a collection of n items in a text document is analysed in a successive manner. By considering a sequence of bytes at a time, we can model nodes with rich embeddings for every n-gram. We convert every packet to its vector representation as in Figure 3 and pad packets with zero up to the MTU size (1500bytes) for data consistency. To increase the representative values of nodes, we use non-overlapping n-grams with the value of 150 for n . Our choice of non-overlapping n-gram reduces the tendency of having a lot of similar nodes as repetitive vectors will increase the similarity between nodes. It also ensures every node has a feature vector length of 150 and the vertex cardinality of every graph is 10. Basically, our packet-level graphs have 10 nodes and a node embedding size of 150. The Graph generation process is consistent for both flow-level and packet-level graphs, as both are created using the same methodology. The packet-level graphs are an abstraction of the flow-level graphs that emphasizes granularity. The only difference is in the initial data-processing as outlined in Algorithm 6, where we convert a packet of dimension (1 x 1500) to (10 x 150). Specifically, for packet-level processing, a packet with dimensions (1 x 1500) is reshaped into (10 x 150) by stacking every 150 elements of the original array to form a new array of dimensions (10 x 150). To create the edges, we follow the trivial and non-trivial creation process as described in Equations 9 and 10. The process described for flow and packet-level graph generation lays the basis for our graph creation process. Algorithm 6 operates on the output of Algorithm 1 and converts flow graphs to packet graphs by taking the array X and dropping the last two columns to obtain X' . Each packet (row in X') is then split into 10 non-overlapping vectors of length 150. Each non-overlapping vector is treated as a new row, and the corresponding label and packet flow number columns are appended. The output is an array of dimension $10n \times 152$. To get packet graph files, the same supporting algorithms 2-5 are followed, the only difference is the input array has to be modified.

3.3. GNN Architecture and Design

Our approach to graph generation focuses on creating fine-grained, interconnected graphs with crucial topological distinctions for comprehensive classification. To leverage these structural and representative differences, we employ a deep learning model inspired by the postulations of Morris et al. [25]. This GNN architecture is particularly effective as it analyzes sub-graphs rather than just individual nodes, enabling operation on higher-order graphs.

Our architecture consists of five stacked GNN layers, each followed by a batch normalization and pooling layer, as illustrated in Figure 2. This choice of five GNN layers is determined by Equation 11. In graph theory, the eccentricity of a node is defined as the greatest distance from that node to any other node within the graph, with the maximum eccentricity representing the graph’s diameter. For optimal message-passing neural network (MPNN) performance, the number of layers L in an MPNN should be at least equal to the graph’s diameter D [20] formalized as $L \geq D$. Directly scaling with diameter can lead to excessively deep networks, causing overfitting, over-smoothing, and computational overhead. To address this, we propose a practical method to determine the optimal number of GNN layers, given by the equation:

$$L \approx \max \left(\left\lceil \frac{1}{N} \sum_{i=1}^N D_i \right\rceil, D_{p85} \right) \quad (11)$$

In this equation, we compare the average diameter of the graphs, $\left\lceil \frac{1}{N} \sum_{i=1}^N D_i \right\rceil$ to the 85th percentile diameter D_{p85} and select the larger value. Using the 85th percentile prevents the network depth from being unduly influenced by outliers with unusually large diameters, while the average diameter ensures that the network depth reflects the most significant average structural spread in any graph. Based on our calculations, the optimal number of GNN layers L is determined to be 5. This refined approach balances depth and computational efficiency, facilitating effective message propagation while avoiding common pitfalls of GNNs.

We utilize batch normalization to stabilize training and mitigate overfitting, while dimensionality reduction is achieved using a top-k pooling layer [7]. This layer selects the top nodes via a trainable projection vector, simplifying computation and emphasizing important nodes for robust graph embeddings. As the graph traverses the network, we compute global max and mean pools at each GNN layer, concatenating these to form a single embedding. Each of the five GNN layers produces its own embedding, resulting in five embeddings that are concatenated into a final graph representation. For classification, we employ a three-layer multi-layer perceptron (MLP) with a softmax classifier as the final layer, illustrated in Figure 2. Dropout layers after each linear layer help prevent overfitting.

4. Experiments

4.1. Dataset

To validate our model, we conducted nine experiments using four datasets. The ISCX VPN dataset, Non-VPN (IS-CXVPN2016) [4] dataset, Tor (ISCTXor2016) [13] dataset. USTC-TFC dataset [43] and the encrypted mobile instant messaging dataset [5]. Table 1 describes the classes for each dataset. Our selection of public datasets ensures effective reproducibility and comparative analysis. The datasets captures network traffic at different levels of encryption (VPN, Non-VPN and Tor). We conducted both flow-level

Table 1

Datasets used in our experiments.

Dataset	Class description			
	chat voip	email	file	stream
non-vpn	chat voip	email	file	stream
vpn	chat voip	email p2p	file	stream
Tor	audio mail	browsing p2p	chat video	file voip
ieee-im	b/ground signal whatsapp	discord teams	gmail telegram	messenger browsing
Ustc(m)	tinba shifu virut	zeus htbot nsis-ay	miuref neris	geodo crindex
Ustc(b)	facetime outlook weibo	skype warcraft SMB	torrent mysql	gmail ftp

and packet-level classification tasks. Further experiments for impact analysis are discussed in this section. All datasets is in “.pcap and pcapng” format.

4.2. Experimental Setup

4.2.1. Pre-processing

For the flow level experiments, we adopt similar processing techniques for the VPN, Non-VPN, USTC-TFC, IEEE-im dataset and a slightly different approach for the Tor data. This is due to the tendency of noise occurring within regular VPN and Non-VPN traffic and not in Tor. In an earlier implementation by Platzer et al. [32], Tor is described as an example of a network without generated traffic noise to make traffic analysis more difficult. This is due to the implementation of uniform packet sizes, circuit construction and padding dummy traffic. We report results on the VPN, Non-VPN and Tor dataset for flow and packet level experiments. For the USTC-TFC and IEEE-im dataset we focus on only flow traffic classification.

Our Data processing is performed as follows: Our window size is calculated as defined in 8. We delete flows with no packets and remove handshake packets within flows. We mask the IP address, and port numbers and strip off the Ethernet headers of all packets. At the transport layer, the TCP and UDP headers are mostly utilised; both protocols have different header orientations, with TCP length being 20 bytes and UDP length 8 bytes, hence we pad UDP packets with 12 bytes to ensure the lengths are uniform. The traffic flows are converted to raw bytes and normalised as in 3. The MTU of network packets is 1500 bytes, so we pad packets less than 1500 with zeros to ensure uniform length of packets. Finally we truncate traffic flows to ensure 10 packets per flow. For Tor traffic flows we truncate traffic flows to ensure 20 packets per flow. The generated graphs have nodes and edges with defining attributes. Nodes are represented by the packets and edges are a synthesis of the non-trivial and trivial edges. the edge weights are the similarity values of packets. The calculated threshold for the trivial edge is 0.7

Table 2
Classification Sample size distribution of datasets.

Dataset		Sample Size		
		Training	Test	Val
non-vpn	pac	13214	4405	4404
	flow	1677	560	559
vpn	pac	18000	6000	6000
	flow	2421	807	807
Tor	pac	9600	3200	3200
	flow	3123	1041	1041
ieee-im	pac	30000	10000	10000
	flow	31833	10615	10607
Ustc(m)	pac	25272	8424	8424
	flow	12000	4000	4000
Ustc(b)	pac	30000	10000	10000
	flow	12000	4000	4000

for the VPN, Non-VPN, IEEE-im and USTC dataset. For the Tor dataset, The calculated threshold for the trivial edge is 0.95 . We create our flow graphs from the processed data as defined in 3.1 and label every graph with its corresponding traffic class (label).

