PERT: PAYLOAD ENCODING REPRESENTATION FROM TRANSFORMER FOR ENCRYPTED TRAFFIC CLASSIFICATION

Hong Ye He[1]; Zhi Guo Yang[1]; Xiang Ning Chen[1]

[1]Zhongxing Telecommunication Equipment (ZTE) Corporation

ABSTRACT

Traffic identification becomes more important yet more challenging as related encryption techniques are rapidly developing nowadays. In difference to recent deep learning methods that apply image processing to solve such encrypted traffic problems, in this paper, we propose a method named Payload Encoding Representation from Transformer (PERT) to perform automatic traffic feature extraction using a state-of-the-art dynamic word embedding technique. Based on this, we further provide a traffic classification framework in which unlabeled traffic is utilized to pre-train an encoding network that learns the contextual distribution of traffic payload bytes. Then, the downward classification reuses the pre-trained network to obtain an enhanced classification result. By implementing experiments on a public encrypted traffic data set and our captured Android HTTPS traffic, we prove the proposed method can achieve an obvious better effectiveness than other compared baselines. To the best of our knowledge, this is the first time the encrypted traffic classification with the dynamic word embedding alone with its pre-training strategy has been addressed.

Keywords – Deep learning, dynamic word embedding, encrypted traffic classification, natural language processing, traffic identification

1. INTRODUCTION

Traffic classification, a task to identify certain categories of network traffic, is crucial for Internet services providers (ISP) to track the source of network traffic, and to further ensure their quality of service (QoS). Also, traffic classification is widely applied in some specific missions, like malware traffic identification and network attack detection. However, this is a challenge since network traffic nowadays is more likely to be hidden with several encryption techniques, making detection hard with a traditional approach.

Typically, there are the following widely applied traffic classification methods: 1) The port-based method which simply identifies traffic data using specific port numbers. It is susceptible to the port number changing and port disguise. 2) Deep packet inspection (DPI), a method which aims to locate patterns and keywords from traffic packets, is not suitable for identifying encrypted traffic because it heavily relies on unencrypted information. 3) The machine learning (ML) based method focuses on using manually designed traffic statistical features to fit a machine learning model for categorization [1]. 4) The deep learning (DL) based method is an extension of the ML-based approach where neural networks are applied for automatic traffic feature extraction.

Although encrypted traffic packets are hard to identify, an encrypted traffic flow (a flow is a consecutive sequence of packets with the same source IP, source port, destination IP, destination port and protocol) is still analyzable because the first few packets of a flow may contain visible information like handshake details [2]. In this way, the ML-based and DL-based methods are considered ideal for encrypted traffic classification since they both extract common features from the traffic data. In fact, the ML-based and DL-based methods share the same concept that traffic flows could be vectorized for supervised training according to their feature extraction strategy.

Rather than extracting hand-designed features from the traffic as the ML-based method does, the DL-based method uses a neural network to perform representation learning (RL) for the traffic bytes which allow it to avoid complex feature engineering. It provides an end-to-end solution for encrypted traffic classification where the direct relationship between raw traffic data and its categories is learned. The classification effect of a DL-based method is highly related to its capacity of representation learning.

In this paper, we propose a new DL-based solution named Payload Encoding Representations from Transformers (PERT) in which a dynamic word embedding technique called the Bidirectional Encoder Representations from Transformers (BERT) [3] is applied during the traffic representation learning phrase. Our work is inspired from the great improvements in the natural language processing (NLP) domain that dynamic word embedding brings. We believe that computer communication protocols and natural language have some common characteristics. According to this point, we shall prove that such a strong embedding technique can also be applied to encode traffic payload bytes and provide substantial enhancement while addressing the encrypted traffic classification task.

2. RELATED WORKS

We shall introduce some related traffic identification works that involve the DL domain, and further categorize them as two major groups.

For feature-engineering: Basically, these methods still use hand-designed features but utilize the DL as a measure of feature processing. For example, Niyaz et al. proposed an approach using the deep belief network (DBN) to make a feature selection before the ML classification [4]. Hochst et al. introduced the auto-encoder network to perform dimension reduction for the manually extracted flow features [5]. Rezaei et al. applied a similar pre-training strategy as we do [6]. What is different is that this work introduced neural networks to reconstruct time series features. Its pre-training plays a role of re-processing the hand-designed features. Ours instead, is to perform a representation learning for the raw traffic.

