# $\mathcal{L}$-ETC: A Lightweight Model Based on Key Bytes Selection for Encrypted Traffic Classification

Jie Cao*†‡, Yuwei Xu*†‡§, Qiao Xiang¶

* School of Cyber Science and Engineering, Southeast University, Nanjing, China
† Purple Mountain Laboratories for Network and Communication Security, Nanjing, China
‡Jiangsu Province Engineering Research Center of Security for Ubiquitous Network, Nanjing, China
§ Research Base of International Cyberspace Governance, Southeast University, Nanjing, China
¶ School of Informatics, Xiamen University, Xiamen, China
Email: cao_jie@seu.edu.cn, xuyw@seu.edu.cn (corresponding author), qiaoxiang@xmu.edu.cn

***Abstract*—To protect the confidentiality of communication data, internet users often use encryption protocols (e.g., TLS/SSL) or tools (e.g., VPN, Tor) for network access. Therefore, as a pivotal network management method, encrypted traffic classification technology is vital for guaranteeing the quality of service, the quality of experience, and network security. Researchers have already developed some end-to-end deep learning-based methods to realize encrypted traffic classification. However, given the constrained computational resources available in real-world network measurement scenarios, the existing approaches with high complexity and computation overhead are not appropriate. In this paper, we propose a lightweight model to tackle this issue. Firstly, we propose a base model based on the self-attention mechanism to obtain the key bytes in packets contributing to the classification. Secondly, we leverage these key bytes to reconstruct the input and then streamline the base model, carrying out a lightweight model, i.e., $\mathcal{L}$-ETC. Finally, we implement experiments on three benchmark datasets. $\mathcal{L}$-ETC's macro F1 score of the three tasks exceeds 0.92 with only 0.076M (Million) parameters, and the throughput reaches 917 pps, which is also superior to state-of-the-art methods.**

***Index Terms*—Encrypted traffic classification, self-attention mechanism, key bytes selection, network management**

## I. Introduction

With the rapid development of internet penetration, the amount and diversity of network traffic are constantly emerging, and in order to preserve user privacy, this network traffic is typically obscured and encrypted using various protocols. For instance, according to the survey [1], the global HTTPS page load percentage has been up to 80% or more. Besides, internet users also frequently utilize The Onion Router (Tor) and Virtual Private Networks (VPN) to guarantee the confidentiality of network communications, and these tools encrypt the data leveraging different encryption protocols and algorithms. Therefore, how to effectively manage encrypted traffic is an urgent problem for internet service providers (ISPs), guaranteeing the quality of service (QoS), the quality of experience (QoE), and network security [2]. The most commonly used and widely discussed technique in network traffic management by researchers is encrypted traffic classification (ETC).

Abundant approaches have been proposed to address the ETC problem, mainly including feature-based methods and end-to-end methods. In terms of granularity of classification, the methods can also be classified at the session, flow, and packet levels. The feature-based methods are proposed to extract statistical traffic features relying on expert knowledge, such as packet arrival time and packet size, and cumulative packet sizes [3]. Their advantage is that the expert's prior knowledge can be used to construct feature sets and perform feature selection for a specific task, and finally, a simpler classifier can be used for classification. However, this is also a double-edged sword leading to an over-reliance on expert knowledge, so the selected features set have less generalization and are unsuitable for real-world network scenarios with variable traffic features.

Therefore, some recent studies propose end-to-end methods to overcome the issue of feature-based methods' limited generalizability. End-to-end means directly inputting raw network traffic bytes without feature engineering and fully exploiting the capability of deep learning models to extract latent features in raw traffic bytes automatically. However, the design of end-to-end ETC models have become progressively more sophisticated, ranging from MLP [4], CNN [5], RNN [6], Transformer [7] to BERT [8]. The model size tends to be enormous to pursue a high level of classification accuracy. Nevertheless, in the practical demand for the provision of QoS, the time sensitivity of the network applications puts forward very high requirements. The large-size model brings the defect of slow inference speed to dissatisfy the traffic classification in real-world network deployment.

