TrafficFormer: An Efficient Pre-trained Model for Traffic Data

***Abstract*—Traffic data contains deep domain-specific knowledge, making labeling difficult, and the lack of labeled traffic data affects the accuracy of traffic analysis. As a way of solving the problem of scarce labeled data, the pre-training technology has been widely adopted in the fields of vision and natural language. Still, exploration in the traffic analysis domain is insufficient. This paper proposes an efficient pre-training model TrafficFormer for traffic data. In the pre-training stage, TrafficFormer introduces a fine-grained multi-classification task to enhance the representation ability of traffic data; in the fine-tuning stage, TrafficFormer proposes a traffic data augmentation method utilizing the random initialization feature of fields, which helps the traffic model focus on key classification information. In addition to evaluating the traffic model's ability to recognize the network entity behaviors behind traffic, multiple new protocol understanding evaluation tasks are proposed to evaluate the model's understanding of network protocol interaction logic. TrafficFormer achieves the best performance on six traffic classification datasets with up to a 10% improvement in F1 score and has significantly superior protocol understanding capabilities compared to existing traffic pre-training models.**

1. Introduction

Traffic data, generated by the interaction of network entities, not only contains the interaction logic of the corresponding protocols but also incorporates the behavioral information of network entities. For example, traffic characteristics are different when users are using different applications or browsing different web pages. The analysis and classification of traffic are of great importance for network security and management. For example, through the identification of malware applications, hosts can alert users to avoid information leakage or property loss; by categorizing different application traffic, service providers can differentiate services and implement dedicated routing policies or queuing mechanisms to improve QoS (Quality of Service).

Traditional ML-based traffic analysis methods, such as FlowPrint [1], CUMUL [2], and Appscanner [3], rely heavily on expert knowledge to select specified features (e.g., packet interval, packet size, protocol, etc.). These features are fed into the machine learning model after feature engineering. With the development of deep learning (DL), researchers attempt to input raw traffic directly into DL models, such as FlowPic [4] based on Convolutional Neural Network (CNN) models, FS-Net [5] based on Autoencoder models, and GraphDapp [6] based on Graph Neural Network (GNN) models. These arts rely on DL models to learn complex feature patterns and classify traffic. A large amount of labeled data is the key to achieving superior results with the aforementioned methods. One of the most important reasons for the rapid development in the field of natural language processing (NLP) and computer vision (CV) is the presence of a large amount of labeled data, such as ImageNet [7] dataset in the CV field. When data is scarce, the effectiveness and generalizability of traffic analysis and identification are constrained.

However, compared with the labeling of traditional data (e.g., text, images, sounds, etc.), traffic data labeling is more difficult. Since text, images, and sounds are closely related to our daily lives, these types of data have little skill requirements for data labelers. On the contrary, traffic data requires labelers to know network protocols and have experiences in specific scenarios (*e.g.,* a certain network attack). Additionally, traffic data is extensive in scale and exhibits rapidly changing behavioral patterns, making manual labeling impractical. Given these challenges, large-scale and high-quality labeled datasets are scarce.

Pre-training technique [8], [9], [10] is a way to solve the problem of scarcity of labeled data, which consists of two stages: pre-training and fine-tuning. The pre-training stage uses unlabeled data to learn general knowledge in a self-supervised learning paradigm and the fine-tuning stage uses labeled data to learn task-specific knowledge in a supervised learning paradigm. Current large language models [11] rely on pre-training to mine information from a large amount of unlabeled data, allowing large models with hundreds of billions of parameters to be adequately trained to achieve superior results on many downstream tasks. Compared to text and image data, traffic data is more voluminous and has more complex behavioral patterns, thus applying the pre-training techniques to traffic analysis and identification tasks is reasonable.