For representation learning: Works similar to ours that apply the DL to learn the encoding representation from raw traffic bytes without manual feature-engineering. These works are also considered as end-to-end implements of traffic classification. Wang et al. proposed this encrypted traffic classification framework for the first time [7]. They transformed payload data to grayscale images and applied convolutional neural networks (CNN) to perform image processing. Afterward, the emergence of a series of CNN-based works like [8] proved the validity of such an end-to-end classification. Lopez-Martin et al. further discussed a possible combination for traffic identification where the CNN is still used for representation learning, but a long short-term memory (LSTM) network is introduced to learn the flow behaviors [9]. It inspired the hierarchical spatial-temporal features-based (HAST) models which obtained a state-of-the-art result in the intrusion detection domain [10].

Nevertheless, for end-to-end encrypted traffic classification nowadays, CNN is still the mainstream whereas the NLP-related network only works as an assistance to do jobs such as capturing flow information. We can hardly find a full-NLP scheme similar to ours, let alone one which applies current dynamic word embedding techniques.

3. MODEL ARCHITECTURE

3.1 Payload Tokenization

According to [2], the payload bytes of a packet are likely to expose some visible information, especially for the first few packets of a traffic flow. Thus, most DL-based methods use this byte data to construct traffic images as the inputs of a CNN model. This is because the byte data is ideal for generating pixel images as its value ranges from 0 to 255, which is just fit for a grayscale image. Rather than applying such an image processing strategy, we treat the payload bytes of a packet as a language-like string for introducing NLP processing.

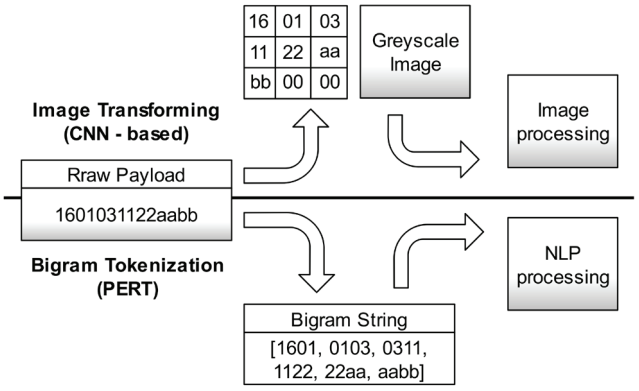

Figure 1 – Comparison of raw payload processing between CNN-based methods and PERT

However, the range of byte value is rather small considering the size of a common NLP vocabulary. To extend the vocab size of traffic bytes, we introduce a tokenization which takes pairs of bytes (value range from 0 to 65535) as basic character units to generate the bigram strings, as illustrated in Figure 1. Afterward, the NLP related encoding methods can be directly applied to the tokenized traffic bytes. Thus, the encrypted traffic identification is transformed to a NLP classification task.

3.2 Representation Learning

While performing representation learning in an NLP task, the word embedding is widely utilized. Recently, a breakthrough was made in this research area as the dynamic word embedding technique overcame the drawback that traditional word embedding methods like the Word2Vec [11] are only capable of mapping words to unchangeable vectors. By contrast, vectors trained by dynamic word embedding can be adjusted according to its context inputs, making it more powerful to learn detailed contextual information. This is just what we need for extracting complex contextual features from the encrypted traffic data.

Current popular dynamic word embedding like BERT could be considered as a stack of a certain type of encoding layers. Each encoder takes the outputs of its former layer as inputs and further learns a more abstract representation. In another word, word embedding will be dynamically adjusted while passing through its next encoding layer.

In our work, we take the tokenized payload string $[w_1, w_2, ..., w_k]$ as our original inputs. The first group of word embedding vectors $[\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_k]$ at the bottom of the network are randomly initialized. After N times of dynamic encoding, we obtain the final word embedding outputs $[\boldsymbol{h}^N_1, \boldsymbol{h}^N_2, ..., \boldsymbol{h}^N_k]$ that imply extremely abstract contextual information of the original payload.

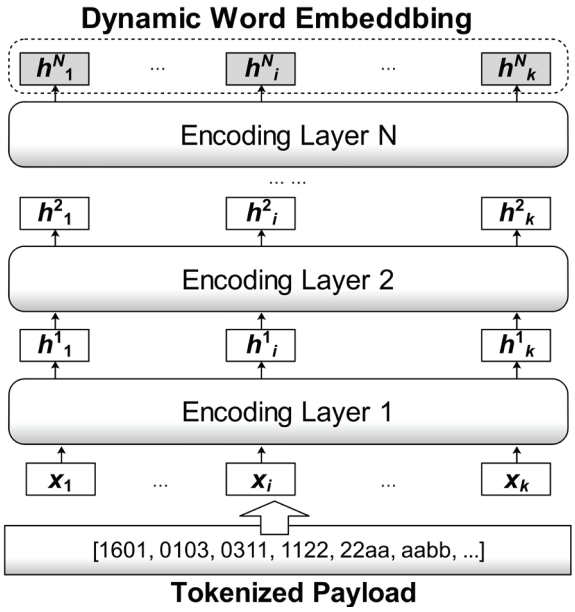

Figure 2 – Representation learning network for payload data using the dynamic word embedding architecture

The illustration of our representation learning is shown in Figure 2.