For the packet level experiment we perform the initial deletion of unwanted packets, IP masking, Ethernet header stripping, TCP/UDP padding and normalization. To create an even data distribution and tackle overfitting from the onset, we ensure every traffic class has at least 5000 packets (5000 graphs). We create our packet graphs from the processed data as defined in 3.2 and label every graph with its corresponding traffic class (label). For the Tor dataset, we adopt a slightly different approach after normalisation, due to its similar packet structure. Duplicate vectors were eliminated and the n-gram size was reduced. By removing the duplicates, the possibility of having a similar byte value distribution for every n-gram is eliminated. The maximum vector length becomes 255. A more granular value for n-gram takes into consideration the minute differences between packet byte sequences necessary for distinguishing them. We model our Tor packet nodes as bi-grams and follow up with the non-trivial and trivial edge creation.

4.2.2. Training Specification

The PyTorch geometric library with a Python 3.10 backend is used to generate the graphs, build, train, validate and test our model. The hardware specification is a Linux Dell 6.2.0-26-generic server, the processor is a 12th Gen Intel(R) Core(TM) i9-12900, 125GB of physical RAM and an NVIDIA RTX A4000 GPU. To reduce the impact of the imbalance nature of the VPN and Non-VPN data we trained with a weighted random sampler for the traffic flow classification. For all experiments, we split our dataset into three 60% training 20% validation and 20% testing (see Table 2). The hyper-parameters are as follows, batch size is 256, epoch is 500. Adam optimizer is used to improve the categorical cross-entropy loss function with a learning rate of 0.001, betas=0.9 and 0.999, epsilon=1e-08 and a decay rate of 0.00001. We use the same training specifications

for all experiments. We evaluate our model using the four standard classification metrics namely, Precision, Recall, $F1$ Score and Accuracy. We perform stratified cross validation across our dataset with a fold of 5 to curb over-fitting.

4.3. Results and Analysis

Table 3 and 4 shows a comparison of our work with selected baseline literature. The selected papers applied AI techniques to the task of traffic classification and can be categorized into machine, deep learning and GNN papers. Although GNN is an aspect of machine learning, we analyse it separately because this paper focuses on GNN. While we compare with various state of the art papers, a fair comparison would be literature that processes data in a similar way i.e. graph data. From the table, we can infer that, DL methods seem to have an edge over ML methods, this is due to the ability of DL methods to perform automatic feature extraction on data. Traditional ML methods utilise feature selection which entails manual hand-crafted features for analysis. For DL methods, feature representation is key to the classification process. Network traffic has to be converted to a different form before analysis. While this method may work for some classes of network traffic e.g. (VPN and Non-VPN). Table 3 shows that enhanced encryption reduces classification accuracy. Most models fail to classify Tor traffic because Tor packets are very similar in nature. Hence when feature representation is performed on Tor traffic there is a tendency that the transformed traffic will also be similar. Due to the granularity of our methods (especially packet level), we are able to overcome this challenge and classify all three traffic classes optimally. At the flow level, our method depends on the packets to model graphs and at the packet level we go even granular, analysing the byte stream of packets for classification.

Our packet-level graphs outperform TFE-GNN [51] by 0.87% and 9.03% for the VPN and Non-VPN datasets respectively. TFE-GNN by Zhang et al. [51] performs slightly better than our model by 2.86% for Tor traffic due to their dual embedding approach. By generating embeddings for both headers and payload, they generate stronger representations of network traffic that fails to generalize optimally. The complexity of their model might prove impractical for real-time detection and induce latency to the network. Overall, our packet-level classification outperforms all other compared models.

Table 4 show results of flow and packet-level experiments on the Ustc [43] and Ieee-im dataset [5] with our method outperforming the baselines. For the USTC(Malware) the flow-level results beats the packet-level by 0.010. This demonstrates the impact of sample size and graph structure on classification accuracy. The sample sizes for the Ustc dataset [43] are evenly distributed and abundant hence the inter-class accuracies are high, making the results balanced. The Ieee-im dataset was used to simulate an open world scenario with the presence of background noise traffic and an uneven sample distribution. The packet level achieved the best accuracy reaching 92% outperforming the baselines.

Table 3

Comparison with existing methods. We focus on methods that use ML (CUMUL to K-FP), DL (MVML to Deep Packet), and GNN (ECD-GNN to TFE-GNN). Our packet-level experiments proved superior for the VPN and Non-VPN classification tasks.

Datasets →	ISCX-VPN				ISCX-NonVPN				ISCX-Tor			
	Acc	Pr	Rc	F1	Acc	Pr	Rc	F1	Acc	Pr	Rc	F1
CUMUL [28]	0.766	0.753	0.785	0.764	0.619	0.594	0.597	0.590	0.669	0.535	0.490	0.500
FAAR [18]	0.836	0.822	0.840	0.829	0.737	0.751	0.712	0.725	0.697	0.592	0.488	0.481
AppScanner [38]	0.889	0.868	0.882	0.872	0.758	0.759	0.747	0.749	0.754	0.663	0.604	0.616
ETC-PS [46]	0.749	0.744	0.746	0.745	0.727	0.741	0.713	0.721	0.737	0.700	0.709	0.705
GRAIN [49]	0.813	0.808	0.811	0.809	0.667	0.653	0.666	0.660	0.541	0.525	0.535	0.523
Flowprint [39]	0.854	0.745	0.792	0.757	0.694	0.707	0.731	0.713	0.240	0.030	0.125	0.048
K-FP [9]	0.871	0.875	0.875	0.875	0.755	0.748	0.735	0.739	0.777	0.742	0.621	0.631
MVML [6]	0.649	0.723	0.620	0.615	0.513	0.575	0.471	0.481	0.634	0.391	0.410	0.375
DF [37]	0.895	0.855	0.871	0.853	0.886	0.839	0.861	0.846	0.713	0.710	0.724	0.695
FlowPic [35]	0.984	0.900	0.908	0.917	0.938	0.834	0.830	0.834	0.869	0.649	0.650	0.649
EDC [14]	0.784	0.776	0.814	0.789	0.709	0.719	0.709	0.706	0.757	0.542	0.537	0.530
FFB [50]	0.833	0.888	0.830	0.806	0.703	0.726	0.721	0.739	0.675	0.647	0.663	0.623
FS-NET [17]	0.939	0.923	0.912	0.915	0.932	0.909	0.926	0.914	0.848	0.842	0.781	0.783
ET-BERT [15]	0.953	0.944	0.951	0.947	0.917	0.925	0.923	0.924	0.954	0.924	0.961	0.942
Deep Packet [19]	0.920	0.920	0.920	0.920	0.951	0.960	0.950	0.955	0.430	0.570	0.440	0.497
ECD-GNN[11]	0.932	0.913	0.940	0.926	0.870	0.882	0.866	0.871	0.057	0.033	0.256	0.058
GraphDapp [36]	0.649	0.567	0.610	0.588	0.568	0.568	0.590	0.579	0.429	0.256	0.251	0.228
Okonkwo et al. [26]	0.975	0.975	0.975	0.975	0.871	0.872	0.871	0.871	0.758	0.757	0.692	0.692
TFE-GNN [51]	0.959	0.953	0.959	0.954	0.904	0.932	0.919	0.924	0.989	0.979	0.994	0.986
Our Work (flow)	0.947	0.948	0.947	0.947	0.964	0.966	0.964	0.965	0.928	0.929	0.928	0.929
Our Work (pac)	0.968	0.968	0.968	0.968	0.994	0.994	0.995	0.995	0.960	0.960	0.960	0.960