To address the limitations of the current works, we propose a lightweight end-to-end ETC model $\mathcal{L}$-ETC, whose main contributions are as follows:

- Firstly, we propose a base model based on the self-attention mechanism to represent the relationship and importance of each byte in the packet. This model consists of two components: two embedding layers and a classification neural network.
- Secondly, we propose a key bytes selection algorithm

978-1-5386-7462-8/23/$31.00 ©2023 IEEE

leveraging the self-attention mechanism in the base model's classification neural network to discover the key bytes of packets in a specific ETC task. We reconstruct the input with only selected key bytes and propose the lightweight end-to-end model, $\mathcal{L}$-ETC.
- Thirdly, we empirically compare $\mathcal{L}$-ETC with three state-of-the-art methods on three benchmark datasets. The macro F1 score of $\mathcal{L}$-ETC on three tasks can exceed 0.92. Moreover, $\mathcal{L}$-ETC has 0.076M parameters, which is significantly fewer than the other methods and outperforms them with a throughput of 917 pps.

The rest of this paper is organized as follows. Section II discusses and sums up the related work. Section III introduces three public datasets, then elaborates the preprocessing method and comprehensive workflow of $\mathcal{L}$-ETC. Section IV demonstrates classification results and analyzes $\mathcal{L}$-ETC's complexity and throughput. Section V concludes this paper.

## II. Related Work

This section analyzes the existing works in the ETC field and discusses the motivation of our work.

### A. Feature-based Methods

Researchers extracted features by analyzing the communication patterns of encrypted traffic and constructing a classifier to classify the features. In [3], the authors initially proposed cumulative packet sizes (CUMUL) features to realize the Tor web fingerprinting attack. In [9], the authors also leveraged the CUMUL feature and combined it with packet position to propose a novel web fingerprinting attack. Xu *et al.* [10] extracted path signature features based on packets' length sequence in a session and proposed to use the Random Forest (RF) to classify it. In addition to machine learning (ML)-based, deep learning (DL)-based methods have also been designed by researchers. In [11], the authors creatively introduced the idea of image classification into ETC. They used the packet arrival time and the packet size in one flow to generate an image and then classified this image by the LeNet-5, achieving encrypted traffic classification. Shen *et al.* [12] proposed the traffic interaction graph feature (TIG) to abstract the session and designed a graph neural network to classify the TIG.

### B. End-to-end Methods

The existing end-to-end methods commonly took the raw bytes as the input and constructed a DL-based classifier. In [6] and [7], the authors used the raw bytes of the consecutive $n$ packets of a flow to construct an input vector and proposed a DL-based classifier to identify it. Compared with flow-level methods, packet level is a fine-grained classification and possesses a shorter execution time, which is more suitable for online ETC. In [4], the authors proposed to use the first 1480 bytes to construct the input, and an MLP classifier with limited parameters was proposed. DeepPacket [5] was a method based on 1DCNN, which displayed good performance on both *ISCX-VPN* and *ISCX-Tor* datasets. Xie *et al.* [13] proposed a model based on a CNN model with the self-attention mechanism, and the authors proved that the above 90% accuracy could be obtained by using only the first 50 bytes of a packet. ET-BERT [8] was a novel pre-training and then fine-tuning method in the ETC field. The accuracy of ET-BERT for seven ETC tasks can reach 97% or above.

### C. Motivation

The drawback of feature-based methods is that they have limited generalizability due to their heavy reliance on expert knowledge for feature engineering. Nevertheless, the end-to-end methods are usually proposed with DL-based models, which chronologically prefer stacking complex structures from MLP, CNN, RNN to Transformer. These methods gradually grow more sophisticated, resulting in significant processing overhead for ISPs, whose computing resources are constrained and should be rationally allocated. Hence, the existing works are inappropriate for online classification scenarios, so we propose a lightweight end-to-end ETC model.

## III. Methodology

This section first demonstrates the whole workflow of $\mathcal{L}$-ETC, which is shown in Fig. 1. Then, we propose a preprocessing method and introduce three benchmark datasets. Finally, $\mathcal{L}$-ETC's base model and key bytes selection algorithm are described in detail.

### A. Preprocessing

$\mathcal{L}$-ETC is an end-to-end packet-level classification model. First of all, we need to preprocess the original traffic according to the following steps.

(1) Packet Segmentation: Each captured flow can be denoted as $\{p_1, p_2, \ldots, p_n\}$, where $p_i$ is a single packet with five-tuple *⟨srcIP, dstIP, srcPort, dstPort, Protocol⟩*. Therefore, we utilize `SplitCap` to split all captured flow into a number of single packets.