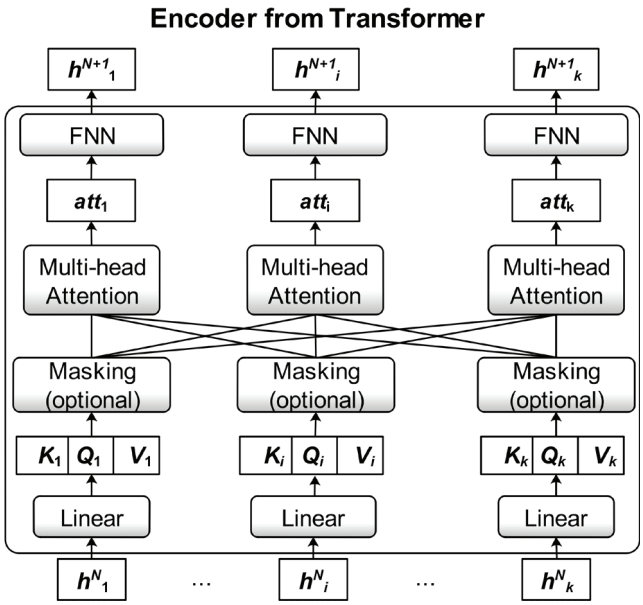

Figure 3 – Detail of the encoding layer

Earlier dynamic word embedding called the Embeddings from Language Models (Elmo) [12] uses the bidirectional LSTM as its encoder unit, which is not suitable for large-scale training since the LSTM has a bad support for parallel calculations. To solve this problem, [3] replaced the LSTM with a self-attention encoder that is firstly applied in the transformer model [13], and named their embedding model as BERT. This is what we also use for encoding the encrypted payload as shown in Figure 3. Taking our first embedding vectors [x_1, x_2, ..., x_k] as examples, there are several steps of the transformer encoding as follows:

Linear projections: Each embedding vector x_i will be projected to three vectors using linear transformations:

$$\begin{aligned} K_i &= W^K x_i \\ Q_i &= W^Q x_i \\ V_i &= W^V x_i \end{aligned} \qquad (1)$$

Where W^K, W^Q and W^V are the three groups of linear parameters.

Self-attention and optional masking: The purpose of linear projections is to generate the inputs for the self-attention mechanism. Generally speaking, self-attention is to figure the compatibility between each input x_i and all the other inputs x_1~x_k via a similarity function, and further to calculate a weighted sum for x_i which implies its overall contextual information. In detail, our self-attention is calculated as follows:

$$att_i = \sum_{j=1}^{k} \frac{\frac{Q_i K_j^T}{\sqrt{d_k}}}{Z} \times V_j \qquad (2)$$

The similarity between x_i and x_j is figured by a scaled dot-product operator, where d_k is the dimension of K_j, and Z is the normalization factor. It should be noticed that not every input vector is needed for self-attention calculation. An optional masking strategy that randomly ignores a few inputs while generating attention vectors is allowed to avoid the over-fitting.

Multi-head attention: In order to grant encoders the ability of reaching more contextual information, the transformer encoding applies a multi-head attention mechanism. Specifically, linear projections will be done for M times for each x_i to generate multiple attention vectors [$att_{i,1}$, $att_{i,2}$, ..., $att_{i,M}$]. Afterward, a concatenation operator is utilized to obtain the final attention vector:

$$att_i = att_{i,1} \oplus att_{i,2} \oplus \ldots \oplus att_{i,M} \qquad (3)$$

Feed-forward network (FFN): A full-connected network to provide the output of current encoder. For x_i, it is as follows:

$$h_i = max(0, W_1 att_i + b_1) + b_2 \qquad (4)$$

W_1, b_1, W_2 and b_2 are the full-connection parameters and max(0, x) is a *ReLU* activation.

Finally, we get the dynamic embedding h_i which is encoded from x_i. It can be further encoded by the next encoding layer or be directly used in downward tasks. Similar to the naming of BERT, we name our encoding network as the Payload Encoding Representation from Transformer (PERT) considering the application of a transformer encoder.