Table 4

Comparison of Baseline Models on USTC-TFC and IEEE-IM Datasets, our method outperforms the baselines. The USTC-TFC dataset shows that when samples are balanced and abundant the flow-graphs can perform as good as the packet-graphs

Datasets →	USTC-TFC (Benign)				USTC-TFC (Malware)				IEEE-IM			
	Acc	Pr	Rc	F1	Acc	Pr	Rc	F1	Acc	Pr	Rc	F1
CUMUL	0.744	0.829	0.744	0.784	0.937	0.938	0.937	0.937	0.659	0.798	0.656	0.720
FAAR	0.901	0.905	0.903	0.904	0.958	0.953	0.958	0.955	0.699	0.772	0.699	0.734
AppScanner	0.733	0.710	0.733	0.721	0.956	0.956	0.956	0.956	0.799	0.809	0.799	0.804
ETC-PS	0.661	0.674	0.661	0.667	0.935	0.938	0.935	0.936	0.712	0.770	0.712	0.740
GRAIN	0.914	0.915	0.914	0.914	0.968	0.964	0.968	0.966	0.707	0.791	0.709	0.748
Flowprint	0.681	0.710	0.690	0.700	0.933	0.933	0.934	0.933	0.737	0.660	0.676	0.668
K-FP	0.654	0.716	0.654	0.684	0.827	0.832	0.827	0.829	0.823	0.848	0.823	0.835
MVML	0.648	0.673	0.648	0.660	0.890	0.903	0.890	0.896	0.685	0.770	0.685	0.725
DF	0.899	0.910	0.899	0.904	0.863	0.898	0.863	0.880	0.813	0.845	0.813	0.829
Flowpic	0.854	0.886	0.854	0.870	0.961	0.968	0.961	0.964	0.813	0.845	0.813	0.829
EDC	0.862	0.838	0.862	0.850	0.951	0.952	0.951	0.951	0.806	0.717	0.747	0.732
FFB	0.905	0.932	0.905	0.918	0.951	0.951	0.951	0.951	0.792	0.826	0.792	0.809
FS-NET	0.951	0.955	0.951	0.953	0.946	0.947	0.946	0.946	0.838	0.868	0.838	0.853
ET-BERT	0.966	0.975	0.966	0.970	0.963	0.964	0.963	0.963	0.827	0.851	0.827	0.839
Deep Packet	0.901	0.946	0.901	0.923	0.914	0.922	0.914	0.918	0.827	0.742	0.803	0.771
ECD-GNN	0.932	0.937	0.932	0.934	0.930	0.935	0.930	0.932	0.827	0.835	0.827	0.831
GraphDapp	0.966	0.974	0.966	0.970	0.969	0.969	0.969	0.969	0.815	0.841	0.815	0.828
Okonkwo et al.	0.983	0.983	0.983	0.983	0.966	0.966	0.966	0.966	0.822	0.822	0.819	0.820
TFE-GNN	0.956	0.956	0.956	0.956	0.960	0.960	0.960	0.960	0.850	0.860	0.850	0.855
Our Work (flow)	0.974	0.992	0.973	0.982	0.974	0.974	0.973	0.974	0.883	0.892	0.913	0.905
Our Work (pac)	0.982	0.983	0.982	0.982	0.964	0.965	0.964	0.964	0.920	0.920	0.920	0.919

4.3.1. Ablation Studies

To further analyze the impact of our proposed method, we conducted an ablation study to assess the significance of graph features and model depth on classification tasks. Table 5 presents our results on the Non-VPN dataset for both

flow and packet-level classifications. In the first experiment, we eliminated edge weights, resulting in a drop in accuracy: flow classification decreased from 0.964 to 0.925, and packet classification dropped from 0.994 to 0.988. In the second experiment, we removed both trivial edges and weights,

Table 5

Ablation Studies. The study assesses the importance of our graph features and the model's depth. The Non-VPN dataset is used to conduct experiments for the study. f | p in the second row signifies flow and packet-level experiments. For every classification metric, results for the flow (on the left) and packet (on the right) experiments are displayed.

Method		non-trivial	trivial	weighted	Acc		Pr		Rc		F1	
Non-VPN (f p)		✓	✓	✓	0.964	0.994	0.966	0.994	0.964	0.995	0.965	0.995
1	w/o weights	✓	✓	×	0.925	0.988	0.932	0.988	0.925	0.988	0.927	0.988
2		✓	×	×	0.786	0.929	0.823	0.928	0.818	0.929	0.817	0.928
3	w/o trivial	✓	×	✓	0.780	0.981	0.839	0.981	0.784	0.981	0.807	0.981
4	3-GNN	✓	✓	✓	0.929	0.976	0.929	0.976	0.929	0.976	0.928	0.976
5	4-GNN	✓	✓	✓	0.925	0.981	0.927	0.981	0.925	0.981	0.925	0.981
6	6-GNN	✓	✓	✓	0.925	0.985	0.926	0.985	0.925	0.985	0.925	0.985
7	7-GNN	✓	✓	✓	0.943	0.947	0.949	0.947	0.943	0.947	0.944	0.947

causing a significant accuracy decline: flow classification fell to 0.786, and packet classification to 0.929. These results underscore the importance of trivial edges for classification. Analysing the results of the packet experiments, its minimum observed accuracy score is 0.929 and this occurs when the trivial edges and weights are excluded from the graphs. This outcome suggests that the granular nature of the packet-level graphs allows it to retain the most salient characteristics from the node embeddings leading to a strong learning performance despite the removal of seemingly trivial elements.

To examine the impact of model depth, we restored the baseline parameters and tested with varying GNN layers (three, four, six, and seven). For packet-level experiments, we observed a slight accuracy decrease as layers were reduced, while flow-level experiments showed the opposite trend. Adding more layers initially maintained performance but eventually led to a decline. This is directly related to our data-processing strategy; for flow-level analysis, we truncate flows to the first 10 packets, simplifying graph generation but potentially weakening embeddings and leading to over-smoothing with additional layers. Although avoiding truncation could enhance performance, it would also increase computational overhead. For efficiency, we applied minimal truncation. These ablation studies confirm the crucial role of trivial edges and weights in classification tasks.

4.3.2. GNN Variant Studies

In Table 6, we choose four variants of GNN, Graph Convolution (GCN), Graph Attention Networks (GAT), GraphSage [8] and Graph Isomorphism Networks (GIN) [45] for our variant studies. While there exist other variants of GNNs, we select these four as they are widely considered the most powerful GNNs. Other GNNs are simply modifications of the above four to perform neighbourhood aggregation and propagation in different ways. In our experiments, we subject the Non-VPN dataset to a variant studies at the packet level. GCN achieves a score of 0.985 across all metric. GAT achieves 0.503 accuracy on the Non-VPN dataset. GraphConv, GIN and GraphSage perform the best. With GraphConv and GraphSage having relatively similar values. Table 6 shows that our network graphs can be adapted and trained with other variants of GNNs. Our results, shown in

Table 6

Variant studies. The leftmost column GRAPHCONV is the GNN adopted by our work; hence, our results from the packet-level experiments in Table3 are given in the column.

