# TFE-GNN: A Temporal Fusion Encoder Using Graph Neural Networks for Fine-grained Encrypted Traffic Classification


Haozhen Zhang
Shenzhen International Graduate School, Tsinghua University
China
zhang-hz21@mails.tsinghua.edu.cn

Le Yu
Department of Computing, The Hong Kong Polytechnic University
Hong Kong, China
yulele08@gmail.com

Xi Xiao*
Shenzhen International Graduate School, Tsinghua University
China
xiaox@sz.tsinghua.edu.cn

Qing Li*
Peng Cheng Laboratory
China
liq@pcl.ac.cn

Francesco Mercaldo
University of Molise
IIT-CNR
Italy
francesco.mercaldo@unimol.it

Xiapu Luo
Department of Computing, The Hong Kong Polytechnic University
Hong Kong, China
csxluo@comp.polyu.edu.hk

Qixu Liu
Institute of Information Engineering, Chinese Academy of Sciences
China
liuqixu@iie.ac.cn



## ABSTRACT

Encrypted traffic classification is receiving widespread attention from researchers and industrial companies. However, the existing methods only extract flow-level features, failing to handle short flows because of unreliable statistical properties, or treat the header and payload equally, failing to mine the potential correlation between bytes. Therefore, in this paper, we propose a byte-level traffic graph construction approach based on point-wise mutual information (PMI), and a model named **T**emporal **F**usion **E**ncoder using **G**raph **N**eural **N**etworks (**TFE-GNN**) for feature extraction. In particular, we design a dual embedding layer, a GNN-based traffic graph encoder as well as a cross-gated feature fusion mechanism, which can first embed the header and payload bytes separately and then fuses them together to obtain a stronger feature representation. The experimental results on two real datasets demonstrate that TFE-GNN outperforms multiple state-of-the-art methods in fine-grained encrypted traffic classification tasks.



## CCS CONCEPTS

• **Security and privacy** → **Network security**; • **Information systems** → **Data mining**.

## KEYWORDS

Traffic Classification, User Behaviour, Graph Neural Networks



*Corresponding authors.





*WWW '23, April 30–May 04, 2023, Austin, TX, USA*



ACM ISBN 978-1-4503-9416-1/23/04.
https://doi.org/10.1145/3543507.3583227

**ACM Reference Format:**
Haozhen Zhang, Le Yu, Xi Xiao, Qing Li, Francesco Mercaldo, Xiapu Luo, and Qixu Liu. 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 (WWW '23), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3543507.3583227


## 1 INTRODUCTION

To protect user privacy and anonymity, various encryption techniques are used to encrypt the transmission of network traffic [28]. Although Internet security is improved for a regular user, encryption technologies also provide a convenient disguise for some malicious attackers. Moreover, some privacy-enhanced tools like VPN and Tor [26] may be utilized to achieve illegal network transactions, such as weapon trading and drug sales, where it is difficult to trace the traffic source [13]. Traditional data packet inspection (DPI) methods concentrate on mining the potential patterns or keywords in data packets, which is time-consuming and loses its accuracy when facing encrypted traffic [24]. Consequently, how to effectively represent encrypted network traffic for more accurate detection and identification is a significant challenge.

To solve the above problems, many approaches have been proposed. The earliest port-based works are no longer effective due to the application of dynamic ports. Subsequently, a series of statistic-based methods emerged [8, 31, 37, 40, 43], which rely on statistical features from traffic flows (e.g., mean of packet length). Then, a machine learning classifier (e.g., random forest) is adopted to get the final prediction results. Unfortunately, these methods need hand-crafted feature engineering and may fail due to the unreliable/unstable flow-level statistical information in some cases [36]. Most statistical features of relatively short flows have higher deviations compared with long flows. For example, the flow length

generally obeys the long-tailed distribution [45], implying the universal existence of unreliable statistical features. Therefore, we use packet bytes instead of those statistical features.

Recently, graph neural networks (GNNs) [14] have been widely used in lots of applications of processing unstructured data. Due to the powerful expressiveness, GNNs can recognize specific topological patterns implied in graphs so that we can classify each graph with a predicted label. For the traffic classification task, most current GNN-based methods [1, 11, 21, 25, 29] construct graphs according to the correlation between packets, which actually is another usage form of statistical features and also suffers from the issue mentioned above. While the others do utilize packet bytes but have two major flaws: **1) Mix usage of the header and payload.** Existing methods simply treat the header and payload of a packet equally but ignore the difference in meaning between them. **2) Inadequate utilization of raw bytes.** Although the packet bytes are utilized, most methods regard packets as nodes and just take their raw bytes as node features, which does not make the most of them [11].

Based on the above observations, in this paper, we propose a byte-level traffic graph construction approach based on point-wise mutual information (PMI) and a novel model named **T**emporal **F**usion **E**ncoder using **G**raph **N**eural **N**etworks (**TFE-GNN**) for encrypted traffic classification. The byte-level traffic graphs are constructed by mining the correlation between bytes and served as inputs for TFE-GNN. TFE-GNN consists of three major submodules (i.e., dual embedding, traffic graph encoder, and cross-gated feature fusion mechanism). The dual embedding treats the header and payload of a packet separately and embeds them using two independent embedding layers. As for the traffic graph encoder which consists of multilayer GNNs, it encodes each graph into a high-dimensional graph vector. Finally, we use the cross-gated feature fusion mechanism to integrate header graph vectors and payload graph vectors, obtaining an overall representation vector of a packet. For end-to-end training, we employ a time series model to get final prediction results for downstream tasks. In the experiment section, we adopt a self-collected WWT dataset (including the data from WeChat, WhatsApp and Telegram) as well as the public ISCX dataset to compare TFE-GNN with more than a dozen baselines. The experimental results show that TFE-GNN surpasses almost all the baselines and comprehensively achieves the most excellent performance on the adopted datasets (e.g., 10.82% ↑ on the Telegram dataset, 4.58% ↑ on the ISCX-Tor dataset).

In summary, the main contributions of this paper include:

- We *first* construct the byte-level traffic graph by converting a sequence of packet bytes into a graph, supporting traffic classification from a different perspective.
- We propose TFE-GNN, which treats the packet header and payload separately and encodes each byte-level traffic graph into an overall representation vector for each packet. Thus, TFE-GNN utilizes a packet-level representation vector rather than a flow-level one.
- To evaluate the performance of the proposed TFE-GNN, we compare it with several existing methods on the self-collected WWT dataset and public ISCX dataset [5, 15]. The result shows that, for user behaviour classification, TFE-GNN outperforms these methods in effectiveness.

# 2 PRELIMINARIES

## 2.1 Notations

In this paper, a graph is denoted by $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathrm{X}\}$, where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set, and $\mathrm{X} \in \mathbb{R}^{|\mathcal{V}| \times d_X}$ is the initial feature matrix of nodes whereby the initial feature of node $v$ can be represented by $x_v$. We use $\mathcal{A} \in \{0,1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ to represent the adjacency matrix of $\mathcal{G}$, which satisfies that the entry $(i, j)$ of $\mathcal{A}$, i.e., $a_{ij}$, equals 1 if there is an edge between nodes $i$ and $j$, otherwise it is 0. We use $N(v)$ to represent the neighborhood of node $v$. Moreover, we use $d_l$ to represent the embedding dimension in the $l$-th layer.

For brevity and convenience, we extend the concept of traffic flows by introducing time-induced **Traffic Segments (TS)**, which are collectively referred to as traffic samples in the rest of the paper.

$$TS = [P_{t_1}, P_{t_2}, \cdots, P_{t_n}],\ t_1 \le t_2 \le \cdots \le t_n \tag{1}$$

where $P_{t_i}$ denotes a single packet with its time stamp $t_i$, $n$ is the sequence length of a traffic segment, $t_1, t_n$ are the start and end times of a traffic segment, respectively. From the definition above, the traffic segment has a broader scope than the traffic flow, i.e., each traffic flow can be seen as a traffic segment, but the reverse does not necessarily hold. In this way, we can directly take traffic segments as training samples and do inference using either traffic flows or traffic segments, which helps to improve flexibility and unleash the expressiveness of an end-to-end model.