(2) Filtration: Based on DeepPacket's [5] preprocessing workflow, firstly, we mask *srcIP* and *dstIP* using "0.0.0.0" and *srcPort* and *dstPort* using "0". Secondly, DNS packets, TCP connection establishment, and release-related packets that do not contain sufficient data for classification are eliminated. Thirdly, to provide the same transport layer header length, we use $0x00$ to pad the UDP packet header to 20 bytes.

(3) Truncation: Previous works [13] and [14] have suggested classifying packets using only the first 50 bytes to safeguard the traffic payload's confidentiality, and these methods have yielded promising results. The time overhead of the preprocessing phase can also be significantly decreased by using fewer bytes input, which can also serve as a foundation for constructing a model structure with less complexity. Therefore, $\mathcal{L}$-ETC will pad the packet whose length is less than 50 bytes with $0x00$ or truncate them if they are longer.

After preprocessing, a single packet input sequence $\mathcal{B}_{p_i}$ can be defined as follows.

$$\mathcal{B}_{p_i} = \{b_1, b_2, \ldots, b_{50}\},\ 0x00 \leq b_i \leq 0xff \tag{1}$$

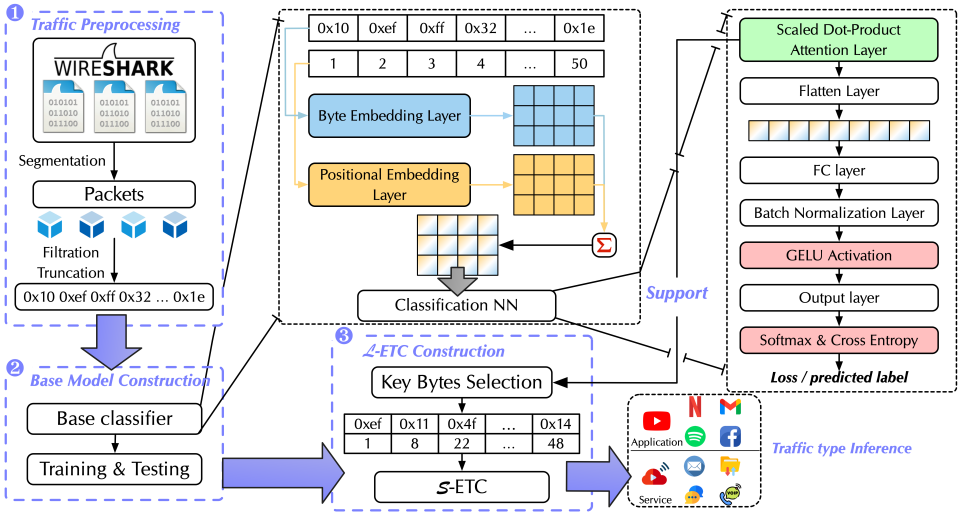


Fig. 1. The structure of $\mathcal{L}$-ETC.

TABLE I
THREE BENCHMARK DATASETS

| Task | Traffic Type | Class | Training set | Validation set | Testing set |
|---|---|---|---|---|---|
| **VPN service classification (T1)** | Chat, Email, File Transfer, Streaming, Voip, VPN: Chat, VPN: File Transfer, VPN: Email, VPN: Streaming, VPN: Torrent, VPN: Voip | 11 | 239185 | 29898 | 29898 |
| **Tor application classification (T2)** | Facebook, Imap, Vuze, Skype, SFTP, ICQ, Spotify, Hangout, Aim, POP, Multiplespeed, FTP, SSL, Youtube, Vimeo, Mam | 16 | 123745 | 15468 | 15468 |
| **Malware classification (T3)** | Worldofwarcraft, Bittorrent, Smb, Outlook, Gmail, Weibo, Mysql, FTP, FaceTime, Skype, Cridex, Nsis-ay, Geodo, Htbot, Tinba, Shifu, Miuref, Zeus, Virut, Neris | 20 | 290104 | 36263 | 36263 |

### B. Datasets

To assist the training and evaluation of $\mathcal{L}$-ETC, we introduce three benchmark datasets within the ETC field, including the *ISCX-VPN* dataset [15], the *ISCX-Tor* dataset [16], and the *USTC-TFC2016* dataset [17]. Then we develop three types of ETC tasks correspondingly, as shown in Table I.

Since there are category imbalance issues in three datasets, in order to obtain the optimal classification results, we use the under-sampling approach to control the number of packets of each category at $30,000$, and if the category is less than $30,000$, all the packets of that category will be selected. Then, the training set, validation set, and testing set will be divided under the ratio $8:1:1$ based on the hold-out method. Table I displays statistical information regarding the packets.