3.3 Packet-level Pre-training

A key factor that makes BERT and its extensive models continuously achieve state-of-the-art results among a wide

range of NLP tasks is their "pre-training + fine-tuning" strategy. To the best of our knowledge, our work is the first to introduce such a strategy to an end-to-end encrypted traffic classification architecture. To perform pre-training is to initialize the encoding network and to give it the ability of contextual information extraction before it is applied to downward tasks. The unsupervised language model (LM) is widely used for word embedding pre-training [14]. BERT, specifically, proposed a masked LM which hides several words from the original string with a unique symbol '*unk*', and uses the rest of the words to predict those hidden ones.

To demonstrate the procedure of the masked LM, we give a masked traffic bigram string as $\boldsymbol{w} = [w_1, ..., \text{'}unk\text{'}, ..., \text{'}unk\text{'}, ..., w_k]$ and a list $\boldsymbol{msk} = [i_1, i_2, ..., i_m]$ which indicates the position of bigram units that are masked. After the encoding, for each embedding vector $\boldsymbol{h_i}$ that are encoded from the i-th position of the original input, a full connection is followed:

$$\boldsymbol{o_i} = \boldsymbol{W'} tanh(\boldsymbol{W h_i} + \boldsymbol{b}) + \boldsymbol{b'} \qquad (5)$$

Where *tanh* is an activation function like the *Relu*. The size of the output vector $\boldsymbol{o_i} = [o_{i,1}, ..., o_{i,|V|}]$ is the vocab size $|V|$. It stores all the likelihoods about what the corresponding traffic bigram is at the i-th position.

In the end, the masked LM uses partial outputs $\{\boldsymbol{o_i}, i \in \boldsymbol{msk}\}$ to perform a large softmax classification with the class number of vocab size. The objective is to maximize the predicted probabilities of all the masked bigrams, which can be simply written as (θ represents the parameters of the entire network):

$$argmax_\theta \sum_i^{i \in \boldsymbol{msk}} P(w_i | \boldsymbol{o_i}, \theta) \qquad (6)$$

The LM is considered as a powerful initialization approach for the encoding network using large-scale unlabeled data, yet it is very time-consuming. Even if we want to perform a flow-level classification for encrypted traffic, we argue that the pre-training should be packet-level considering the possible calculation costs. Particularly, we collect raw traffic packets from the Internet despite their sources and extract their payload bytes to generate an unsupervised data set. Then, the extracted payload bytes are tokenized as bigram strings and are utilized to perform a PERT pre-training. After the training converges, we save the adjusted encoding network.

3.4 Flow-level Classification

While implementing a certain task like classification, the pre-trained encoding network will be totally reused and be further adjusted to learn the real relationship between the inputs and a specific task objective. This is the concept of "fine-tuning", where a network is trained based on a proper initialization to achieve a boosted effect in downward tasks.

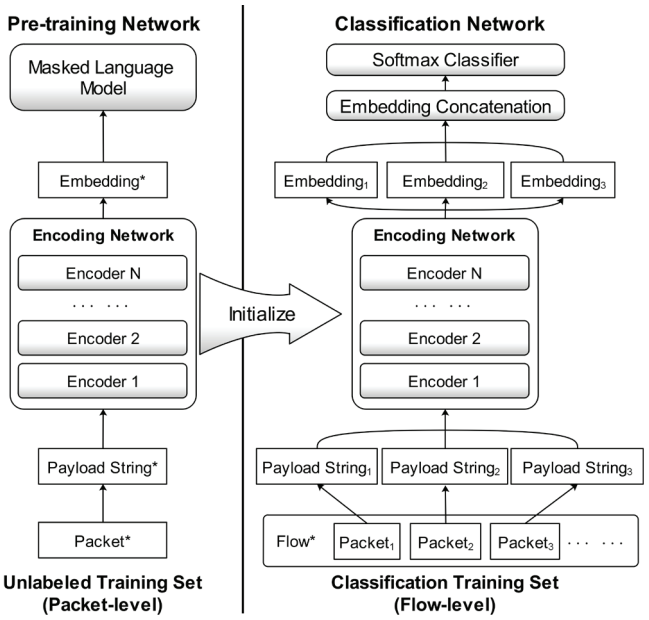

Figure 4 – Illustration of the flow-level encrypted traffic classification initialized with a packet-level pre-training

Figure 4 shows our encrypted traffic classification framework. Below are the detailed descriptions:

Packets extraction: While classifying an encrypted traffic flow, only the first M packets (3 for example in Figure 4) need to be used. The bigram tokenization is performed to payload bytes in each packet to generate a list of tokenized payload strings $[\boldsymbol{str_1}, \boldsymbol{str_2}, ..., \boldsymbol{str_M}]$.