Metric	non-VPN				
	G_Conv	GCN	GAT	GIN	G_Sage
Pr	0.994	0.985	0.735	0.979	0.991
Rc	0.995	0.985	0.503	0.979	0.991
F1	0.995	0.985	0.467	0.979	0.991
Acc	0.994	0.985	0.503	0.979	0.991

the “*GraphConv*” column of Table 6, slightly outperform the selected variants.

4.3.3. Explainability

GNN explainability aims to elucidate the inner workings of graph neural networks. We illustrate this by addressing how predictions are made on generated network graphs. Our explainability analysis employs four methods. Firstly, we assess the impact of packet header and payload data on classification. As shown in Table 7, headers and payloads are used independently for flow-level classification tasks, revealing that headers play a more crucial role in traffic detection. While headers outperform payloads, the latter still achieve accuracies of 88% for nonVPN and 83% for VPN traffic, similar to findings in previous studies [51, 19, 1, 41]. This is particularly noteworthy given that cryptographic algorithms produce pseudo-random ciphertext. These results are rather counter-intuitive, suggesting that further experiments are warranted to uncover the mechanisms by which the payload contributes to the classification.

To test this hypothesis, we conduct a *feature importance analysis* on the nonVPN dataset using GNNExplainer [48]. GNNExplainer identifies the node features most informative for the model's output. As outlined in section 3.1, network packets are modeled as graph nodes, with node features representing raw packet bytes. By applying GNNExplainer, we determine the bytes with the greatest importance and trace these bytes back to their actual representation in the original network packet. For optimal explanations, we analyze batches of network graphs instead of individual instances. Table 4.3.3 lists the top 10 features influencing

Table 7
Header/Payload impact.

Metric	Non VPN		VPN	
	Header	Payload	Header	Payload
Pr	0.930	0.885	0.938	0.871
Rc	0.921	0.880	0.932	0.834
F1	0.924	0.880	0.934	0.842
Acc	0.921	0.880	0.932	0.834

the model's predictions, highlighting the predominance of packet header contents. Specifically, the most important byte corresponds to the *Timestamp Echo Reply (TSecr)* in TCP packets, suggesting the model initially distinguishes TCP from UDP packets through these bytes. It is important to note that our data processing steps altered the positions of bytes, akin to changes that occur with different packet header configurations. For clarity, we have appended the relevant information for each byte position.

In the next explainability analysis, we focus on elucidating the role of the trivial edges in classification, utilizing the PGExplainer [22], a parameterized explainer for GNNs. Our choice is driven by the distinct topological structures produced in our graph generation process, necessitating an explainer adept at providing topological explanations.

Consider an input graph G_o comprising two sub-graphs G_s and ΔG , such that $G_o = G_s + \Delta G$. The sub-graph G_s significantly contributes to the GNN's prediction, while ΔG consists of less critical nodes and edges. PGExplainer aims to identify G_s while maximizing the mutual information between the GNN's prediction and the underlying structure G_s . To examine this structural importance, we utilize the Ieee-im dataset focusing on the Gmail application. After training the model on the network graphs with parameters as stated in Section 4.2.2, we train the explainer with the following default parameters: the coefficient of size regularization is set to 0.05, and entropy regularization is 1.0. Epoch is set to 30, and temperatures τ_0 and τ_1 are 0.5 and 0.2, respectively. We randomly sample 40 sub-graphs G_s for the application traffic using the trained explainer. Figure 7 illustrates the minimal graph structures yielding equivalent classification results. A noteworthy observation from these sub-graphs G_s is the prevalent occurrence of trivial edges (non-sequential connections between nodes). The presence of trivial edges in the sub-graphs underscores their importance in the GNN's prediction. Figure 7 akin to 5 reveal clear patterns, reinforcing the structural impact of our generated graphs.

Finally, to better understand the factors driving our model's predictions, we resort to decision trees due to their inherent interpretability. Jacobs et al. [12] developed trustee, a system to probe ML/DL network traffic based classifiers thereby elucidating the decision-making process. Leveraging this framework, we generate explanations by initially processing our network graphs with our classifier. Instead of using linear layers for the final classification we feed the embeddings to the trustee based decision tree. This

Table 8
Top 10 Features from GNNExplainer.

Byte	TCP	UDP
49	ts echo reply	payload byte
41	tls record version	payload byte
28	ack num	payload byte
10	header checksum	header checksum
40	content type	payload byte
3	total length	total length
8	time to live	time to live
43	tls record version	payload byte
51	echo reply	payload byte
45	timestamp value	payload byte

test is conducted on the non-VPN dataset, the resulting decision tree, depicted in Figure 8, has a depth of 6 and highlights that byte 3 (referred to as *feature* in the fig 8) corresponding to the total length of the packet, emerges as the most significant predictor. Additionally, bytes 2, 4, and 1 also contribute significantly. Bytes 2 and 3 (total length) suggest that variations in packet length are highly indicative of the class. This observation is further corroborated by Table 4.3.3. Feature 1 represents the type of service (TOS), and feature 4, representing identification, contributes significantly to further splits within the tree, particularly influencing the classification at intermediate nodes.

Upon further analysis, it is evident that the decision tree can be effectively pruned to a depth of 4. However, persistent confusion between chat and email classes, combined with the requirements to achieve a gini value of zero, necessitates deeper splits. These additional splits rely on random payload bytes, which, due to encryption, are inherently random. This reliance results in less robust decision-making. Our observations on decisions based on payload bytes are itemized as follows:

Decisions based on payloads

- Node *B3*: Decisions are made based on byte 1096, a payload byte. Confusion arises from one sample (see *value* in node *B3*) finally classified as VoIP in node *B4*. The decision tree, in its effort to minimize gini impurity to zero and ensure pure leaf nodes, must classify every sample. This results in additional nodes until a the gini value of zero is reached, which weakens the decision-making criteria. Pruning the tree involves a trade-off between accuracy and depth. The decision tree indicates that payload bytes cannot be used to generalize optimally. Therefore, node *B3* can be considered a leaf node for the class label "file" and nodes *B4* and *B5* can be pruned.
- Node *A5*: Decisions are based on byte 1448, another payload byte. The decision here is due to two samples classified as chat. Nodes *A8* and *A9* can be considered a leaf node for the file traffic class, and its child nodes can be pruned.