### C. Base Model Construction

To implement $\mathcal{L}$-ETC, we first propose a base model. As shown in Step 2 of Fig. 1, it consists of two components: embedding layers and a classification neural network.

*1) Embedding Layers:* The embedding layers are used to automatically extract the latent features in the packet's raw data and are an effective technique to represent the sequence format input. In this base model, we design a byte embedding layer and a positional embedding layer to represent $\mathcal{B}_{p_i}$. Towards each $\mathcal{B}_{p_i}$, we covert them using the learnable embedding matrix to obtain byte embedding $BE_{p_i} \in \mathbb{R}^{50\times d}$ as depicted in Eq. (2), where $d$ is the embedding dimension. The knowledge of byte order is essential for traffic classification because different byte positions in the packet have various practical impacts and play vital roles in network communication. Thus, the byte order information is crucial for traffic classification, and the learnable positional embedding layer based on [18] is carried out. Therefore, the positions corresponding to each $\mathcal{B}_{p_i}$ can be denoted as $\mathcal{P}_{p_i} = \{1, 2, \ldots, 50\}$. According to Eq. (3), we obtain the positional embedding $PE_{p_i} \in \mathbb{R}^{50\times d}$. Consequently, the final representation $X_{p_i}$ to packet $p_i$ can be obtained in Eq. (4).

$$BE_{p_i} = \mathcal{B}_{p_i} E^{(1)} \tag{2}$$

$$PE_{p_i} = \mathcal{P}_{p_i} E^{(2)} \tag{3}$$

$$X_{p_i} = BE_{p_i} + PE_{p_i} \tag{4}$$

*2) Classification Neural Network:* The structure of the base model is shown in Fig. 1 right part. To represent the relationship between each byte and then ascertain the contribution of each byte position to the ETC, we introduce the Scaled Dot-Product Attention layer, viz., Self-attention mechanism [18]. It is the first layer to further extract features from $X_{p_i}$ and represents the potential relationship between every byte $b_i$.

The core of self-attention is the construction of Query ($Q$) and Key-Value ($K-V$) pair mapping to represent the relationship between different byte's position in a packet. Specifically,

$Q$, $K$, and $V$ are identical, and are the linear transformations of $X_{p_i}$, which can be calculated by the trainable weights $W^Q$, $W^K$, and $W^V$ in Eq. (5). The formula of Scaled Dot-Product Attention layer is given in Eq. (6).

$$Q = X_{p_i}W^Q; K = X_{p_i}W^K; V = X_{p_i}W^V; W^{Q,K,V} \in \mathbb{R}^{d\times d} \tag{5}$$

$$\textit{Self-Attention}(Q,K,V) = Softmax\left(\frac{QK^T}{\sqrt{d^k}}\right)V \tag{6}$$

Where $d^k$ is the dimension of $K$ and is equal to the embedding dimension $d$. Furthermore, $Softmax(\frac{QK^T}{\sqrt{d^k}})$ can be conceived as a weight coefficient matrix to $V$. After training, it can be viewed as a score function to assess how much each byte contributes to the classification task. As a result, it will be the pivotal basis for implementing the Key Bytes Selection algorithm in Section III-D.

We denote the output of the Scaled Dot-Product Attention layer as $O_{p_i} \in \mathbb{R}^{50\times d}$. It will be flattened to be the fully connected (FC) layer's input, then move through each of the neural network structures we created in turn, including the FC layer, GELU activation function, batch normalization layer, and output layer. Finally, the loss will be calculated by the $Softmax$ function and $Cross\_Entropy$ function. This procedure's formula is shown in Eq. (7).

$$\begin{aligned} O^{(1)} &= Flatten(O_{p_i}) \\ O^{(2)} &= W^{(1)}\cdot O^{(1)} + b^{(1)}; W^{(1)} \in \mathbb{R}^{50d\times h} \\ O^{(3)} &= GELU(BN(O^{(2)})) \\ O^{(4)} &= W^{(2)}\cdot O^{(3)} + b^{(2)}; W^{(2)} \in \mathbb{R}^{h\times|Y|} \\ Loss &= Cross\_Entropy(Softmax(O^{(4)})) \end{aligned} \tag{7}$$

Where $h$ is the number of neurons in the FC layer and $|Y|$ is the number of categories for a specific ETC task. In this paper, the hyper-parameters are selected manually and given in Table II.