Encoding for packets: Before classification, initialize the encoding network of the classifier with the pre-trained counterpart. As the encoding network is packet-level, each tokenized string will be individually transported to the encoders. According to [3], while carrying out a classification with BERT, a unique token '*cls*' should be added at the beginning of the input as the classification mark. For i-th packet, its tokenized string will be modified as $\boldsymbol{str_i'} = [cls, w_{i,1}, w_{i,2}, ..., w_{i,k}]$. After encoding, a series of embedding vectors $[\boldsymbol{h^N_{i,CLS}}, \boldsymbol{h^N_{i,1}}, ..., \boldsymbol{h^N_{i,k}}]$ is outputted, yet only the $\boldsymbol{h^N_{i,CLS}}$ will be picked as the further classification input. We simply represent $\boldsymbol{h^N_{i,CLS}}$ as $\boldsymbol{emb_i}$. In order to make use of all of the information extracted from the first M packets, we apply a concatenation to merge the encoded packets:

$$\boldsymbol{emb} = \boldsymbol{emb_1} \oplus \boldsymbol{emb_2} \oplus ... \oplus \boldsymbol{emb_M} \qquad (7)$$

Final classification: In the end, a softmax classification layer is used to learn the probability distribution of the input flows among possible traffic classes. The objective of the flow-level classification can be written as below:

$$argmax_\theta \sum_f^{f \in \boldsymbol{R_{flow}}} P(y_f | \boldsymbol{emb}(f), \theta) \qquad (8)$$

Where $\boldsymbol{R_{flow}}$ represents the flow-level training set. Given a flow sample f, y_f represents its true label (class) and $\boldsymbol{emb}(f)$

indicates its concatenated embedding. *P* is the conditional probability which the softmax layer provides. In a matter of speaking, the objective is to maximize the probability that each encoded flow sample is predicted as its corresponding category. The flow-level information is involved in the final softmax classifier, and thus will be used to fine-tune the packet-level encoding network during the back propagation. The main point of such a fine-tuning strategy is to separate the learning of the packets relationship from the time-consuming pre-training procedures.

4. EXPERIMENTS

4.1 Experiment Settings

4.1.1 Data sets

Unlabeled Traffic Data: The data set that is utilized for the pre-training of our PERT encoding network. To generate this data set, we capture a large amount of raw traffic data from different sources using different devices through a network sniffer. Typically, there is no special requirement for the unlabeled traffic data except you should make sure your collected samples can cover the mainstream protocols, as many as possible.

ISCX Data Set: We chose a popular encrypted traffic data set "ISCX2016 VPN-nonVPN"[1] [15] to make our classification evaluations more persuasive. However, this data set only marks where its encrypted traffic data is captured from and whether the capturing is through a VPN session or not, which means a further labeling should be performed. The ISCX data set is utilized in several works yet the results are rather different even when the same model is applied [7],[8]. This is mainly due to how the raw data is processed and labeled. We only found [7] provided their pre-processing and labeling procedures in their github[2]. In this way, we follow this open source project to process the raw ISCX data set and label it with 12 classes.

Android Application Data Set: We find the ISCX data set is not entirely encrypted as it also contains data of some unencrypted protocols like the DNS. To make a better evaluation, in this work, we manually capture traffic samples from 100 Android applications via the Android devices and network sniffer tool-kit. All the captured data belongs to the top activated applications of the Chinese Android app markets. Afterward, we exclusively pick the HTTPS flows to ensure only the encrypted data remains.

4.1.2 Parameters

Pre-training: First of all, to perform the packet-level PERT pre-training for our unlabeled traffic data, we introduce public Python library transfomers[3] which provide implements of the original BERT model and several recently published modified models. In practice, we chose the optimized BERT implement named A Lite BERT (ALBERT) [16] which is more efficient and less resource-consuming. However, even to be properly optimized, current BERT pre-training is very costly that we use 4 Nvidia Tesla P100 GPU cards.

Table 1 – Pre-training parameter settings

Parameter	Value	Description
hidden_size	768	Vector size of the encoding outputs (embedding vectors).
num_hidden_layers	12	Number of encoders used in the encoding network.
num_attention_heads	12	Number of attention heads used in the multi-head attention mechanism.
intermediate_size	3072	Size of the hidden vectors in the FNN networks.
input_length	128	Amount of tokenized bigrams used in a single packet.

Table 1 shows the settings of our pre-training and corresponding description of each parameter. Such settings refer to what common NLP works with BERT encoding use. After sufficient training, we save the encoding network as a Pytorch[4] format which can be reused in our classification networks. Also, all of our other networks are implemented using the Pytorch.