## 2.2 Encrypted Traffic Classification

The encrypted traffic classification task aims to differentiate the traffic generated from various sources (e.g., applications, web pages or services) by using the information of traffic packets captured by professional software or programs. In this paper, we concentrate on in-app user behaviour classification which differentiates fine-grained user actions such as sending texts and sending pictures.

Assume that there are $M$ training samples and $N$ categories in total, let the $i$-th traffic sample be a sequence $s_i = [bs_1^i, bs_2^i, \cdots, bs_n^i]$, where $n$ is the sequence length and $bs_j^i$ is the $j$-th byte sequence of the $i$-th traffic sample denoted by $bs_j^i = [b_1^{ij}, b_2^{ij}, \cdots, b_m^{ij}]$ where $m$ is the byte sequence length and $b_k^{ij}$ denotes the $k$-th byte value in the $j$-th byte sequence of the $i$-th traffic sample. According to the definition above, the (segment-level) encrypted classification task can be described formally as predicting the category $C_s$ of an unseen test sample $s_i$ with a designed and well-trained end-to-end model $F(s_i)$ on $M$ training samples, where $C_s = 0, 1, \cdots, N-1$.

## 2.3 Message Passing Graph Neural Networks

Graph Neural Networks (GNNs) [14] are powerful models for handling unstructured data. With the application of the message passing paradigm (MP) [6] to GNNs (MP-GNNs), the node embedding vectors can be updated iteratively by integrating nodes' embedding vectors in neighborhood through a specific aggregation strategy. Generally, the $l$-th layer MP-GNNs can be formalized as two procedures (i.e., the message computation and aggregation):

$$\begin{aligned} \mathbf{m}_u^{(l)} &= \mathrm{MSG}^{(l)}\left(\mathbf{h}_u^{(l-1)}; \theta_m^l\right) \\ \mathbf{h}_v^{(l)} &= \mathrm{AGG}^{(l)}\left(\mathbf{h}_v^{(l-1)}, \left\{\mathbf{m}_u^{(l)}, u \in N(v)\right\}; \theta_a^l\right) \end{aligned} \tag{2}$$

where $\mathbf{h}_u^{(l)}, \mathbf{h}_v^{(l)} \in \mathbb{R}^{d_l}$ are the embedding vectors of nodes $u$ and $v$ in layer $l$. $\mathbf{m}_u^{(l)}$ is the computed message from node $u$ in layer $l$. $\mathrm{MSG}^{(l)}(\cdot)$ is a message computation function parameterized by $\theta_m^l$ and $\mathrm{AGG}^{(l)}(\cdot)$ is a message aggregation function parameterized by $\theta_a^l$ in layer $l$. Notably, $\theta_m^l$ is optional and the inputs of MP-GNNs are given by initial node feature vectors (i.e., $\mathbf{h}_v^{(0)} = \mathbf{x}_v$).

Due to the high scalability of our proposed model, various GNN architectures can be easily adapted. Section 3.3 discusses the concrete choice of message aggregation strategies and our designed GNN architecture according to the design space of GNNs [42].

# 3 METHODOLOGY

## 3.1 Byte-level Traffic Graph Construction

We attempt to convert a sequence of bytes into a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathrm{X}\}$ by mining the potential correlation between bytes, where each element in $\mathcal{V}$ denotes a byte (i.e., a byte corresponds to a node in $\mathcal{G}$). Note that all the bytes with the identical value share the same nodes so that there are no more than 256 nodes in $\mathcal{G}$, which ensures a relatively small scale of traffic graphs.

**Correlation representation between bytes.** For edges, we can easily connect all bytes chronologically, which means creating an edge from byte $i$ to $j$ if byte $i$ comes before byte $j$ in a byte sequence. But we do not adopt this method since it will lead to a very dense graph and the topological structure will lack distinguishability. Therefore, inspired by [22] which uses cosine similarity to measure the correlation between two bytes, we adopt point-wise mutual information (PMI) [41], which is a prevalent measure for word association computation in natural language processing (NLP), to model the correlation between two bytes. In this paper, we represent the PMI value of bytes $i$ and $j$ as $\mathrm{PMI}(i, j)$.

**Edge creation.** The PMI value makes a comprehensive measurement of two co-occurrence bytes from the perspective of semantic associativity of bytes. We utilize it to create an edge between two bytes. A positive PMI value implies a high semantic correlation of bytes while a zero or a negative one implies little or no semantic correlation of bytes. Consequently, we only create an edge between two bytes whose PMI value is positive.

**Graph construction.** Below, we give the formal description of edges through the entries of adjacency matrix $\mathcal{A}$ of nodes $i$ and $j$:

$$a_{ij} = \begin{cases} 1, & \mathrm{PMI}(i, j) > 0 \\ 0, & \text{Otherwise} \end{cases} \tag{3}$$

The initial features of each node in graph $\mathcal{G}$ are given by the corresponding byte value, which ranges from 0 to 255. Notably, since $\mathrm{PMI}(i, j) = \mathrm{PMI}(j, i)$, the byte-level traffic graphs are undirected.

## 3.2 Dual Embedding

The byte value is commonly utilized to serve as initial features for further vector embedding. Two bytes with different values correspond to two distinct embedding vectors. However, the meaning of a byte varies not only with the byte value itself, but also with the part of the byte sequence in which it is located. In other words, the representation meaning of two bytes with the identical value within the header and payload of a packet respectively may be completely different. The reason is that the payload carries the transmission contents of a packet while the header is the first part of a packet that describes its contents. If we make two bytes with the identical value in the header and payload correspond to a same embedding vector, it is difficult for a model to converge to the optimum on these embedding parameters because of the obfuscated meaning.

For the rationale mentioned above, we treat the header and payload of a packet separately and construct byte-level traffic graphs for the two parts, respectively (i.e., byte-level traffic header graphs and byte-level traffic payload graphs). We adopt dual embedding with two embedding layers that do not share parameters to embed initial byte value features into high-dimensional embedding vectors for the two kinds of graphs, respectively.

**Dual embedding layer.** Assume that $d_0$ denotes the embedding dimension and $K$ is the number of embedding elements (i.e., byte value). The dual embedding matrices, which consist of two embedding matrices, can be viewed as $E_{header} \in \mathbb{R}^{K \times d_0}$ and $E_{payload} \in \mathbb{R}^{K \times d_0}$, where each row-wise entry represents the embedding vector of each byte value.

## 3.3 Traffic Graph Encoder with Cross-gated Feature Fusion

Since we construct byte-level traffic graphs based on the header and payload of packets, respectively, the following modules of TFE-GNN in this section are also dual, do not share parameters (architecture is the same) and can process in parallel.

**Traffic graph encoder.** To encode each traffic graph into a graph feature vector, we elaborately design a traffic graph encoder using stacked GraphSAGE [7], which is a powerful graph neural network. For every node $v$ in graph $\mathcal{G}$, GraphSAGE computes the message from each neighboring node $u \in N(v)$ by normalizing its embedding vector using the degree of node $v$. Then, GraphSAGE computes the overall message of all neighboring nodes $N(v)$ through element-wise mean operation and aggregates the overall message as well as the embedding vector of node $v$ through concatenation operation. Finally, a nonlinear transformation is applied to the embedding vector of node $v$, finishing the forward procedure of one GraphSAGE layer. Formally, the message computation and aggregation of GraphSAGE can be described by:

$$\begin{aligned} \mathbf{m}_{N(v)}^{(l)} &= \sum_{u \in N(v)} \frac{\mathbf{h}_u^{(l-1)}}{|N(v)|} \\ \mathbf{h}_v^{(l)} &= \sigma\left(\mathbf{w}^{(l)} \cdot \mathrm{CONCAT}\left(\mathbf{h}_v^{(l-1)}, \mathbf{m}_{N(v)}^{(l)}\right)\right) \end{aligned} \tag{4}$$

where $|N(v)|$ is the neighbor number of node $v$, $\mathbf{w}^{(l)} \in \mathbb{R}^{d_{l-1} \times d_l}$ is the parameter in layer $l$, $\mathrm{CONCAT}(\cdot)$ denotes the concatenation operation and $\sigma(\cdot)$ denotes the activation function. Specially, we employ parametric ReLU (PReLU) [9] as an activation function. PReLU scales each negative element value by a factor, which not only plays the effect of nonlinear transformation but also plays a role similar to that of the attention mechanism by different scale factors for each channel in the negative axis. Lastly, we normalize the updated feature vector $\mathbf{h}_v^{(l)}$ by batch normalization (BN) [12].