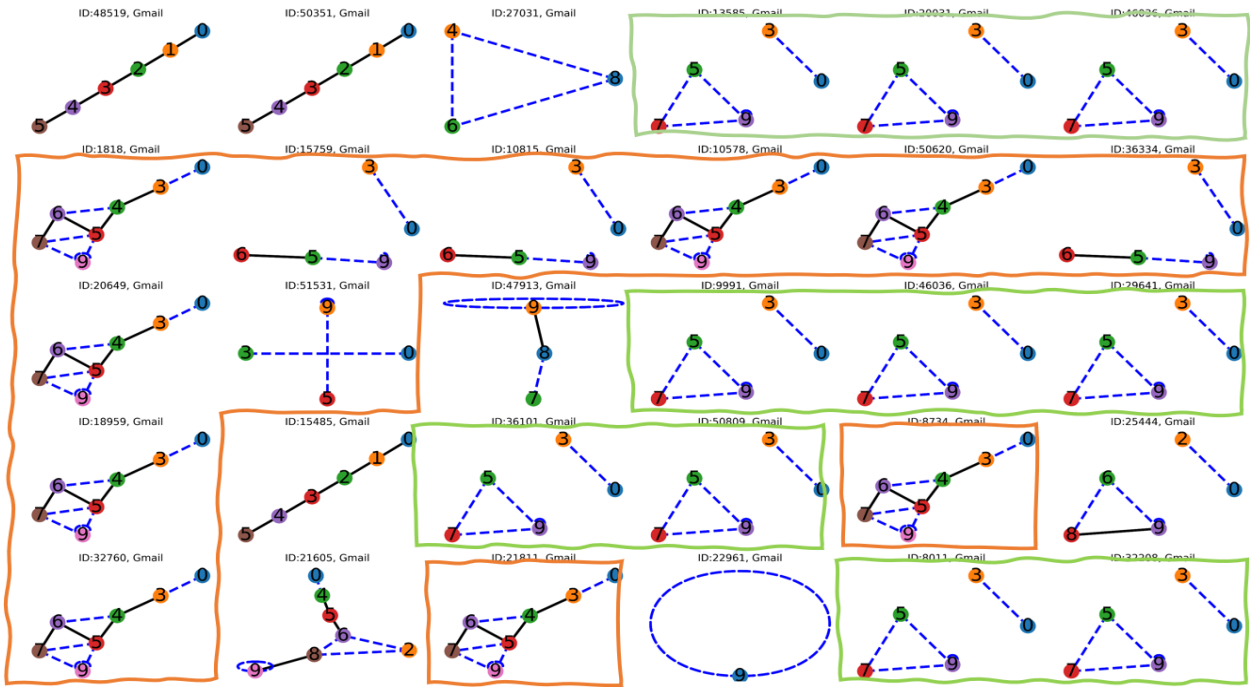


Figure 7: Flow-Level explanatory graphs. The graphs are the corresponding explanations for the sampled graphs shown in Figure 5 and highlights the nodes and edges that are crucial for the model's predictions. The distinctive structures, emphasized by our trivial edges (represented by dashed blue lines connecting nodes), reveal unique patterns specific to the application class. This underscores the intricate nature of trivial edges and their significant implications for traffic classification.

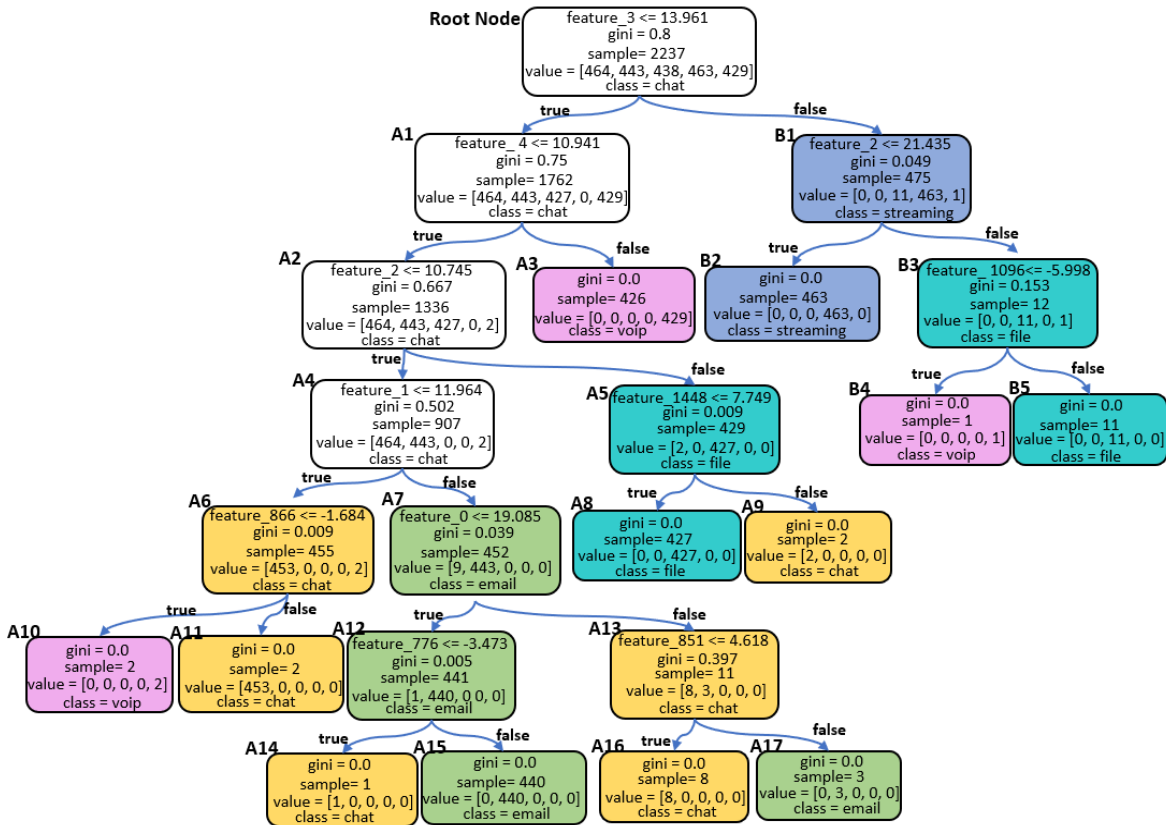


Figure 8: Decision tree depicting the classification of network traffic flows of the non-VPN dataset. Nodes represent decision points based on features such as total packet length (feature 3), with colors indicating different classes. All the nodes are labelled on the top left corner to aid identification. The tree structure highlights the hierarchical importance of features and their contribution to the final classification.

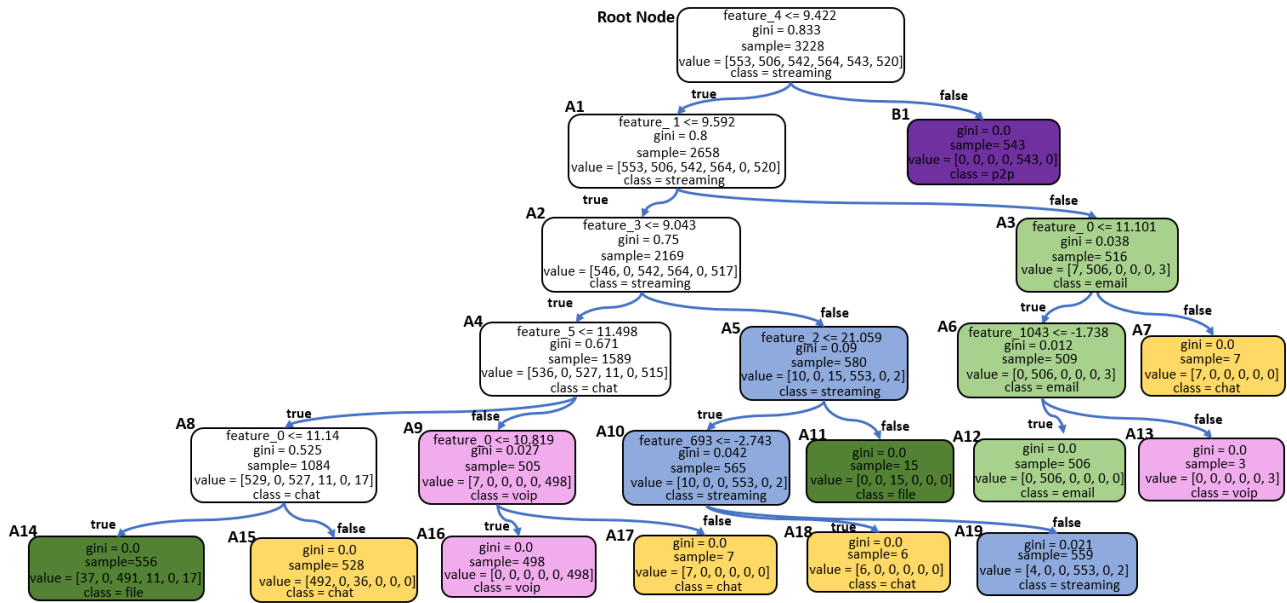


Figure 9: Decision tree depicting the classification of network traffic flows of the VPN dataset.