Prior arts demonstrate the possibilities of pre-training techniques for traffic data. PERT [12] introduces the bigram approach to transform the hexadecimal representation of packet contents into words and uses the Masked Language Modeling (MLM) for packet-level pre-training. ET-BERT [13] considers a burst (a sequence of successive packets in the same direction) as a sentence and adopts MLM and next sentence prediction (NSP) tasks of BERT [9] to learn traffic patterns. YaTC [14] processes each flow as an image and adopts the Masked Image Modeling (MIM) task. However, these works have only explored the traffic representations to accommodate existing pre-training techniques, and have not

Prior Works	Traffic Representation	Pre-training Stage	Fine-tuning Stage
PERT [12]	Word	MLM	✗
ET-BERT [13]	Word	MLM & NSP	✗
YaTC [14]	Image	MIM	✗
TrafficFormer	Word	MLM & SODF	RIFA

TABLE 1: Designs of prior art in pre-trained traffic model.

tailored to traffic data during the pre-training and fine-tuning stages.

In this paper, we propose a pre-training model TrafficFormer[1] for traffic data to learn the basic traffic semantics from unlabeled traffic data to improve the accuracy of downstream traffic detection tasks. As shown in Table 1, in addition to traffic representation, we make innovative designs in the pre-training and fine-tuning stages of the traffic model by capturing the traffic data characteristics.

- First, traffic data is a form of sequential data, similar to natural language. However, the direction and order of its sequence units are more important. Therefore, TrafficFormer retains the masked modeling task during the pre-training stage to learn the sequential relationships of input units. And TrafficFormer proposes a fine-grained multi-classification task, *i.e.,* Same Origin-Direction-Flow (SODF), which mines the direction and order information of packets, enhancing the representation capability of traffic data (see §3.2).
- Second, traffic data is structured, and redundant information is ubiquitous in packet headers. In the fine-tuning stage, TrafficFormer proposes a traffic data augmentation method, *i.e.,* Random Initialization Field Augmentation (RIFA), which preserves traffic semantics. Data augmentation facilitates TrafficFormer in focusing on essential classification information quickly (see §3.3).

In addition to the pre-training and fine-tuning of the model, we also make innovations for the evaluation of the model. The design rationale is that traffic reflects the behavioral information of network entities and the interaction logic of network protocols. Different from the traditional traffic classification tasks that only evaluate the ability of a traffic model to recognize the behavior of entities, we evaluate the model's comprehension of the protocol interactions by introducing multiple novel protocol understanding tasks (e.g., packet direction recognition, packet loss detection, out-of-order detection, and packet prediction). This assures a comprehensive evaluation of the ability of traffic models.

Specifically, we evaluate the traffic classification performance of TrafficFormer on six public datasets. Experimental results demonstrate that TrafficFormer achieved the best performance across all datasets with up to a 10% improvement in F1 score. Meanwhile, in the protocol understanding task, TrafficFormer outperforms existing traffic pre-training models. We further conduct a study on the key components of TrafficFormer and reveal the impact of each on the model performance.

1. https://github.com/IDP-code/TrafficFormer

2. Background and Related Work

2.1. Pre-training Technique

The pre-training technique is a method that utilizes unlabeled data. It is divided into two stages: pre-training and fine-tuning. In the pre-training stage, general knowledge is learned in a self-supervised manner based on unlabeled data. In the fine-tuning stage, specific task knowledge is learned through supervised learning based on labeled data. The self-supervised learning constructs labels for unlabeled data, allowing the model to be trained in a supervised manner.

Common self-supervised methods include sequence modeling and contrastive learning. Sequence modeling predicts one part of a sequence based on another part of the sequence, while the training objective in contrastive learning is to bring closer the distance between similar samples and increase the distance between dissimilar samples. Sequence modeling typically also includes autoregressive modeling (*e.g.,* GPT [10] and MAE [15]), autoencoding modeling (*e.g.,* BERT [9] and BEiT [16]), and permutation coding modeling (*e.g.,* XLNet [17]). Typical pre-training works based on contrastive learning include MoCo [18], SimCLR [19] and SimCSE [20].

In terms of model structure, the current pre-training works are mostly based on the Transformer [21], which includes both encoder and decoder parts. Works that apply the encoder structure include BERT [9], XLNet [17], BEiT [16], SimCSE [20], etc., while works that apply the decoder structure include GPT [10], MAE [15], etc. Works that apply both encoder and decoder include BART [22], etc. Since CNN models are more suitable for image processing, early works in the visual field like MoCo [18] and SimCLR [19] were based on the ResNet [23] model. Subsequent works, like MoCo v3 [24], also began to be built based on the encoder structure.