Due to the over-smoothing issue [2] in the deep GNN model, we only stack GraphSAGE up to 4 layers and concatenate the output

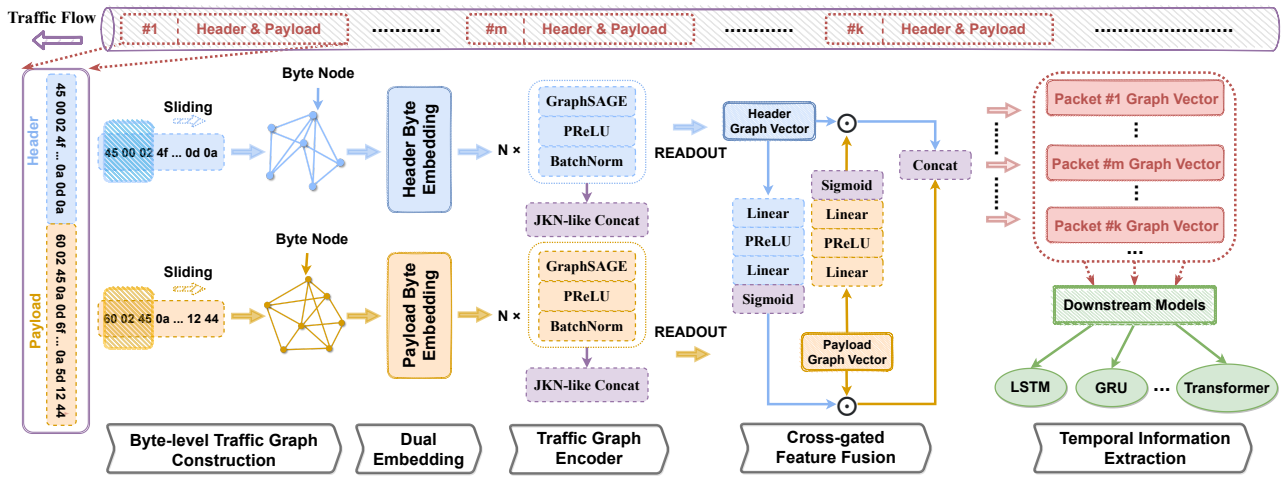



**Figure 1: TFE-GNN Model Architecture**

feature vectors of each layer for each node $v$ to alleviate this problem, which is similar to Jumping Knowledge Network (JKN) [39]:

$$\mathbf{h}_v^{\text{final}} = \text{CONCAT}(\mathbf{h}_v^{(1)}, \mathbf{h}_v^{(2)}, \mathbf{h}_v^{(3)}, \mathbf{h}_v^{(4)}) \tag{5}$$

where $\mathbf{h}_v^{\text{final}}$ is the final feature vector of node $v$. Finally, we apply mean pooling on all nodes to get a graph feature vector $\mathbf{g}$:

$$\mathbf{g} = \frac{\mathbf{h}_1^{\text{final}} \oplus \ldots \oplus \mathbf{h}_{|\mathcal{V}|}^{\text{final}}}{|\mathcal{V}|} \tag{6}$$

where $\oplus$ denotes element-wise addition. For simplicity, we use $\mathbf{g}_h$ and $\mathbf{g}_p$ to represent graph feature vectors extracted from traffic header graphs and traffic payload graphs, respectively.

**Cross-gated feature fusion.** Since we extract features from traffic header graphs and traffic payload graphs respectively mentioned in Section 3.2, we aim to create a reasonable relationship between $\mathbf{g}_h$ and $\mathbf{g}_p$ to get an overall representation of packet bytes. To this end, we carefully design a feature fusion mechanism named cross-gated feature fusion, to fuse $\mathbf{g}_h$ and $\mathbf{g}_p$ into a final encoded feature vector for each packet.

As shown in Figure 1, we adopt two filters, each of which consists of two linear layers with a PReLU activation function between them. First the two filters, which do not share parameters, are applied to $\mathbf{g}_h$ and $\mathbf{g}_p$, respectively and then an element-wise sigmoid function is used to scale each element to $[0, 1]$. We consider the scaled vectors as gated vectors ($\mathbf{s}_h$ and $\mathbf{s}_p$ for the header and the payload) and use them to crosswise filter the corresponding $\mathbf{g}_h$ and $\mathbf{g}_p$. Such a mechanism allows the model to filter out unimportant information and reserve the significant one for the two feature vectors. As the first part of the packet, the header describes its important features. Thus, it is reasonable to use header gated vector $\mathbf{s}_h$ to filter payload graph feature vector $\mathbf{g}_p$ and conversely use payload gated vector $\mathbf{s}_p$ to filter header graph feature vector $\mathbf{g}_h$.

The cross-gated feature fusion can be formally represented by:

$$\mathbf{s}_h = \text{Sigmoid}(\mathbf{w}_{h2}^T \, \text{PReLU}(\mathbf{w}_{h1}^T \mathbf{g}_h + \mathbf{b}_{h1}) + \mathbf{b}_{h2}) \tag{7}$$

$$\mathbf{s}_p = \text{Sigmoid}(\mathbf{w}_{p2}^T \, \text{PReLU}(\mathbf{w}_{p1}^T \mathbf{g}_p + \mathbf{b}_{p1}) + \mathbf{b}_{p2}) \tag{8}$$

$$\mathbf{z} = \text{CONCAT}(\mathbf{s}_h \odot \mathbf{g}_p, \mathbf{s}_p \odot \mathbf{g}_h) \tag{9}$$

where $\mathbf{w}_{h1}, \mathbf{w}_{h2}, \mathbf{w}_{p1}, \mathbf{w}_{p2} \in \mathbb{R}^{d_g \times d_g}$ and $\mathbf{b}_{h1}, \mathbf{b}_{h2}, \mathbf{b}_{p1}, \mathbf{b}_{p2} \in \mathbb{R}^{d_g}$ are the weights and biases of linear layers. The symbol $\odot$ denotes element-wise product and $\mathbf{z}$ is the overall representation vector of the packet bytes, which can be used for the downstream tasks.

## 3.4 End-to-End Training on Downstream Tasks

Based on the overall representation vector $\mathbf{z}$ for each packet, a packet-level or a segment-level classification task can be easily solved using a downstream classifier. We primarily focus on the segment-level task in this paper.

**Temporal information extraction.** Since we have already encoded raw bytes of each packet in a traffic segment into a representation vector $\mathbf{z}$, the segment-level classification task can be considered as a time series prediction task. Here, we just adopt long short-term memory (LSTM) [10], which is a classical and famous time series model, as our baseline downstream model. LSTM is bidirectional with two layers and its output vectors are fed into a two-layer linear classifier with PReLU as its activation function to get the final prediction results. Seeing that we need to compute the difference between prediction results and the ground truth, we just adopt the cross entropy function as the loss function:

$$\mathcal{L}_{TFE-GNN} = \text{CE}(\text{Classifier}(\text{LSTM}(\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n)), y) \tag{10}$$

where $n$ is the segment length, $y$ is the ground truth and $\text{CE}(\cdot)$ denotes the cross entropy function.

Specially, we also attempt to employ a transformer layer [33] as a downstream model, which is also an effective time series model based on the self-attention mechanism. The experimental results for transformers are also presented in the experiment section.