- Node A6: Decisions are based on byte 866, nodes A10 and A11 reveal two VoIP samples cause this confusion. Node A6 can be considered a leaf node, and its child nodes pruned.
- Node A7: This node spawns nodes A12 and A13. However, it can be considered a leaf node for the email traffic class, and nodes A12 and A13 can be pruned.

Figure 9 provides insights into how predictions are made on the VPN dataset, and a comparison with Figure 8 highlights key patterns in network traffic classification. Notably, the header bytes such as 4, 1, 3, 0, 5, and 2 are prominent in both decision trees, emphasizing their critical role in influencing classification decisions. The recurrence of these bytes across both figures suggests that they are pivotal features, contributing directly to network traffic classification tasks. However, in Figure 9, the payload bytes 1043 and 693 appearing in nodes A6 and A10 differ from the payload bytes observed in Figure 8, specifically 1096, 1448, and 866. This discrepancy indicates potential challenges in generalizing the model across different datasets. The variation in payload byte usage across the two decision trees implies that direct reliance on payload bytes for classification will lead to overfitting. This outcome suggests that while payload data might provide useful features for specific datasets, it lacks the robustness necessary for broad generalization.

This analysis provides further insight into Table 7 demonstrating that while payload features may improve classification performance, overfitting to these features can significantly impair the model’s generalization capabilities, particularly when applied to varied datasets. The reliance on payload bytes, which are random across different traffic types, should thus be approached cautiously.

Our explainability studies reveal that header features directly influence the model’s predictions in a generalizable manner. In contrast, using the payload directly as a feature may result in poor generalizations that do not significantly impact the model’s performance. However, the metadata defining both the header and payload can indirectly enhance the model’s efficacy. This suggests that systematically extracting structural information about the payload plays a crucial role in improving accuracy and the overall decision-making process.

5. Discussions

5.1. Encrypted Network Classification tasks

In the domain of encrypted network traffic classification (NTC), the traditional approach focuses heavily on the classification stage, often overlooking the complexities of network traffic data. This oversight makes NTC particularly challenging. Our approach, however, emphasizes the data processing and feature representation stages, simplifying the traffic structure and streamlining the classification process. We propose two graph generation processes: flow and packet. Each method has its distinct advantages and drawbacks. The flow traffic approach enhances the relationships between network packets at the node level and can extend to evaluate relations between traffic flows at the graph level, which is crucial for network behaviour analysis. However, a significant drawback is data imbalance. With limited traffic flows, the generated graphs are insufficient for training, reducing the model’s generalization capability. While techniques to mitigate over-fitting exist, under-represented classes still pose a challenge during classification. Conversely, the packet-level approach, although superior in traffic classification tasks, might not be as effective for



Figure 10: Confusion matrices of experiments carried out VPN, Non-VPN and Tor datasets. The packet-level experiments show better results than flow-level experiments for all experiments.

network behaviour analysis. Its fine granularity focuses on the intricacies of individual packets, potentially overlooking the broader network context. Our experiments, as illustrated in Figure 10, demonstrate that packet-level classification outperforms flow-level classification. This is primarily due to the abundance of data samples in packet-level experiments, ensuring balanced and well-represented classes during training. In summary, packet-level experiments tend to be more straightforward than flow-level due to the availability of data samples. The processed packet-level graphs maintain balance, allowing for a better representation of classes during training and leading to more robust classification outcomes.

5.2. Impact of Trivial edges

In our secondary analysis, including ablation, variant, and explainability studies, we primarily utilize the Non-VPN dataset due to its imbalanced nature, which accurately reflects real-world network data. In contrast, the VPN, Tor, Ustc and Im datasets are relatively balanced. Our studies reveal the critical role of trivial edges in overall task performance and model generalization. Figure 11 highlights the

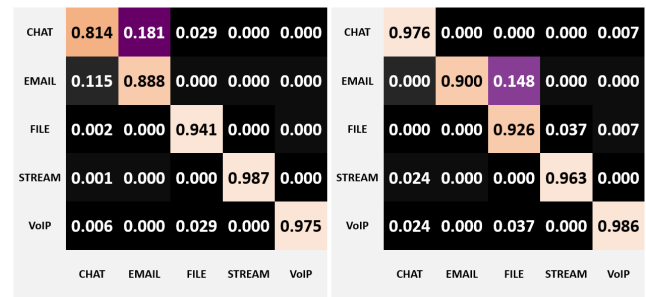


Figure 11: Confusion matrix of flow traffic classification on the Non-VPN dataset. The left image shows results from only non-trivial edges. The right results from both non-trivial and trivial edges.

classification performance of graphs generated with only non-trivial edges compared to those with both non-trivial and trivial edges. The inclusion of both edge types enhances performance, particularly for traffic types like chat and email. This differentiation underscores the varying behaviours of different traffic types when considering various edge types. These findings indicate that seemingly semantically meaningless application traffic contains subtle nuances that aid in classification. The model's generalization strength is notably improved when both edge types are used to construct graphs. The topological distinctions introduced by trivial edges are pivotal in this enhancement. The impact of the trivial edges can also be seen from figure 7. The connected sub-graphs are evidence that incorporating topographical distinction allows for a more nuanced understanding of encrypted NTC, enabling the classifiers to discern subtle patterns that may be overlooked by methods that do not consider these spatial relationships. The observed superiority of the packet-level method in Table 3 is attributed primarily to the granularity of the data processing stage, which plays a crucial role in generating abundant and well-balanced samples across all datasets. The flow-level method is more prone to data imbalances and a reduced sample size following data preprocessing, as illustrated in Table 2. These factors can adversely affect the model's accuracy, as shown in Table 3. Conversely, when sample balancing becomes an issue (a common real-world problem), the flow graphs demonstrate a stronger dependence on the underlying graph structure for accurate classification. This can be seen on Table 5, as accuracy drops to 0.780 when trivial edges are excluded.

6. Related Work

NTC has evolved from traditional methods like ports and Deep Packet Inspection (DPI) to advanced statistical, ML, and DL-based schemes [16, 35, 24, 33, 38, 9, 39, 49, 35, 26]. The dynamic nature of network traffic complicates the establishment of a universal classification scheme, necessitating distinct approaches for different tasks and traffic classes. We

focus on literature employing DL approaches, particularly GNNs, for NTC.

Early significant works, such as those by Wang et al. [42, 41] and Lotfollahi et al. [19], utilized raw bytes for traffic classification. Lotfollahi's Deep Packet framework leveraged CNNs and Stacked Autoencoders, performing well on VPN and non-VPN tasks but struggling with Tor traffic. This highlighted the insufficiency of traditional NTC methods for handling various encryption schemes. Recent advancements include Zhang et al. [51], who proposed a method that processes packet headers and payloads separately then concatenates their embeddings for enhanced classification. By calculating the point Wise mutual information between bytes, they [51] generated byte-level graphs for both headers and payloads. As recent encryption algorithms are considered secure, the performance of their methodology suggests a weakness in encryption which was not addressed. Furthermore, their method did not leverage the temporal information embedded in the byte sequences, instead relying solely on the basic embeddings generated by their graphs for downstream classification tasks. Their findings resulted that the header embeddings were more crucial than the payload embedding.