Table 2 – Classification parameter settings

Parameter	Value	Description
packet_num	alternative (5 by default)	Amount of the first packets in a flow that are chosen.
softmax_hidden	768	Size of the hidden vectors in the softmax layer.
dropout	0.5	The dropout rate of the softmax layer.

Classification: The encoding network used at the classification stage shares strictly the same structure as the pre-trained one. Other settings of the classification layers are shown in Table 2. As fine-tuning the encoding network in a classification task is relatively inexpensive [3], a single GPU card will be just enough.

4.1.3 Baselines

Below are the baseline classification methods we use for comparison:

1 https://www.unb.ca/cic/datasets/vpn.htm

2 https://github.com/echowei/DeepTraffic

3 https://huggingface.co/transformers/

4 https://pytorch.org

ML-based: We refer to [15] to implement our ML-based methods using the decision tree classifier (we named it ML-1). However, it only contains basic flow-statistical features that we consider as not the most optimized ML-based method. Thus, based on ML-1, we further add some time series features as the source ports, destination ports, directions, packet lengths and arrival time intervals of the first 10 packets in a flow to generate the ML-2 model.

CNN: The two types of CNN models that are provided by [7], the 1D-CNN and the 2D-CNN. They both use the first 784 bytes of a traffic flow to perform the classification.

HAST: The two HAST models proposed by [10] are the state-of-art end-to-end methods for intrusion detection. HAST-I uses the first 784 bytes of a flow for direct representation learning. HAST-II, however, only performs packet-level encoding. It further introduces an LSTM to merge the encoded packets.

During the evaluation, we randomly chose 90% of samples from the data set as the the training set, and the remaining 10% for validation. Then, three widely used classification metrics are applied:

$$Precision(P) = \frac{TP}{TP+FP}$$
$$Recall(R) = \frac{TP}{TP+FN} \qquad (9)$$
$$F1-score(F1) = \frac{2\times P\times R}{P+R}$$

Take a class y_i as an example, the TP_i is the amount of samples correctly classified as y_i, FP_i is the amount of samples mistakenly classified as y_i, FN_i is the amount of samples mistakenly classified as nor-y_i. As for the overall evaluation for all classes, we use the average values of those metrics.

4.2 Overall Analysis

Table 3 – Classification results (ISCX data set)

Model	Precision	Recall	F1
ML-1 [15]	0.8194	0.8136	0.8164
ML-2	0.8901	0.8896	0.8898
CNN-1D [7]	0.8616	0.8605	0.8610
CNN-2D [7]	0.8425	0.8420	0.8422
HAST-I [10]	0.8757	0.8729	0.8742
HAST-II [10]	0.8502	0.8427	0.8409
PERT(Ours)	**0.9327**	**0.9322**	**0.9323**

Results on the ISCX data set: This group of experiments are used to discuss the classification based on the consistent data settings of [7]. As we can see in Table 3, our flow-level PERT classification achieves the best classification results where the precision reaches 93.27% and the recall reaches 93.22%. It proves the PERT is a power representation learning method for encrypted traffic classification.

As for other models, using the same manner of data preprocessing, the CNN classification results are pretty close to what is provided by [7]. The CNN methods obviously obtain higher precision and recall than the ML-1 which are implemented based on [15]. However, the ML-1 can still be improved. When the time series features are added, the precision of the ML-2 classification can exceed 89% which is much better than what the basic CNN methods get. In other words, the basic CNN methods actually have no absolute advantage while classifying the ISCX data set.

HAST-I achieves better results than typical CNN models yet HAST-II with an LSTM works relatively worse. In fact, we think using the first few bytes of a flow to perform a direct deep learning (like HAST-I and CNN-1D) is considered better than merging the packet-level encoded vectors, since the representation learning can directly capture flow-level information. However, the encoding costs on a long string are not affordable for complex dynamic word embedding. At the current stage, the "packet-level encoding + flow-level merging" is the best option for our PERT classification.

Table 4 – Classification results (Android data set)

Model	Precision	Recall	F1
ML-1 [15]	/	/	/
ML-2	0.7351	0.7335	0.7321
CNN-1D [7]	0.7709	0.7683	0.7668
CNN-2D [7]	0.7684	0.7659	0.7643
HAST-I [10]	0.8201	0.8185	0.8167
HAST-II [10]	0.7924	0.7813	0.7826
PERT(Ours)	**0.9042**	**0.9003**	**0.9007**

Results on the Android data set: Experiments based on full HTTPS traffic to evaluate the actual encrypted traffic classification ability of each method. As all the data here are HTTPS flows, in comparison with the ISCX data set whose data covers several traffic protocols, it is harder to locate distinctly different flow behaviors among the chosen applications. Consequently, the ML-based methods that strongly rely on flow statistics features work extremely weakly. Even when enhanced by time series features, the ML-2 still obtains a worse result than basic DL methods. As for the original ML-1, we find it is entirely not capable of addressing this 100-class HTTPS classification that we ignore its result in Table 4.