# 4 EXPERIMENTS

In this section, we first present experimental settings. Then, we conduct experiments on multiple datasets and baselines and analyze the results. We also conduct an ablation study to show the effectiveness of each component in TFE-GNN. For comprehensive analysis, we design some model variants to evaluate the scalability of TFE-GNN and compare several baselines w.r.t. their model complexity. Finally,

we analyse the model sensitivity of TFE-GNN. In detail, we conduct the experiments to answer the following questions:
**RQ1**: How is the usefulness of each component (Section 4.3)?
**RQ2**: Which GNN architecture performs best (Section 4.4)?
**RQ3**: How is the complexity of the TFE-GNN model (Section 4.5)?
**RQ4**: To what extent can changes in hyper-parameters affect the effectiveness of TFE-GNN (Section 4.6)?

## 4.1 Experimental Settings

*4.1.1 Dataset.* In order to comprehensively evaluate the effectiveness of TFE-GNN, we adopt multiple datasets, i.e., ISCX VPN-nonVPN [5], ISCX Tor-nonTor [15], and self-collected WWT datasets.

ISCX VPN-nonVPN is a public traffic dataset which contains ISCX-VPN and ISCX-nonVPN datasets. The ISCX-VPN dataset is collected over virtual private networks (VPNs) which are used for accessing some blocked websites or services and difficult to be recognized due to the obfuscation technology. Conversely, the traffic in ISCX-nonVPN is regular and not collected over VPNs.

Similarly, ISCX Tor-nonTor is a public dataset and ISCX-Tor dataset is collected over the onion router (Tor) whose traffic can be difficult to trace. Besides, ISCX-nonTor is also regular and not collected over Tor. For comparison, we use the ISCX VPN-nonVPN and ISCX Tor-nonTor datasets with six and eight user behaviour categories, respectively. We use SplitCap to obtain bidirectional flows from public datasets. Specially, due to the scarcity of flows in the ISCX-Tor dataset, we increase the training samples by dividing each flow into 60-second non-overlapping blocks in our experiments [27]. Finally, we utilize stratified sampling to sequentially partition the training and testing dataset into 9:1 for all datasets.

The WWT dataset includes fine-grained user behaviour traffic data from three social media apps (i.e., WhatsApp, WeChat and Telegram), which have twelve, nine and six user behaviour categories, respectively. Unlike the public ISCX dataset, we additionally record the start and end timestamps of each user behaviour sample for traffic segmentation.

*4.1.2 Pre-processing.* For each dataset, we define and filter out two kinds of "anomalous" samples: **(1) Empty flows or segments**: the traffic flows or segments where all packets have no payload. **(2) Overlong flows or segments**: the traffic flows or segments whose length (i.e., the number of packets) is larger than 10000. An empty flow or segment does not contain any payload, thus we can not construct the corresponding graph. In fact, such samples are generally used to establish connections between clients and servers, having little discriminating information that helps to classify. An overlong flow or segment contains too many packets and a large number of bad packets or retransmission packets may appear in it due to temporarily bad network environment or other potential reasons. In most cases, such samples introduce too much noise, so we also consider overlong flows or segments as anomalous samples and remove them. Additionally, as for each rest sample of datasets, we remove bad packets and retransmission packets within.

For each packet in a flow or segment, we first remove the ones without payload. Then we remove the Ethernet header, which only provides some irrelevant information for classification. The source and destination IP addresses, and the port numbers are all removed for the purpose of eliminating interference with sensitive information deriving from these IP addresses and port numbers.

*4.1.3 Implementation Details and Baselines.* In the stage of traffic graph construction, we set the max packet number of one sample to 50. The max payload byte length and the max header byte length are set to 150 and 40, respectively. The PMI window size is set to 5 by default. In the stage of training, we set the max training epoch to 120. The initial learning rate is set to 1e-2 and we use the Adam optimizer with a learning rate scheduler, which gradually decays the learning rate from 1e-2 to 1e-4. The batch size is 512, the ratio of warmup is 0.1 and the dropout rate is 0.2. We implement all models with PyTorch and run each experiment 10 times independently to take average on a single NVIDIA RTX 3080 GPU.

To give a fair comparison, we use four metrics, i.e., Overall Accuracy (AC), Precision (PR), Recall (RC) and Macro F1-score (F1), to evaluate TFE-GNN with following state-of-the-art baselines, including ***Traditional Feature Engineering Based Methods*** (i.e., AppScanner [31], CUMUL [23], K-FP (K-Fingerprinting) [8], FlowPrint [32], GRAIN [43], FAAR [19], ETC-PS [40]), ***Deep Learning Based Methods*** (i.e., FS-Net [18], EDC [16], FFB [44], MVML [4], DF [30], ET-BERT [17]), and ***Graph Neural Network Based Methods*** (i.e., GraphDApp [29], ECD-GNN [11]).

## 4.2 Comparison Experiments

The comparison results on WWT and ISCX datasets are shown in Tables 1 and 2. According to Tables 1 and 2, we can draw the following conclusions: (1) TFE-GNN reaches the best performance compared with several baselines on the WWT dataset. Additionally, TFE-GNN also achieves the best results on four metrics, which further comprehensively demonstrates the effectiveness of our method. (2) Notably we can find that almost all the baselines perform poor on the Telegram dataset, it is due to the fact that the usage of VPNs increases the classification difficulty and introduces some background noise in case of provisionally bad network conditions caused by VPNs. However, TFE-GNN also has an outstanding result on the Telegram dataset (10.82% f1-score improvement over the second highest), which benefits from the powerful byte encoding capability of TFE-GNN. (3) Compared with the two GNN-based methods similar to ours, i.e., GraphDApp and ECD-GNN, TFE-GNN outperforms both in all aspects. As for GraphDApp, its scheme of traffic interaction graph construction limits the expressiveness of the model. Although graphs from different traffic flows are slightly distinct in the aspect of traffic bursts, the edges between different bursts are unreasonable, which hinders feature extraction. Furthermore, an earlier burst can not "interact" with the later one using shallow GNNs because of the long and continuous connections between bursts. However, ECD-GNN is very unstable on different datasets. The reason is that the constructed graphs are lack of graph topology specificity and have highly similar structures, which significantly decreases its performance stability. With our elaborated byte-level graph construction approach, TFE-GNN can encode raw bytes well and has greater distinguishability among different traffic categories. (4) The extensive experiments on public datasets, i.e., the ISCX VPN-nonVPN and the ISCX Tor-nonTor, show that TFE-GNN can also perform well on more complicated datasets. From the Table 2, TFE-GNN is superior to almost all the baselines on public