Huoh et al. [11] and others[26, 30] represented network traffic as graphs, modeling packets as nodes and chronological relationships as edges. However, the structural uniformity of these graphs limited the full potential of GNNs.

Pang et al. [29] introduced an approach that leverages the cumulative acknowledgment (ACK) mechanism of TCP to create additional edges for network graphs, significantly enhancing the classification of malicious traffic. However, the scope of their network graphs is confined exclusively to TCP traffic, which inherently limits the applicability of their method to other communication protocols. This focus on TCP traffic does not optimally address the challenges associated with representing encrypted traffic, as the generated graphs tend to appear similar, thereby reducing the distinctiveness needed for accurate classification. The cumulative acknowledgment mechanism, while essential for ensuring reliability in TCP communication, does not fully capture the behavior of application-level traffic. Instead, it reflects the state of the network, such as congestion and packet loss. This reliance on the ACK(n) mechanism introduces volatility, as its behavior can vary significantly depending on the network environment. Consequently, while effective in certain contexts, this approach may not be universally applicable, particularly when dealing with diverse and encrypted traffic.

Zheng et al. [52] proposed a method that effectively models traffic flows as graph nodes, with edges created when flows share a common IP address. This approach allows for the construction of a primary graph representing the entire dataset, and the task of encrypted network traffic classification (NTC) is framed as a node classification problem. The methodology demonstrated strong performance on the dataset utilized, showcasing its potential in certain contexts. This methodology of edge creation creates a problem in cases where shared IP addresses dominate edge formation

leading to a bias in learned representations, ultimately causing overfitting. Their method [52] performed well on the dataset utilised however the static nature of the approach limits the flexibility of the GNN to adapt to a more nuanced relationships. Cai et al. [3] proposes a sophisticated method that encodes packet length information into graphs. By converting packet lengths flows into states and representing these states as nodes in a graph, with edges reflecting the transition probabilities between states, they capture both the structural and sequential properties of the traffic flow. By mapping packet lengths to a finite number of states (e.g., 10 states based on packet length ranges), there is an inherent loss of granular information. Packets with different lengths within the same range are treated identically, which might obscure subtle variations in traffic patterns that could be relevant for classification. The complexity of their model also limits interpretability. While traditional methods have laid the groundwork, contemporary research highlights the potential of GNNs and advanced DL techniques to address the nuanced challenges of NTC, emphasizing the need for adaptable, environment-agnostic solutions.

7. Conclusion

In this paper, we propose a novel approach to encrypted traffic classification that leverages the sequential and affinity of network packets in geometrical space to generate structurally rich graphs. Recognizing the significance of both traffic flow and individual packet analysis, we introduce methods for each. Flow graphs model traffic patterns and behaviors, while packet graphs are crucial for granular multi-classification tasks. To facilitate comprehensive analysis, we employ a GNN that captures the structural properties essential for accurate classification. During data processing, we account for the impact of various network security protocols and privacy technologies. Our evaluation spans four public datasets, benchmarking against several state-of-the-art baselines. Our approach consistently outperforms existing methods. Ablation and explainability studies further highlight the effectiveness of our graph structures, revealing consistent sub-graphs for similar classes. Our results demonstrate the suitability of packet analysis for multi-classification tasks, achieving high accuracies of 0.968 on VPN traffic, 0.994 on Non-VPN traffic, and 0.960 on Tor traffic. Our findings show that despite encryption, network traffic can still be analysed and fingerprinted. For future work, we aim to conduct granular classification and fingerprinting tasks for encrypted application traffic, with a particular emphasis on open and real world testing. This approach will allow for a more detailed and robust analysis, contributing significantly to the field of network security and traffic analysis.

8.

References

- [1] Aceto, G., Ciuonzo, D., Montieri, A., Pescapé, A., 2018. Mobile encrypted traffic classification using deep learning, in: 2018 Network traffic measurement and analysis conference (TMA), IEEE. pp. 1–8.