TABLE II
HYPER-PARAMETERS SETTING

| Hyper-parameters | Setting |
|---|---|
| Embedding dimension ($d$) | 36 |
| Hidden layer size ($h$) | 128 |
| Training epoch | 15 |
| Batch size | 256 |
| Learning rate | 0.001 |
| Optimizer | Adam |
| Weight initialization | Xavier |

### D. Key Bytes Selection Algorithm and $\mathcal{L}$-ETC Construction

As mentioned previously, the *Self-Attention*$(Q,K,V)$ function can measure the importance of each byte. Towards IP packets, each byte of the header has a predetermined structure and actual network transmission significance. For instance, the first three bytes of the TLS header specify the data type of the TLS payload and the TLS protocol version. Consequently, a relatively accurate classification can be achieved by using only the key $\mathcal{N}$ bytes when the target ETC task is fixed and the traffic pattern does not fluctuate. As depicted in Fig. 2, the base model for **T1** places more emphasis on bytes`#49`, `#29`, and `#2`. However, the key bytes for **T2**, **T3** are not the same as **T1** due to the heterogeneity of network communications embedded in the traffic data in different types of datasets.

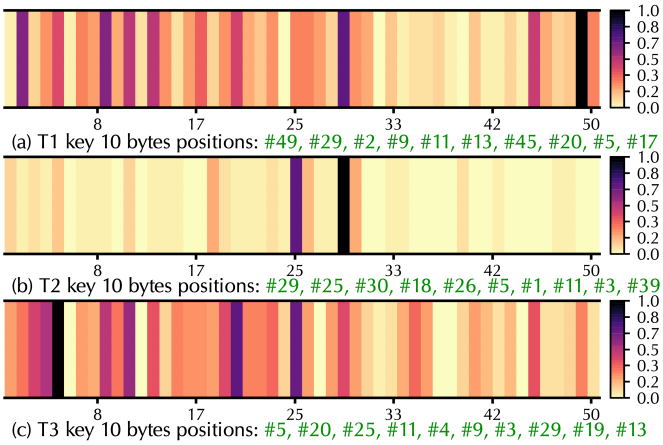


Fig. 2. Visualization of the importance of the first 50 bytes for three tasks.

**Algorithm 1** $\mathcal{L}$-ETC construction

**Input:** Target ETC task training set and validation set $\mathbb{D}_{train\&dev}$, base model $\phi$ trained on $\mathbb{D}_{train}$, the number of key bytes $\mathcal{N}$;
**Output:** $\mathcal{L}$-ETC model $\phi^{\downarrow}$;

**Initialize** 2d array $scores \in \mathbb{R}^{|\mathbb{D}_{train\&dev}|\times 50}$
**Initialize** 1d array $bytescore \in \mathbb{R}^{50}, keybyte\_pos \in \mathbb{R}^{\mathcal{N}}$
**# Key bytes selection**
$Attention \leftarrow get\_function(\phi(\mathbb{D}_{train}))$;
**for** $i \leftarrow 1$ **to** $|\mathbb{D}_{train\&dev}|$ **do**
  $scores[i] \leftarrow Attention(\mathbb{D}_{train\&dev}[i])$;
**end for**
$bytescore \leftarrow \sum_{i=1}^{|\mathbb{D}_{train\&dev}|} scores[i]$;
$keybyte\_pos \leftarrow get\_key\mathcal{N}\_pos(bytescore, \mathcal{N})$;
**# Streamline the base model**
$\phi^{\downarrow} \leftarrow \phi$;
**Lessen** $Flatten$ **in** $\phi^{\downarrow}$ **from** $50d$ **to** $\mathcal{N}d$;
**Lessen** $W^{(1)}$ **in** $\phi^{\downarrow}$ **from** $50d\times h$ **to** $\mathcal{N}d\times h$;
**# Reconstruct the input and train** $\phi^{\downarrow}$
**for** $i \leftarrow 1$ **to** $|\mathbb{D}_{train}|$ **do**
  $\mathcal{P}_{p_i} \leftarrow keybyte\_pos$;
  **for** $j \leftarrow 1$ **to** $\mathcal{N}$ **do**
    $\mathcal{B}_{p_i}[j] \leftarrow \mathbb{D}_{train}[i][\mathcal{P}_{p_i}[j]]$;
  **end for**
  `# Train` $\phi^{\downarrow}$
  **Train** $\phi^{\downarrow}$ **with the input of** $\mathcal{B}_{p_i}$ and $\mathcal{P}_{p_i}$ **via** Eq. (2)-(7);
**end for**
**return** $\phi^{\downarrow}$