**Table 1: Experimental Results on Self-collected WeChat, WhatsApp and Telegram Datasets**

| Dataset | WeChat | | | | WhatsApp | | | | Telegram | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | AC | PR | RC | F1 | AC | PR | RC | F1 | AC | PR | RC | F1 |
| AppScanner [31] | 0.9927 | 0.9908 | 0.9904 | 0.9905 | 0.9790 | 0.9688 | 0.9601 | 0.9628 | 0.8379 | 0.8154 | 0.8653 | 0.8304 |
| K-FP [8] | 0.9741 | 0.9665 | 0.9630 | 0.9645 | 0.9710 | 0.9589 | 0.9515 | 0.9526 | 0.8797 | 0.8378 | 0.8990 | 0.8567 |
| FlowPrint [32] | 0.7429 | 0.5302 | 0.6380 | 0.5532 | 0.6197 | 0.3820 | 0.4934 | 0.3880 | 0.4833 | 0.4025 | 0.5000 | 0.4311 |
| CUMUL [23] | 0.9472 | 0.9497 | 0.9412 | 0.9390 | 0.9855 | 0.9835 | 0.9699 | 0.9756 | 0.8330 | 0.7980 | 0.8471 | 0.8053 |
| GRAIN [43] | 0.9404 | 0.9366 | 0.9369 | 0.9251 | 0.9724 | 0.9685 | 0.9475 | 0.9550 | 0.8003 | 0.7615 | 0.7903 | 0.7407 |
| FAAR [19] | 0.9873 | 0.9863 | 0.9847 | 0.9854 | 0.9790 | 0.9683 | 0.9566 | 0.9597 | 0.8184 | 0.7884 | 0.8507 | 0.8018 |
| ETC-PS [40] | 0.9863 | 0.9864 | 0.9831 | 0.9846 | 0.9833 | 0.9743 | 0.9685 | 0.9703 | 0.8477 | 0.8295 | 0.8710 | 0.8382 |
| FS-Net [18] | 0.9223 | 0.9446 | 0.9196 | 0.8964 | 0.9942 | 0.9949 | 0.9919 | 0.9933 | 0.8469 | 0.8161 | 0.8744 | 0.8272 |
| DF [30] | 0.9800 | 0.9757 | 0.9751 | 0.9751 | 0.8978 | 0.7954 | 0.8406 | 0.8143 | 0.8251 | 0.8206 | 0.8542 | 0.7960 |
| EDC [16] | 0.9746 | 0.9653 | 0.9625 | 0.9620 | 0.9732 | 0.9589 | 0.9526 | 0.9546 | 0.8153 | 0.8118 | 0.8372 | 0.7896 |
| FFB [44] | 0.9675 | 0.9691 | 0.9661 | 0.9644 | 0.9350 | 0.8957 | 0.8767 | 0.8708 | 0.7990 | 0.7630 | 0.8169 | 0.7802 |
| MVML [4] | 0.9643 | 0.9519 | 0.9540 | 0.9526 | 0.9456 | 0.9302 | 0.9009 | 0.8971 | 0.7943 | 0.7413 | 0.7636 | 0.7340 |
| ET-BERT [17] | 0.9505 | 0.9368 | 0.9285 | 0.9266 | 0.9107 | 0.8934 | 0.8697 | 0.8439 | 0.7679 | 0.8712 | 0.7989 | 0.7858 |
| GraphDApp [29] | 0.8763 | 0.8368 | 0.8378 | 0.8330 | 0.8753 | 0.7615 | 0.8060 | 0.7791 | 0.7766 | 0.7575 | 0.7627 | 0.7227 |
| ECD-GNN [11] | 0.9863 | 0.9847 | 0.9830 | 0.9835 | 0.9811 | 0.9722 | 0.9647 | 0.9672 | 0.1810 | 0.0353 | 0.1641 | 0.0528 |
| **TFE-GNN** | **0.9956** | **0.9953** | **0.9939** | **0.9946** | **0.9971** | **0.9957** | **0.9966** | **0.9961** | **0.9586** | **0.9584** | **0.9742** | **0.9649** |

**Table 2: Experimental Results on Public ISCX VPN-nonVPN and ISCX Tor-nonTor Datasets**

| Dataset | ISCX-VPN | | | | ISCX-nonVPN | | | | ISCX-Tor | | | | ISCX-nonTor | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Model | AC | PR | RC | F1 | AC | PR | RC | F1 | AC | PR | RC | F1 | AC | PR | RC | F1 |
| AppScanner [31] | 0.8889 | 0.8679 | 0.8815 | 0.8722 | 0.7576 | 0.7594 | 7465 | 0.7486 | 0.7543 | 0.6629 | 0.6042 | 0.6163 | 0.9153 | 0.8435 | 0.8140 | 0.8273 |
| K-FP [8] | 0.8713 | 0.8750 | 0.8748 | 0.8747 | 0.7551 | 0.7478 | 0.7354 | 0.7387 | 0.7771 | 0.7417 | 0.6209 | 0.6313 | 0.8741 | 0.8653 | 0.7792 | 0.8167 |
| FlowPrint [32] | 0.8538 | 0.7451 | 0.7917 | 0.7566 | 0.6944 | 0.7073 | 0.7310 | 0.7131 | 0.2400 | 0.0300 | 0.1250 | 0.0484 | 0.5243 | 0.7590 | 0.6074 | 0.6153 |
| CUMUL [23] | 0.7661 | 0.7531 | 0.7852 | 0.7644 | 0.6187 | 0.5941 | 0.5971 | 0.5897 | 0.6686 | 0.5349 | 0.4899 | 0.4997 | 0.8605 | 0.8143 | 0.7393 | 0.7627 |
| GRAIN [43] | 0.8129 | 0.8077 | 0.8109 | 0.8027 | 0.6667 | 0.6532 | 0.6664 | 0.6567 | 0.6914 | 0.5253 | 0.5346 | 0.5234 | 0.7895 | 0.6714 | 0.6615 | 0.6613 |
| FAAR [19] | 0.8363 | 0.8224 | 0.8404 | 0.8291 | 0.7374 | 0.7509 | 0.7121 | 0.7252 | 0.6971 | 0.5915 | 0.4876 | 0.4814 | 0.9103 | 0.8253 | 0.7755 | 0.7959 |
| ETC-PS [40] | 0.8889 | 0.8803 | 0.8937 | 0.8851 | 0.7273 | 0.7414 | 0.7133 | 0.7208 | 0.7486 | 0.6811 | 0.5929 | 0.6033 | 0.9365 | 0.8700 | 0.8311 | 0.8486 |
| FS-Net [18] | 0.9298 | 0.9263 | 0.9211 | 0.9234 | 0.7626 | 0.7685 | 0.7534 | 0.7555 | 0.8286 | 0.7487 | 0.7197 | 0.7242 | 0.9278 | 0.8368 | 0.8254 | 0.8285 |
| DF [30] | 0.8012 | 0.7799 | 0.8152 | 0.7921 | 0.6742 | 0.6857 | 0.6717 | 0.6701 | 0.6514 | 0.4803 | 0.4767 | 0.4719 | 0.8568 | 0.8003 | 0.7415 | 0.7590 |
| EDC [16] | 0.7836 | 0.7747 | 0.8108 | 0.7888 | 0.6970 | 0.7153 | 0.7000 | 0.6978 | 0.6400 | 0.4980 | 0.4528 | 0.4504 | 0.8692 | 0.7994 | 0.7411 | 0.7451 |
| FFB [44] | 0.8304 | 0.8714 | 0.8149 | 0.8335 | 0.7020 | 0.7274 | 0.6945 | 0.7050 | 0.6343 | 0.4870 | 0.5203 | 0.4952 | 0.8954 | 0.7545 | 0.7430 | 0.7430 |
| MVML [4] | 0.6491 | 0.7231 | 0.6198 | 0.6151 | 0.5126 | 0.5751 | 0.4707 | 0.4806 | 0.6343 | 0.3914 | 0.4104 | 0.3752 | 0.7235 | 0.5488 | 0.5512 | 0.5457 |
| ET-BERT [17] | 0.9532 | 0.9436 | 0.9507 | 0.9463 | **0.9167** | 0.9245 | **0.9229** | 0.9235 | 0.9543 | 0.9242 | 0.9606 | 0.9397 | 0.9029 | 0.8560 | 0.8217 | 0.8332 |
| GraphDApp [29] | 0.6491 | 0.5668 | 0.6103 | 0.5740 | 0.4495 | 0.4230 | 0.3647 | 0.3614 | 0.4286 | 0.2557 | 0.2509 | 0.2281 | 0.6936 | 0.5447 | 0.5398 | 0.5352 |
| ECD-GNN [11] | 0.1111 | 0.0185 | 0.1667 | 0.0333 | 0.0606 | 0.0101 | 0.1667 | 0.0190 | 0.0571 | 0.0071 | 0.1250 | 0.0135 | 0.9078 | 0.8015 | 0.8168 | 0.7977 |
| **TFE-GNN** | **0.9591** | **0.9526** | **0.9593** | **0.9536** | 0.9040 | **0.9316** | 0.9190 | **0.9240** | **0.9886** | **0.9792** | **0.9939** | **0.9855** | **0.9390** | **0.8742** | **0.8335** | **0.8507** |

datasets except for the ISCX-nonVPN dataset, on which TFE-GNN and ET-BERT all reach similar results. However, ET-BERT is a large model with very complex model architecture while TFE-GNN is a slighter model which achieves the steadiest and best results on all datasets. We will analyse the model complexity in Section 4.5 later.