Results on the Android data set demonstrate that the DL-based methods are more suitable for processing full encrypted traffic data. More importantly, our PERT classification again shows its superiority as it introduces a more powerful representation learning strategy. Its F1-score on the 100-class encrypted traffic classification exceeds 90% whereas the HAST can only achieves a result of 81.67%.

4.3 Discussion: Selection of the Packet Number

In a flow-level classification model, the increase in the use of packets will cause significant costs. This is particularly

true when the representation learning is applied to traffic packets.

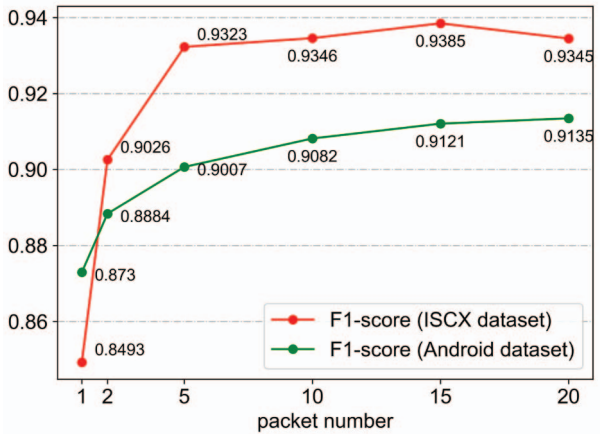

Figure 5 – Illustration of the packet number selection for PERT classification

We performed PERT classifications multiple times on the two data sets with different settings of the "packet_num" and the results are shown in Figure 5. As we can see, at the beginning, the classification result on each data set is greatly improved with more packets used. However its increase is slight after the continuous adding of packets. For example, the F1-score is shown to reach 91.35% while classifying the Android data set with 20 packets. But this result is merely boosted by 1.28% in comparison with using 5 packets. It is not recommended considering the costs of PERT encoding for so many packets and such minor further improvements.

We point out that using 5-10 packets for our PERT classification will be sufficient. Similar conclusions can be also found in other flow-level classification research like [9],[10].

4.4 Discussion: Merging of the Encoded Packets

A major difference between our PERT classification and most flow-level DL-based methods like the HAST-II is how the encoded packets are merged. HAST-II constructs a 2-layer LSTM after encoding the packet data whereas we simply apply a concatenation. To make a comparison between these two approaches, we modify our PERT model and the HAST-II model.

Firstly, we refer to HAST-II and construct the PERT_lstm model by using a 2-layer LSTM to follow our PERT encoded packets. Then, we remove the LSTM layer from HAST-II and further generate the HAST_con by concatenating the HAST encoded packets to fit an ordinary softmax classifier, just as our original PERT model does. For all the compared methods, we consistently select 5 packets for classification based on our former discussion.