With this in mind, we propose the Key Bytes Selection algorithm based on the training set and validation set of each ETC task to select $\mathcal{N}$ key bytes of a specific task, then further streamline the base model proposed in Section III-C and obtain

the final $\mathcal{L}$-ETC. This workflow is described in Step 3 of Fig. 1 and **Alg. 1**. By selecting the top $\mathcal{N}$ key bytes with their corresponding positions, we can reduce the input length to $\mathcal{N} = |\mathcal{B}_{p_i}| = |\mathcal{P}_{p_i}| < 50$. Meanwhile, from the perspective of classifier construction, this algorithm assists us to reduce the Flatten layer's size from $50d$ down to $\mathcal{N}d$ and decrease hidden layer $W^{(1)}$'s size from $50d \times h$ down to $\mathcal{N}d \times h$ without changing the other part of base model structure. After all, the final lightweight model $\mathcal{L}$-ETC will be obtained, and we effectively control the model size and reduce the time overhead of classification without significantly reducing the classification accuracy.

## IV. Experiment and Analysis

In this section, we first discuss the experimental setup and metrics, then we evaluate the effectiveness of $\mathcal{L}$-ETC with three state-of-the-art methods.

### A. Experiment Setup

$\mathcal{L}$-ETC is deployed and estimated based on a computer equipped with Intel(R) Core(TM) i9 10900K CPU with 64G RAM, an NVIDIA RTX3080 GPU, and Ubuntu 20.04.2 LTS operating system. Moreover, the deep learning framework used in this paper is Pytorch 1.9.

Furthermore, we introduce three state-of-the-art packet-level ETC methods, and their details are as follows. Moreover, all hyper-parameter settings are consistent with the original paper.

- DeepPacket [5]: This method normalized each byte in the packet to $[0, 1]$, getting an input of $1 \times 1500$, and proposed a classification model using a 1DCNN network.
- SAM [13]: This method normalized each byte in the packet to $[0, 255]$ and just leveraged the first 50 bytes, getting a matrix input of $1 \times 50$, and proposed a classification model using a self-attention-1DCNN network.
- ET-BERT [8]: This method used a 2-byte string as a unit, like $0xff0x23$, then constructed a text-like sequence. The structure of this method is similar to BERT in NLP.

To evaluate the classification performance of different methods, we use four metrics, i.e., accuracy (AC), macro average Precision (PR), Recall (RC), and F1 score.

### B. Top $\mathcal{N}$ Key Bytes Selection

To choose the concrete $\mathcal{N}$, we performed experiments on the relationship between $\mathcal{N}$ and Macro F1. $\mathcal{L}$-ETC performs well when $\mathcal{N}$ is only equal to 15, and its F1 can approach $94\%$ on three tasks, as illustrated in Fig. 3(a). To highlight the effectiveness of key bytes, we also randomly selected $\mathcal{N}$ bytes from the first 50 bytes for three tasks, and the results in Fig. 3(b) prove that the randomly selected $\mathcal{N}$ bytes cannot achieve acceptable results at all. It experimentally demonstrates that a specific ETC task depends on several key bytes and their corresponding position information. Therefore, this paper specifies $\mathcal{N} = 15$ to construct $\mathcal{L}$-ETC and conducts a subsequent discussion.

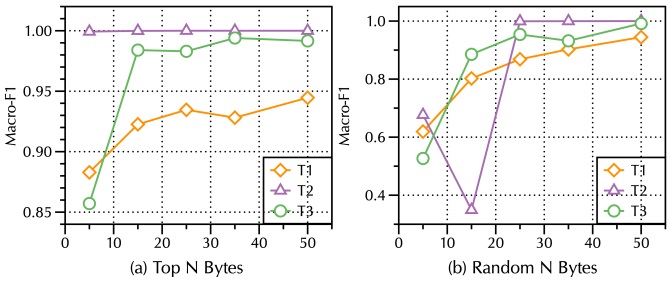


Fig. 3. $\mathcal{L}$-ETC $\mathcal{N}$ bytes selection results in three Tasks.