## 4.3 Ablation Study (RQ1)

In this section, we conduct an ablation study of TFE-GNN on the ISCX-VPN and the ISCX-Tor datasets and show experimental results in Table 3. To facilitate the presentation of results, we denote header, payload, dual embedding module, jumping knowledge network-like concatenation, cross-gated feature fusion and activation function and batch normalization as 'H', 'P', 'DUAL', 'JKN', 'CGFF' and 'A&N', respectively. Specially, we not only verify the effectiveness of each component in TFE-GNN, but also test the impact of some alternative modules or operations, including 'SUM' and 'MAX' operation on node features to get graph representation vectors instead of the default 'MEAN', and 'GRU' or 'TRANSFORMER' modules to serve as downstream models instead of LSTM.

From the component ablation study of Table 3, we can draw the following conclusions: (1) The packet headers play a more important role in classification than the packet payloads and different datasets have different levels of the header and payload importance (the f1-score decreases by 2.5% when switching the header to payload on the ISCX-VPN dataset and by 21.06% on the ISCX-Tor dataset). (2) The usage of dual embedding increases the f1-score by 3.63% and 0.95%, which indicates its general effectiveness. JKN-like concatenation and cross-gated feature fusion both enhance the performance of TFE-GNN by a similar margin on two datasets. (3) We further verify the impact of the activation function and batch normalization and a significant performance drop can be seen on both datasets, which demonstrates the necessity of this two operations.

While on the rest part of Table 3, we can also obtain the following several points: (1) The element-wise summation on node features performs worse than the mean operation by a margin of 11.1% and

**Table 3: Ablation Study of TFE-GNN on ISCX-VPN and ISCX-Tor Datasets**

| Method | H | P | DUAL | JKN | CGFF | A&N | AC | PR | RC | F1 |
|---|---|---|---|---|---|---|---|---|---|---|
| w/o P | ✓ | × | × | ✓ | × | ✓ | 0.8713 \| 0.9874 | 0.8261 \| 0.9788 | 0.8228 \| 0.9934 | 0.8230 \| 0.9806 |
| w/o H | × | ✓ | × | ✓ | × | ✓ | 0.8051 \| 0.8726 | 0.8232 \| 0.7780 | 0.7957 \| 0.7809 | 0.7980 \| 0.7700 |
| w/o DUAL | ✓ | ✓ | × | ✓ | ✓ | ✓ | 0.9310 \| 0.9829 | 0.9149 \| 0.9747 | 0.9209 \| 0.9801 | 0.9173 \| 0.9760 |
| w/o JKN | ✓ | ✓ | ✓ | × | ✓ | ✓ | 0.9474 \| 0.9790 | 0.9365 \| 0.9736 | 0.9397 \| 0.9879 | 0.9374 \| 0.9795 |
| w/o CGFF | ✓ | ✓ | ✓ | ✓ | × | ✓ | 0.9445 \| 0.9800 | 0.9329 \| 0.9717 | 0.9371 \| 0.9847 | 0.9339 \| 0.9770 |
| w/o A&N | ✓ | ✓ | ✓ | ✓ | ✓ | × | 0.6105 \| 0.2212 | 0.5576 \| 0.0555 | 0.5487 \| 0.1180 | 0.5289 \| 0.0548 |
| w/ SUM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 0.8497 \| 0.8194 | 0.8549 \| 0.7287 | 0.8380 \| 0.6986 | 0.8426 \| 0.6891 |
| w/ MAX | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 0.8480 \| 0.9870 | 0.8328 \| 0.9752 | 0.8115 \| 0.9778 | 0.8094 \| 0.9751 |
| w/ GRU | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 0.8550 \| 0.8932 | 0.8489 \| 0.8702 | 0.8287 \| 0.8664 | 0.8294 \| 0.8610 |
| w/ TRANSFORMER | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | 0.6754 \| 0.9777 | 0.5706 \| 0.9753 | 0.5992 \| 0.9820 | 0.5658 \| 0.9828 |
| **TFE-GNN (default)** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | **0.9591 \| 0.9886** | **0.9526 \| 0.9792** | **0.9593 \| 0.9939** | **0.9536 \| 0.9855** |

29.64%, respectively on two datasets w.r.t. f1-score. However, the element-wise maximum decreases the f1-score to worse results on the ISCX-VPN dataset while only decreases f1-score a little on the ISCX-Tor dataset. (2) We change the default downstream model LSTM to GRU, which worsens all metrics about ~10% on both datasets because of the simpler architecture of GRU. Furthermore, we employ a transformer as a downstream model for comprehensive experiments. The results show that transformer performs well on the ISCX-Tor dataset (drops f1-score within ~1%) while receives almost ~40% drop on the ISCX-VPN dataset.

## 4.4 GNN Architecture Variants Study (RQ2)

To illustrate the scalability of GNN-based temporal fusion encoder, we select some classical GNN architectures as variants (e.g., GAT [34], GIN [38], GCN [14] and SGC [35]) for comparison on Telegram, ISCX-VPN and ISCX-Tor datasets.

From Figure 2, we can find that GraphSAGE [7] achieves the best f1-score on three datasets. As for the rest variants, a noticeable drop in performance can be discovered, especially for GAT [34]. The rationale behind the results is that GNN models are easy to overfit on small-scale graphs like ours (number of nodes is up to 256). As for GAT [34], the application of the attention mechanism in neighborhood feature aggregation exacerbates overfitting, which leads to a significant decline in f1-score. Among the three datasets, the relatively small fluctuation of the results on the Telegram dataset further validates the analysis above, which benefits from its larger number of training samples.

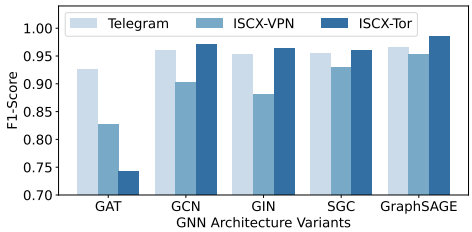



**Figure 2: GNN Architecture Variants Study w.r.t. F1-score**

Also, a segment-level global feature can be added and shared with all nodes within one traffic graph using our model architecture if needed, which naturally takes local (packet) and global (segment) context into account simultaneously. It can be easily realized by performing concatenation, element-wise addition or other similar operations due to the strong scalability of TFE-GNN.

## 4.5 Model Complexity Analysis (RQ3)

To comprehensively evaluate the trade-off between model performance and model complexity, we present the floating point operations (FLOPs) and the model size of all baselines except for the traditional models in Table 4.

From Tables 4 and 2, we can draw a conclusion that TFE-GNN achieves the most significant improvement on public datasets with relatively slight model complexity increasing. Although ET-BERT reaches comparable results on the ISCX-nonVPN dataset, the FLOPs of ET-BERT are approximately five times as large as that of TFE-GNN and the number of model parameters are also doubled, which generally indicates longer model inference time and requires more computation resources. Furthermore, the pre-training stage of ET-BERT is very time-consuming and costs a lot due to the large amount of extra data during pre-training and the high model complexity. In comparison, TFE-GNN can achieve higher accuracy while reducing the training or inference costs.