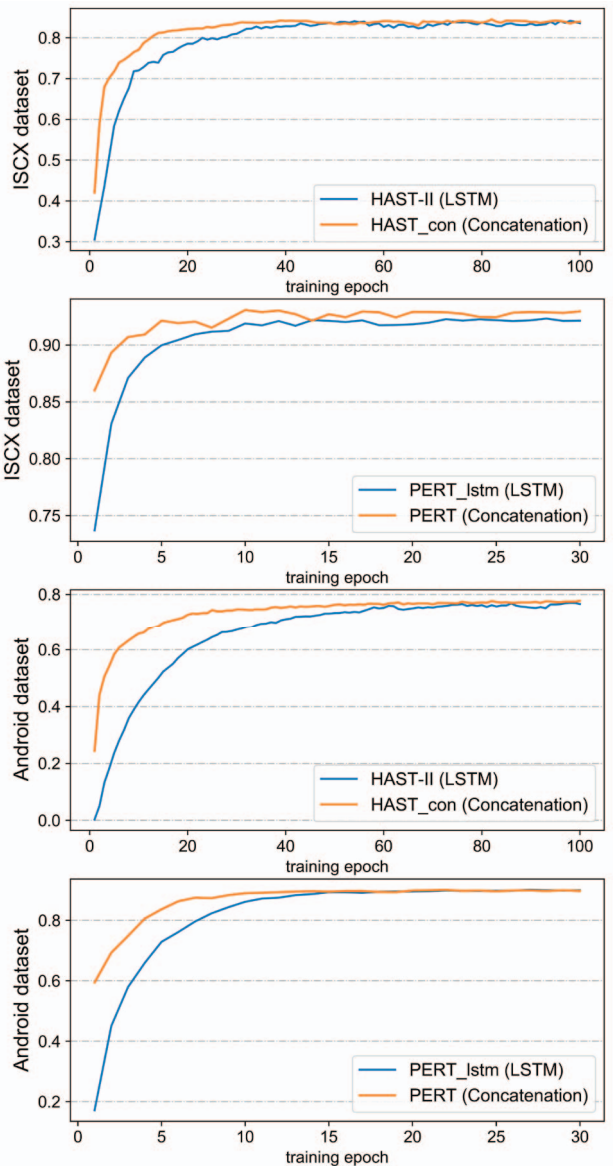

Figure 6 – F1-score converging speed comparison

We perform validation every training epoch for each classification experiment and record corresponding F1-scores for evaluation. As illustrated in Figure 6, we cannot actually tell which merging approach is better for classification accuracy. Whether using the concatenation or the LSTM approach for merging, it does not have a major influence in the final classification results.

However, using different merging approaches will have an obvious impact on the converging speed of the classification training. In Figure 6, it always takes less training rounds before the model converges while introducing the concatenation merging. We believe the LSTM is not a satisfactory option for merging the encoded packets as applying a simple concatenation can reach a very close classification result, yet it is much faster.

5. CONCLUSION

After a thorough analysis of the possibility of applying the full-NLP scheme for encrypted traffic classification, we point out that the byte data of raw traffic packets can be transformed to character strings by proper tokenization. Based on this, we propose a new method named PERT to encode the encrypted traffic data and to serve as an automatic traffic feature extractor. In addition, we discuss the pre-training strategy of dynamic word embedding in a condition of the flow-level encrypted traffic classification. In accordance with a series of experiments on the public ISCX data set and Android HTTPS traffic, our proposed classification framework can provide significantly better results than current DL-based methods and traditional ML-based methods.

REFERENCES

[1] P. Velan, M. Cermak, P. Celeda, et al., "A Survey of Methods for Encrypted Traffic Classification and Analysis," International Journal of Network Management, 2015.

[2] S. Rezaei and X. Liu, "Deep Learning for Encrypted Traffic Classification: An Overview," IEEE Communications Magazine, vol. 57, no. 5, pp. 76-81, 2019.

[3] J. Devlin, M.W. Chang, K. Lee, et al., "BERT: Pre-training of deep bidirectional transformers for language understanding," Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019.

[4] A.Y. Javaid, Q. Niyaz, W. Sun , et al., "A Deep Learning Approach for Network Intrusion Detection System," 9th EAI International Conference on Bio-inspired Information and Communications Technologies. ICST., 2015.

[5] J. Hochst, L. Baumgartner, M. Hollick, et al., "Unsupervised Traffic Flow Classification Using a Neural Autoencoder," IEEE 42nd Conference on Local Computer Networks (LCN), Singapore, pp. 523-526, 2017.

[6] S. Rezaei, X. Liu, "How to Achieve High Classification Accuracy with Just a Few Labels: A Semi-supervised Approach Using Sampled Packets," 2018.

[7] W. Wang, M. Zhu, J. Wang, et al., "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," IEEE International Conference on Intelligence and Security Informatics (ISI), Beijing, pp. 43-48, 2017.

[8] M.Lotfollahi, R.S.H. Zade, M.J. Siavoshani, et al., "Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning," Soft Computing, 2017.

[9] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, et al., "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things," IEEE Access, vol. 5, pp. 18042-18050, 2017

[10] W. Wang, Y. Sheng , J. Wang , et al., "HAST-IDS: Learning Hierarchical Spatial-Temporal Features Using Deep Neural Networks to Improve Intrusion Detection," IEEE Access, vol. 6, pp. 1792-1806, 2018.

[11] T. Mikolov, K. Chen, G. Corrado, et al., "Efficient Estimation of Word Representations in Vector Space," Proceedings of Workshop at ICLR, 2013.

[12] M.E. Peters, M. Neumann, M. Iyyer, et al., "Deep contextualized word representations," 2018.

[13] A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention Is All You Need," 2017.

[14] Y. Bengio, R. Ducharme, P. Vincent, "A Neural Probabilistic Language Model," Journal of Machine Learning Research, 2000.

[15] G. Draper-Gil, A.H. Lashkari, M.S.I. Mamun, et al., "Characterization of Encrypted and VPN Traffic using Time-related Features," The International Conference on Information Systems Security and Privacy (ICISSP), 2016.

[16] Z. Lan, M. Chen, S. Goodman, et al., "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations," 2019.