- [2] Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P., 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 18–42.
- [3] Cai, W., Gou, G., Jiang, M., Liu, C., Xiong, G., Li, Z., 2021. Memg: Mobile encrypted traffic classification with markov chains and graph neural network, in: 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), IEEE. pp. 478–486.
- [4] Draper-Gil, G., Lashkari, A.H., Mamun, M.S.I., Ghorbani, A.A., 2016. Characterization of encrypted and vpn traffic using time-related, in: Proceedings of the 2nd international conference on information systems security and privacy (ICISSP), pp. 407–414.
- [5] Erdenebaatar, Z., Alshammari, R., Zincir-Heywood, N., Elsayed, M., Nandy, B., Seddigh, N., 2023. Encrypted mobile instant messaging traffic dataset. *IEEE/IFIP Network Operations and Management Symposium (NOMS 2023)* URL: <https://dx.doi.org/10.21227/aer2-kq52>, doi:10.21227/aer2-kq52.
- [6] Fu, Y., Liu, J., Li, X., Xiong, H., 2018. A multi-label multi-view learning framework for in-app service usage analysis. *ACM Transactions on Intelligent Systems and Technology (TIST)* 9, 1–24.
- [7] Gao, H., Ji, S., 2019. Graph u-nets, in: international conference on machine learning, PMLR. pp. 2083–2092.
- [8] Hamilton, W., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30.
- [9] Hayes, J., Danezis, G., 2016. k-fingerprinting: A robust scalable website fingerprinting technique, in: 25th USENIX Security Symposium (USENIX Security 16), pp. 1187–1203.
- [10] HE, H., YANG, Z., CHEN, X., 2022. Payload encoding representation from transformer for encrypted traffic classification. *ZTE Communications* 19, 90–97.
- [11] Huoh, T.L., Luo, Y., Zhang, T., 2021. Encrypted network traffic classification using a geometric learning model, in: 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), IEEE. pp. 376–383.
- [12] Jacobs, A.S., Beltiukov, R., Willinger, W., Ferreira, R.A., Gupta, A., Granville, L.Z., 2022. Ai/ml for network security: The emperor has no clothes, in: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 1537–1551.
- [13] Lashkari, A.H., Gil, G.D., Mamun, M.S.I., Ghorbani, A.A., 2017. Characterization of tor traffic using time based features, in: International Conference on Information Systems Security and Privacy, SciTePress. pp. 253–262.
- [14] Li, W., Quenard, G., 2021. Towards a multi-label dataset of internet traffic for digital behavior classification, in: 2021 3rd International Conference on Computer Communication and the Internet (ICCCI), IEEE. pp. 38–46.
- [15] Lin, X., Xiong, G., Gou, G., Li, Z., Shi, J., Yu, J., 2022. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification, in: Proceedings of the ACM Web Conference 2022, pp. 633–642.
- [16] Lin, Y.D., Lu, C.N., Lai, Y.C., Peng, W.H., Lin, P.C., 2009. Application classification using packet size distribution and port association. *Journal of Network and Computer Applications* 32, 1023–1030.
- [17] Liu, C., He, L., Xiong, G., Cao, Z., Li, Z., 2019. Fs-net: A flow sequence network for encrypted traffic classification, in: IEEE INFOCOM 2019-IEEE Conference On Computer Communications, IEEE. pp. 1171–1179.
- [18] Liu, X., Zhang, S., Li, H., Wang, W., 2021. Fast application activity recognition with encrypted traffic, in: Wireless Algorithms, Systems, and Applications: 16th International Conference, WASA 2021, Nanjing, China, June 25–27, 2021, Proceedings, Part II 16, Springer. pp. 314–325.
- [19] Lotfollahi, M., Jafari Siavoshani, M., Shirali Hossein Zade, R., Saberian, M., 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 1999–2012.
- [20] Loukas, A., 2020. How hard is to distinguish graphs with graph neural networks? *Advances in neural information processing systems* 33, 3465–3476.
- [21] Lu, B., Luktarhan, N., Ding, C., Zhang, W., 2021. Iclstm: encrypted traffic service identification based on inception-lstm neural network. *Symmetry* 13, 1080.
- [22] Luo, D., Cheng, W., Xu, D., Yu, W., Zong, B., Chen, H., Zhang, X., 2020. Parameterized explainer for graph neural network. *Advances in neural information processing systems* 33, 19620–19631.
- [23] Ma, X., Zhu, W., Wei, J., Jin, Y., Gu, D., Wang, R., 2023. Eetc: An extended encrypted traffic classification algorithm based on variant resnet network. *Computers & Security* 128, 103175.
- [24] Miller, B., Huang, L., Joseph, A.D., Tygar, J.D., 2014. I know why you went to the clinic: Risks and realization of https traffic analysis, in: Privacy Enhancing Technologies: 14th International Symposium, PETS 2014, Amsterdam, The Netherlands, July 16–18, 2014. Proceedings 14, Springer. pp. 143–163.
- [25] Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M., 2019. Weisfeiler and leman go neural: Higher-order graph neural networks, in: Proceedings of the AAAI conference on artificial intelligence, pp. 4602–4609.
- [26] Okonkwo, Z., Foo, E., Hou, Z., Li, Q., Jadidi, Z., 2023. Encrypted network traffic classification with higher order graph neural network, in: Australasian Conference on Information Security and Privacy, Springer. pp. 630–650.
- [27] Okonkwo, Z., Foo, E., Li, Q., Hou, Z., 2022. A cnn based encrypted network traffic classifier, in: Proceedings of the 2022 Australasian Computer Science Week. Springer. pp. 74–83.
- [28] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., Wehrle, K., 2016. Website fingerprinting at internet scale., in: NDSS.
- [29] Pang, B., Fu, Y., Ren, S., Jia, Y., 2023. High-performance network traffic classification based on graph neural network, in: 2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), IEEE. pp. 800–804.
- [30] Pang, B., Fu, Y., Ren, S., Wang, Y., Liao, Q., Jia, Y., 2021. Cgmn: traffic classification with graph neural network. *arXiv preprint arXiv:2110.09726*.
- [31] Peng, L., Yang, B., Chen, Y., Wu, T., 2014. How many packets are most effective for early stage traffic identification: An experimental study. *China Communications* 11, 183–193.
- [32] Platzer, F., Schäfer, M., Steinebach, M., 2020. Critical traffic analysis on the tor network, in: Proceedings of the 15th International Conference on Availability, Reliability and Security, pp. 1–10.
- [33] Rahman, M.S., Sirinam, P., Mathews, N., Gangadhara, K.G., Wright, M., 2019. Tik-tok: The utility of packet timing in website fingerprinting attacks. *arXiv preprint arXiv:1902.06421*.
- [34] Sengupta, S., Ganguly, N., De, P., Chakraborty, S., 2019. Exploiting diversity in android tls implementations for mobile app traffic classification, in: The World Wide Web Conference, pp. 1657–1668.
- [35] Shapira, T., Shavitt, Y., 2019. Flowpic: Encrypted internet traffic classification is as easy as image recognition, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE. pp. 680–687.
- [36] Shen, M., Zhang, J., Zhu, L., Xu, K., Du, X., 2021. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Transactions on Information Forensics and Security* 16, 2367–2380.
- [37] Sirinam, P., Imani, M., Juarez, M., Wright, M., 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 1928–1943.
- [38] Taylor, V.F., Spolaor, R., Conti, M., Martinovic, I., 2016. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic, in: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE. pp. 439–454.

- [39] Van Ede, T., Bortolameotti, R., Continella, A., Ren, J., Dubois, D.J., Lindorfer, M., Choffnes, D., van Steen, M., Peter, A., 2020. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic, in: Network and distributed system security symposium (NDSS).
- [40] Wang, W., Sheng, Y., Wang, J., Zeng, X., Ye, X., Huang, Y., Zhu, M., 2017a. Hast-ids: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE access* 6, 1792–1806.
- [41] Wang, W., Zhu, M., Wang, J., Zeng, X., Yang, Z., 2017b. End-to-end encrypted traffic classification with one-dimensional convolution neural networks, in: 2017 IEEE international conference on intelligence and security informatics (ISI), IEEE. pp. 43–48.
- [42] Wang, W., Zhu, M., Zeng, X., Ye, X., Sheng, Y., 2017c. Malware traffic classification using convolutional neural network for representation learning, in: 2017 International conference on information networking (ICOIN), IEEE. pp. 712–717.
- [43] Wang, W., Zhu, M., Zeng, X., Ye, X., Sheng, Y., 2017d. Malware traffic classification using convolutional neural network for representation learning. 2017 International Conference on Information Networking (ICOIN) , 712–717URL: <https://api.semanticscholar.org/CorpusID:3725747>.
- [44] Wang, Z., Thing, V.L., 2023. Feature mining for encrypted malicious traffic detection with deep learning and other machine learning algorithms. *Computers & Security* 128, 103143.
- [45] Xu, K., Hu, W., Leskovec, J., Jegelka, S., 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* .
- [46] Xu, S.J., Geng, G.G., Jin, X.B., Liu, D.J., Weng, J., 2022. Seeing traffic paths: encrypted traffic classification with path signature features. *IEEE Transactions on Information Forensics and Security* 17, 2166–2181.
- [47] Yao, H., Liu, C., Zhang, P., Wu, S., Jiang, C., Yu, S., 2019. Identification of encrypted traffic through attention mechanism based long short term memory. *IEEE Transactions on Big Data* 8, 241–252.
- [48] Ying, Z., Bourgeois, D., You, J., Zitnik, M., Leskovec, J., 2019. Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems* 32.
- [49] Zaki, F., Afifi, F., Abd Razak, S., Gani, A., Anuar, N.B., 2022. Grain: Granular multi-label encrypted traffic classification using classifier chain. *Computer Networks* 213, 109084.
- [50] Zhang, H., Gou, G., Xiong, G., Liu, C., Tan, Y., Ye, K., 2021. Multi-granularity mobile encrypted traffic classification based on fusion features, in: Science of Cyber Security: Third International Conference, SciSec 2021, Virtual Event, August 13–15, 2021, Revised Selected Papers 4, Springer. pp. 154–170.
- [51] Zhang, H., Yu, L., Xiao, X., Li, Q., Mercaldo, F., Luo, X., Liu, Q., 2023. Tfe-gnn: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification, in: Proceedings of the ACM Web Conference 2023, pp. 2066–2075.
- [52] Zheng, J., Zeng, Z., Feng, T., 2022. Gcn-eta: high-efficiency encrypted malicious traffic detection. *Security and Communication Networks* 2022, 1–11.