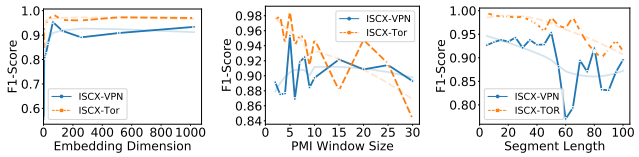
**Table 4: Model FLOPs and Parameters**

| Model | FLOPs(M) | Parameters(M) |
|---|---|---|
| FS-Net[18] | 1.0e+2 | 3.2e+0 |
| DF[30] | 2.8e+0 | 9.3e-1 |
| EDC[16] | 2.2e+1 | 2.2e+1 |
| FFB[44] | 2.6e+2 | 1.7e+0 |
| MVML[4] | 7.2e-4 | 3.7e-4 |
| ET-BERT[17] | 1.1e+4 | 8.6e+1 |
| GraphDApp[29] | 3.8e-2 | 1.1e-2 |
| ECD-GNN[11] | 2.9e+1 | 1.4e+0 |
| **TFE-GNN** | 2.2e+3 | 4.4e+1 |

## 4.6 Model Sensitivity Analysis (RQ4)

**(1) The Impact of Dual Embedding Dimension**. To investigate the influence of hidden dimension of the dual embedding layer, we conduct sensitivity experiments and show results in Figure 3a. As we can see, f1-score is increasing rapidly when embedding dimension is lower than 100. After that point, the model performance tends to be stable as the dimension changes. For reducing computation consuming, we just take embedding dimension 50 as our

default setting. **(2) The Impact of PMI Window Size**. From Figure 3b, we can find that a smaller window size usually results in better f1-score. The larger the window size, the more edges will be added in the traffic graphs, and the model will be harder to discriminate different traffic categories due to the too dense graphs. **(3) The Impact of Segment Length**. From Figure 3c, we can draw a conclusion that a short segment length for training usually makes the performance better. When the segment length becomes longer, more noise will be introduced and the downstream model LSTM has shortcomings in long sequence modeling, affecting the evaluation results. On the other hand, our method can achieve high accuracy when facing a short traffic flow or segment, reducing the amount of computation while improving performance.




**(a) Embedding Dim** **(b) PMI Window Size** **(c) Segment Length**

**Figure 3: Model Sensitivity Analysis w.r.t. Dual Embedding Dimension, PMI Window Size and Segment Length on ISCX-VPN and ISCX-Tor Datasets (The shallow lines are attained by smoothing the dark plotted lines)**

## 5 RELATED WORK

**Traditional Feature Engineering Based Methods.** Various methods leverage statistical features to depict the packet property and employ traditional machine learning models to conduct classification. AppScanner uses bidirectional flow characteristics (i.e., outgoing and incoming) to extract features from traffic flows w.r.t. packet length and interval time [31]. CUMUL uses the features of cumulative packet length [23] and GRAIN [43] uses payload length as its features. ETC-PS utilizes the path signature theory to enhance the original packet length features [40], and Liu *et al.* exploited packet length sequences using wavelet decomposition [19]. Conti *et al.* [3] adopts hierarchical clustering for feature extraction. The fingerprinting matching is also used in the traffic classification task. FlowPrint [32] constructs correlation graphs as traffic fingerprinting by computing activity value between destinations IP. K-FP [8] creates fingerprinting using random forest and matches unseen samples by k-nearest neighbor. All of these methods suffer from the unreliable features (mentioned in Section 1).

**Deep Learning Based Methods.** With the popularity of deep learning models, many traffic classification approaches are developed based on them. EDC [16] uses some header information of packets (e.g., protocol types, packet length and time duration) to build features for multilayer perceptions (MLPs). MVML [4] designs local and global features using packet length and time delay sequences, and simply employs a fully-connected layer for classification. Furthermore, FS-Net [18], DF [30] as well as RBRN [47] all utilize traffic flow sequences like packet length sequences to serve as the inputs of deep learning models. Additionally, DF and RBRN use convolutional neural networks (CNNs) while FS-Net utilizes gated recurrent units (GRUs) to extract temporal information of such sequences. For some other methods, packet bytes are used as model inputs to extract features. FFB [44] uses raw bytes and packet length sequences as features to feed into CNNs and RNNs. While Deep Packet [20] utilizes CNNs and autoencoders for feature extraction. Recently, pre-training models are utilized to pre-train on large-scale traffic data. To give an example, ET-BERT [17] designs two novel pre-training tasks for traffic classification, which enhance the representation ability of raw bytes but are very time-consuming and costly. In a word, these methods can not obtain the discriminative information which is contained in raw bytes very well in a relatively efficient way, while our approach solves this pain and difficulty by introducing byte-level traffic graphs.

**Graph Neural Network Based Methods.** Graph neural networks have strong potential in processing unstructured data and can be migrated to many fields. For encrypted traffic classification, GraphDApp [29] constructs traffic interaction graphs using traffic bursts and employs graph isomorphism network [38] to learn representations. MAppGraph [25] constructs traffic graphs based on different flows and time slices within a traffic chunk, which is almost impossible to construct a complete graph in the face of a short traffic segment. GCN-ETA [46] is a malicious traffic detection method. To construct a graph, it will create an edge if two flows share common IP, which may result in a very dense graph. MEMG [1] utilizes markov chains to construct graphs from flows while GAP-WF [21] maps a flow as a node in graphs and connects edges between flows which share the same identity of the clients. Besides, Huoh *et al.* [11] directly created edges based on the chronological relationship of packets among a flow, being lack of specificity. These methods all construct graphs at the level of traffic flows, which are vulnerable if there is too much noise within flows.

## 6 CONCLUSION AND FUTURE WORK

We propose an approach to construct byte-level traffic graphs and a model named TFE-GNN for encrypted traffic classification. The byte-level traffic graph construction approach can mine the potential correlation between raw bytes and generate discriminative traffic graphs. TFE-GNN is designed to extract high-dimensional features from constructed traffic graphs. Finally, TFE-GNN can encode each packet into an overall representation vector, which can be used for some downstream tasks like traffic classification. Several baselines are selected to evaluate the effectiveness of TFE-GNN. The experimental results show that our proposed model comprehensively surpasses all the baselines on the WWT and the ISCX datasets. Elaborately designed experiments further demonstrate that TFE-GNN has strong effectiveness.

In the future, we will attempt to improve TFE-GNN in terms of the following limitations. **(1) Limited graph construction approach**. The graph topology of the proposed model is determined before the training procedure, which may result in non-optimal performance. Moreover, the TFE-GNN can not cope with the byte-level noise implied in the raw bytes of each packet. **(2) Unused temporal information implied in byte sequences**. The byte-level traffic graphs are constructed without introducing the explicit temporal characteristics of byte sequences.

## ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (61972219,62202406,61972189), the Research and Development Program of Shenzhen (JCYJ20190813174403598, SGDX20190918101201696), the Overseas Research Cooperation Fund of Tsinghua Shenzhen International Graduate School (HW2021013), Shenzhen Science and Technology Innovation Commission: Research Center for Computer Network (Shenzhen) Ministry of Education, HK ITF Project (GHP/052/19SZ), and the Key Laboratory of Network Assessment Technology, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China.

## REFERENCES

[1] Wei Cai, Gaopeng Gou, Minghao Jiang, Chang Liu, Gang Xiong, and Zhen Li. 2021. MEMG: Mobile Encrypted Traffic Classification With Markov Chains and Graph Neural Network. In *IEEE International Conference on High Performance Computing and Communications*. 478–486.

[2] Weilin Cong, Morteza Ramezani, and Mehrdad Mahdavi. 2021. On Provable Benefits of Depth in Training Graph Convolutional Networks. In *Conference on Neural Information Processing Systems*. 9936–9949.

[3] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. 2015. Analyzing Android Encrypted Network Traffic to Identify User Actions. *IEEE Transactions on Information Forensics and Security* 11, 1 (2015), 114–125.

[4] Yanjie Fu, Junming Liu, Xiaolin Li, and Hui Xiong. 2018. A Multi-Label Multi-View Learning Framework for In-App Service Usage Analysis. *ACM Transactions on Intelligent Systems and Technology* 9, 4 (2018), 1–24.

[5] Gerard Drapper Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In *International Conference on Information Systems Security and Privacy*. 407–414.

[6] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning*. 1263–1272.

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Conference on Neural Information Processing Systems*.

[8] Jamie Hayes and George Danezis. 2016. k-fingerprinting: a Robust Scalable Website Fingerprinting Technique. In *USENIX Security Symposium*. 1187–1203.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *IEEE International Conference on Computer Vision*. 1026–1034.