In terms of model structure, TrafficFormer adopts the encoder structure of the Transformer. In terms of self-supervision, TrafficFormer employs self-supervised tasks designed for traffic data, which fully explore the composition of data packets in the traffic and the relationships between them.

2.2. Data Augmentation

Data augmentation increases the volume of data by creating different copies of the data, which helps enhance model performance and mitigate overfitting on training data. Different data augmentation methods have been designed for different types of data.

For image data, basic methods include cropping, rotating, color transformation, geometric transformation, and random erasing [25]. For text data, basic methods include replacing synonyms, random word insertion, word deletion,

and word substitution. Text can also be augmented through the back-translation technique, which translates the text into a target language and then translates it back. In addition to basic methods, some deep learning-based data augmentation methods have also been widely studied. BAGAN [26] and DAGAN [27] use Generative Adversarial Networks (GAN) for image data augmentation, while SeqGAN [28] and LeakGAN [29] use GANs for text data augmentation.

For traffic data, Vu [30] and Oeung [31] use Synthetic Minority Oversampling Technique (SMOTE) [32] to augment the traffic. Horowicz [33] transforms the traffic into images and uses image enhancement methods for traffic enhancement. Ring [34] explores data pre-processing strategies to generate traffic by GAN. ODDS [35] uses GAN to generate the behavioral distribution of bot hosts in feature space, which improves the bot detection ability with a limited amount of labeled data. Ta-GAN [36] uses GAN to generate minority class samples in the traffic, which alleviates the class imbalance problem in traffic classification. Compared to the above methods, our approach directly modifies the original data instead of modifying it in the feature space. Meanwhile, our approach is based on domain knowledge and the enhanced data obtains the original semantics.

2.3. Traffic Classification

Traditional traffic classification identifies traffic based on information such as port numbers, but as traffic behavior becomes more complex and some disguising techniques are used, the accuracy of this method has significantly decreased. Currently, traffic classification is generally based on machine learning and deep learning.

Machine learning-based methods [37], [38], [39], [40], [41] rely on expert-designed statistical features of traffic, which are then fed into machine learning models for training. Commonly used machine learning models include the Supported Vector Machine, Naive Bayes classifier, K-Nearest Neighbor, and Decision Tree models. The difference between traffic classification methods is mainly in the design of the features, which is closely related to the final performance. Machine learning models are usually smaller, which often results in faster training and inference speeds. Since the features in machine learning methods are handcrafted, they are better in terms of interpretability. Deep learning-based methods directly input the content of the raw data packets into the model, relying on the deep learning model to learn the complex patterns in the input and perform classification. Commonly used deep learning models include the CNN, Long Short-Term Memory (LSTM), GNN, and Encoder models. Some of the works [5], [42], [43], [44], [45], [46]input key attribute information of data packets, such as packet size and inter-packet interval, while other works [12], [13], [14], [47] directly input the raw byte of the data packets.

Some works pay attention to real-world problems, such as the open world problem with different distributions of test and training sets [1], [48], [49], efficient training and inference problems [50], [51], [52], [53], [54], and traffic mixing problem [55], [56]. Our work focuses on the real-world problem of achieving high-precision traffic classification when labeled traffic data is scarce.

3. TrafficFormer

In this section, we first introduce the design goal, challenges, and solutions of TrafficFormer, then present the overall framework of TrafficFormer, and specifically discuss its pre-training stage and fine-tuning stage.

Goal. The design goal of TrafficFormer is to fully exploit the information in unlabeled traffic data to learn the basic semantics of traffic, achieving better performance in downstream traffic tasks.

Challenges. Traffic is generated by the communication between two parties executing network protocols. Traffic data is a form of sequential data, similar to natural language. Several studies in natural language [57], [58] have shown that misordering of words has little effect on reading, whereas misordering of packets can lead to packet drops due to violation of interaction logic. Therefore, the order of input units (packets) in traffic is more critical compared with the order of words in language. In addition, the packet has the attribute of direction, and header information is highly redundant. The challenges include how to fully mine the sequence, direction, and order relationship of the input units, and to quickly locate valuable information for downstream traffic analysis tasks from the redundant information.