### C. Performance Analysis

Experiment results for three ETC tasks are shown in Table III. Towards **T1**, $\mathcal{L}$-ETC is better than DeepPacket and SAM, and its four metrics all reach above 0.91. Nevertheless, it needs to be acknowledged that the accuracy of $\mathcal{L}$-ETC is $5.6\%$ less than that of ET-BERT on **T1**, owing to ET-BERT's complex pre-training and fine-tuning strategy and huge model size. In **T2**, SAM, ET-BERT, and $\mathcal{L}$-ETC perform better since Tor traffic has a more pronounced anonymous communication feature. In **T3**, the results of $\mathcal{L}$-ETC and ET-BERT are close, and $\mathcal{L}$-ETC's accuracy and F1 both can exceed 0.98.

The primary benefit and intention of $\mathcal{L}$-ETC is to propose a lightweight model to cater to the real-world network traffic management device. Because these devices generally do not employ high-performance GPU, the large size and complex structure model cannot be trained online. However, the number of parameters of $\mathcal{L}$-ETC is only 0.076M, as shown in Table IV, which is 65% lower than the base model. Furthermore, $\mathcal{L}$-ETC also possesses the minimum weight file size, only 310KB. These demonstrate that our Key Bytes Selection algorithm can simplify the base model without causing a significant decrease in model accuracy. Compared with ET-BERT, which has the best classification performance, it is worthwhile to sacrifice some classification performance in exchange for a more lightweight model structure.

In addition, we fixed the batch size for all methods to 256 and the epoch to 15 with the exception of ET-BERT, as depicted in Fig. 4(a). We test the training times of methods

TABLE III
Comparison Results on Three ETC Tasks

| Task | **T1** | | | | **T2** | | | | **T3** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | AC | PR | RC | F1 | AC | PR | RC | F1 | AC | PR | RC | F1 |
| DeepPacket [5] | 0.8911 | 0.9031 | 0.8756 | 0.8793 | 0.9739 | 0.9186 | 0.8993 | 0.9069 | 0.9425 | 0.9137 | 0.9413 | 0.9185 |
| SAM [13] | 0.8987 | 0.9089 | 0.9074 | 0.9075 | 0.9986 | 0.9989 | 0.9991 | 0.9990 | 0.8987 | 0.9240 | 0.9199 | 0.9196 |
| ET-BERT [8] | 0.9724 | 0.9707 | 0.9699 | 0.9702 | 1.0000 | 1.0000 | 1.0000 | 1.0000 | 0.9977 | 0.9980 | 0.9979 | 0.9979 |
| Base model | 0.9446 | 0.9488 | 0.9486 | 0.9487 | 0.9999 | 1.0000 | 0.9999 | 1.0000 | 0.9915 | 0.9905 | 0.9928 | 0.9916 |
| $\mathcal{L}$-ETC | 0.9172 | 0.9258 | 0.9244 | 0.9245 | 0.9999 | 0.9999 | 1.0000 | 0.9999 | 0.9899 | 0.9924 | 0.9780 | 0.9840 |

based on **T1**. The results illustrate that $\mathcal{L}$-ETC takes the shortest training time overhead and DeepPacket takes up to 6 minutes. When the batch size is set to 16 and the GPU is fully utilized, ET-BERT takes 4 hours. Therefore, DeepPacket and ET-BERT cannot to follow fresh traffic data and iterate appropriately. $\mathcal{L}$-ETC has the fastest training speed, so it is more suitable for local training scenarios. The above results indicate that $\mathcal{L}$-ETC has lightweight and flexible features that can be effectively deployed on ISP devices without consuming large computational resources.

The lightweight model structure can increase the ETC method's throughput and alleviate ETC's impact on the network transmission delay. According to [7], we define the throughput as follows:

$$Throughput = \frac{1}{PT + CT} \tag{8}$$

Where $PT$ is the preprocessing time (s) per packet, and $CT$ is the classification time (s) per packet. In our experiment, we test all methods' throughput with $10,000$ randomly selected packets each time and repeat ten times, as shown in Table IV and Fig. 4(b). The average throughput of $\mathcal{L}$-ETC can reach 917 packets per second (pps), outperforming other methods, and is 9 times as much as ET-BERT. Thus, $\mathcal{L}$-ETC can help ISPs minimize the impact of the traffic classification process on network latency, ensuring the QoS.