[10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[11] Ting-Li Huoh, Yan Luo, and Tong Zhang. 2021. Encrypted Network Traffic Classification Using a Geometric Learning Model. In *IFIP/IEEE Symposium on Integrated Network Management*. 376–383.

[12] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. 448–456.

[13] Julian Jang-Jaccard and Surya Nepal. 2014. A survey of emerging threats in cybersecurity. *J. Comput. System Sci.* 80, 5 (2014), 973–993.

[14] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[15] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2017. Characterization of Tor Traffic Using Time Based Features. In *International Conference on Information System Security and Privacy*. 253–262.

[16] Wenbin Li and Gaspard Quenard. 2021. Towards a Multi-Label Dataset of Internet Traffic for Digital Behavior Classification. In *International Conference on Computer Communication and the Internet*. 38–46.

[17] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification. In *The Web Conference*. 633–642.

[18] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. 2019. FS-Net: A Flow Sequence Network For Encrypted Traffic Classification. In *IEEE Conference on Computer Communications*. 1171–1179.

[19] Xue Liu, Shigeng Zhang, Huihui Li, and Weiping Wang. 2021. Fast Application Activity Recognition with Encrypted Traffic. In *International Conference on Wireless Algorithms, Systems, and Applications*. 314–325.

[20] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2020. Deep packet: a novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 3 (2020), 1999–2012.

[21] Jie Lu, Gaopeng Gou, Majing Su, Dong Song, Chang Liu, Chen Yang, and Yangyang Guan. 2021. GAP-WF: Graph Attention Pooling Network for Fine-grained SSL/TLS Website Fingerprinting. In *IEEE International Joint Conference on Neural Network*. 1–8.

[22] Kelong Mao, Xi Xiao, Guangwu Hu, Xiapu Luo, Bin Zhang, and Shutao Xia. 2021. Byte-Label Joint Attention Learning for Packet-grained Network Traffic Classification. In *International Workshop on Quality of Service*. 1–10.

[23] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2016. Website Fingerprinting at Internet Scale. In *Annual Network and Distributed System Security Symposium*.

[24] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures. *Comput. Surveys* 54, 6 (2021), 1–35.

[25] Thai-Dien Pham, Thien-Lac Ho, Tram Truong-Huu, Tien-Dung Cao, and Hong-Linh Truong. 2021. MAppGraph: Mobile-App Classification on Encrypted Network Traffic using Deep Graph Convolution Neural Networks. In *Annual Computer Security Applications Conference*. 1025–1038.

[26] E Ramadhani. 2018. Anonymity communication VPN and Tor: a comparative study. *Journal of Physics: Conference Series* 983, 1 (2018), 012060.

[27] Tal Shapira and Yuval Shavitt. 2021. FlowPic: A Generic Representation for Encrypted Traffic Classification and Applications Identification. *IEEE Transactions on Network and Service Management* 18, 2 (2021), 1218–1232.

[28] Ritik Sharma, Sarishma Dangi, and Preeti Mishra. 2021. A Comprehensive Review on Encryption based Open Source Cyber Security Tools. In *IEEE International Conference on Signal Processing, Computing and Control*. 614–619.

[29] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. 2021. Accurate Decentralized Application Identification via Encrypted Traffic Analysis Using Graph Neural Networks. *IEEE Transactions on Information Forensics and Security* 16, 1 (2021), 2367–2380.

[30] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew K Wright. 2018. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *Conference on Computer and Communications Security*. 1928–1943.

[31] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2016. AppScanner: Automatic Fingerprinting of Smartphone Apps From Encrypted Network Traffic. In *IEEE European Symposium on Security and Privacy*. 439–454.

[32] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, and Martina Lindorfer. 2020. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In *Annual Network and Distributed System Security Symposium*.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Conference on Neural Information Processing Systems*.

[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[35] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*. 6861–6871.

[36] Hua Wu, Qiuyan Wu, Guang Cheng, Shuyi Guo, Xiaoyan Hu, and Shen Yan. 2021. SFIM: Identify user behavior based on stable features. *Peer-to-Peer Networking and Applications* 14, 6 (2021), 3674–3687.

[37] Guorui Xie, Qing Li, and Yong Jiang. 2021. Self-attentive deep learning method for online traffic classification and its interpretability. *Computer Networks* 196 (2021), 108267.

[38] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[39] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *International Conference on Machine Learning*. 5453–5462.

[40] Shi-Jie Xu, Guang-Gang Geng, and Xiao-Bo Jin. 2022. Seeing Traffic Paths: Encrypted Traffic Classification With Path Signature Features. *IEEE Transactions on Information Forensics and Security* 17, 1 (2022), 2166–2181.

[41] Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. Graph Convolutional Networks for Text Classification. In *AAAI Conference on Artificial Intelligence*. 7370–7377.

[42] Jiaxuan You, Zhitao Ying, and Jure Leskovec. 2020. Design Space for Graph Neural Networks. In *Conference on Neural Information Processing Systems*. 17009–17021.

[43] Faiz Zaki, Firdaus Afifi, Shukor Abd Razak, Abdullah Gani, and Nor Badrul Anuar. 2022. GRAIN: Granular multi-label encrypted traffic classification using classifier chain. *Computer Networks* 213, 1 (2022), 109084.

[44] Hui Zhang, Gaopeng Gou, Gang Xiong, Chang Liu, Yuewen Tan, and Ke Ye. 2021. Multi-granularity Mobile Encrypted Traffic Classification Based on Fusion Features. In *International Conference on Science of Cyber Security*. 154–170.

[45] Songyang Zhang, Zeming Li, Shipeng Yan, Xuming He, and Jian Sun. 2021. Distribution Alignment: A Unified Framework for Long-Tail Visual Recognition. In *Computer Vision and Pattern Recognition*. 2361–2370.
[46] Juan Zheng, Zhiyong Zeng, and Tao Feng. 2022. GCN-ETA: High-Efficiency Encrypted Malicious Traffic Detection. *Security and Communication Networks* 2022, 1 (2022), 11 pages.
[47] Wenbo Zheng, Chao Gou, Lan Yan, and Shaocong Mo. 2020. Learning to Classify: A Flow-Based Relation Network for Encrypted Traffic Classification. In *The Web Conference*. 13–22.

# A THREAT MODEL AND ASSUMPTIONS

In this section, we present the threat model and assumptions. Normal users employ mobile apps to communicate with remote servers. The attacker is a passive observer (i.e., he cannot decrypt or modify packets). The attacker captures the packets of the target apps by compromising the device or sniffing the network link. Then, the attacker analyzes the captured packets to infer the behaviors of normal users.

# B LONG-TAILED DISTRIBUTION OF THE ISCX DATASET

We count the flow length on three datasets, i.e., ISCX-VPN, ISCX-NonVPN and ISCX-NonTor datasets and the figure below shows that the flow length generally obeys the long-tailed distribution.

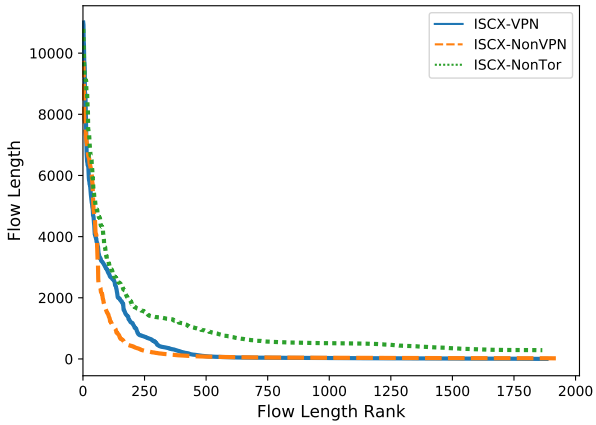



**Figure 4: Long-tailed Distribution of the Flow Length on ISCX-VPN, ISCX-NonVPN and ISCX-NonTor Datasets (Considering the amount of data, only part of the sorted data is shown)**