TABLE IV
MODEL COMPLEXITY AND PERFORMANCE COMPARISON

| Method | Parameters | File size (*.ckpt) | Avg Throughput |
|---|---|---|---|
| DeepPacket | 10M | 40.4MB | 652 pps |
| SAM | 0.8M | 3.3MB | 686 pps |
| ET-BERT | 134M | 505MB | 102 pps |
| Base model | 0.22M | 0.9MB | 852 pps |
| $\mathcal{L}$-ETC | 0.076M | 310KB | 917 pps |

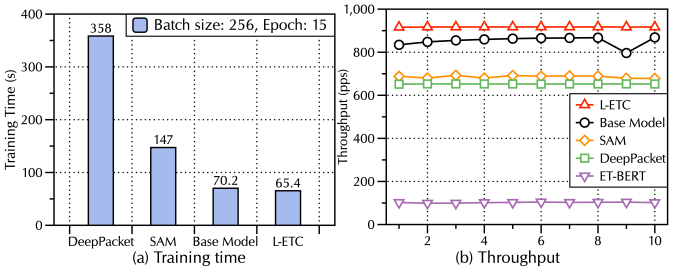


Fig. 4. Training time and throughput of different methods.

## V. CONCLUSION

In this paper, we proposed $\mathcal{L}$-ETC, a lightweight end-to-end model for encrypted traffic classification. $\mathcal{L}$-ETC leverages the self-attention mechanism to realize key bytes selection, then reconstructs the input and reduces the model size markedly. We conduct comprehensive experiments on three encrypted traffic datasets. Experimental results illustrate that $\mathcal{L}$-ETC's F1 can reach 0.92. Besides, the number of parameters of $\mathcal{L}$-ETC is only 0.076M, which is extremely less than three state-of-the-art methods. At the same time, $\mathcal{L}$-ETC offers a fast training speed and high throughput for online classification.

## ACKNOWLEDGMENT

This work was supported in part by the National Key R&D Program of China under Grant 2020YFB1005500, and in part by the Fundamental Research Funds for the Central Universities, Southeast University.

## REFERENCES

[1] E. Papadogiannaki and S. Ioannidis, "A survey on encrypted network traffic analysis applications, techniques, and countermeasures," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–35, 2021.
[2] X. Yun, Y. Wang, Y. Zhang, C. Zhao, and Z. Zhao, "Encrypted tls traffic classification on cloud platforms," *IEEE/ACM ToN*, 2022.
[3] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.
[4] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55 380–55 391, 2018.
[5] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.
[6] H. Yao, C. Liu, P. Zhang, S. Wu, C. Jiang, and S. Yu, "Identification of encrypted traffic through attention mechanism based long short term memory," *IEEE Transactions on Big Data*, 2019.
[7] J. Cheng, Y. Wu, E. Yuepeng, J. You, T. Li, H. Li, and J. Ge, "Matec: A lightweight neural network for online encrypted traffic classification," *Computer Networks*, vol. 199, p. 108472, 2021.
[8] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 633–642.
[9] M. Shen, Y. Liu, L. Zhu, X. Du, and J. Hu, "Fine-grained webpage fingerprinting using only packet length information of encrypted traffic," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2046–2059, 2020.
[10] S.-J. Xu, G.-G. Geng, X.-B. Jin, D.-J. Liu, and J. Weng, "Seeing traffic paths: Encrypted traffic classification with path signature features," *IEEE TIFS*, 2022.
[11] T. Shapira and Y. Shavitt, "Flowpic: A generic representation for encrypted traffic classification and applications identification," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1218–1232, 2021.
[12] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, "Accurate decentralized application identification via encrypted traffic analysis using graph neural networks," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2367–2380, 2021.
[13] G. Xie, Q. Li, and Y. Jiang, "Self-attentive deep learning method for online traffic classification and its interpretability," *Computer Networks*, vol. 196, p. 108267, 2021.
[14] O. Barut, Y. Luo, P. Li, and T. Zhang, "R1dit: Privacy-preserving malware traffic classification with attention-based neural networks," *IEEE Transactions on Network and Service Management*, 2022.
[15] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.
[16] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, A. A. Ghorbani *et al.*, "Characterization of tor traffic using time based features." in *Proceedings of the 3rd international conference on information systems security and privacy (ICISSP)*, 2017, pp. 253–262.
[17] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International conference on information networking (ICOIN)*. IEEE, 2017, pp. 712–717.
[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.