Solutions. To address the aforementioned challenges, TrafficFormer retains the masked modeling task from the NLP domain in the pre-training stage to learn the sequential relationships. And TrafficFormer designs the Same Origin-Direction-Flow (SODF) task to mine the direction and order information of the input units. In the fine-tuning stage, TrafficFormer proposes a traffic data augmentation method, *i.e.,* Random Initialization Field Augmentation (RIFA), to reduce the model's reliance on irrelevant information and quickly locate valuable information.

3.1. Framework

The overall framework of TrafficFormer is shown in Figure 1, which includes two stages: pre-training and fine-tuning. The tasks of the pre-training stage are Masked Burst Modeling (MBM) and Same Origin-Direction-Flow (SODF) multi-classification tasks. The trained model can be transferred to various downstream tasks, such as malware detection, website fingerprinting, and the newly proposed protocol interaction comprehension tasks. Considering the scarcity of training data for downstream tasks, TrafficFormer performs data augmentation in the fine-tuning stage.

TrafficFormer adopts a structure consistent with the BERT model. The input data is first transformed into vector representations through an encoding layer. The encoding layer includes three parts: word semantic encoding, word position encoding, and word segment encoding. The three parts are combined to form the final vector encoding of the word. The parameters of the encoding layer are continuously

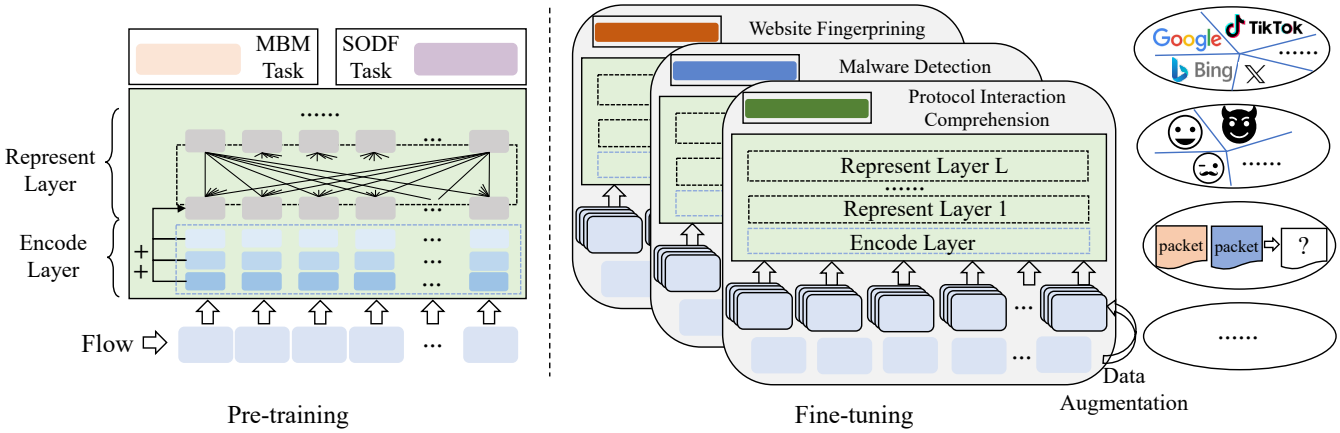


Figure 1: The framework of TrafficFormer.

updated with the training of the model, thus the word encoding can be continuously optimized. Then, the encoded vector representations enter representation layers, each of which is composed of a multi-head attention mechanism and a feedforward neural network. The attention mechanism establishes connections between each token, allowing each token to optimize its representation by combining the representations of other tokens. The feedforward neural network allows the model to learn nonlinear features, further enhancing the model's expressive capabilities. Finally, the token vectors are input into models corresponding to different tasks. The model computes gradients to optimize model parameters after receiving losses from different tasks. The number of parameters in TrafficFormer and BERT-base [9] are on the same order of magnitude.

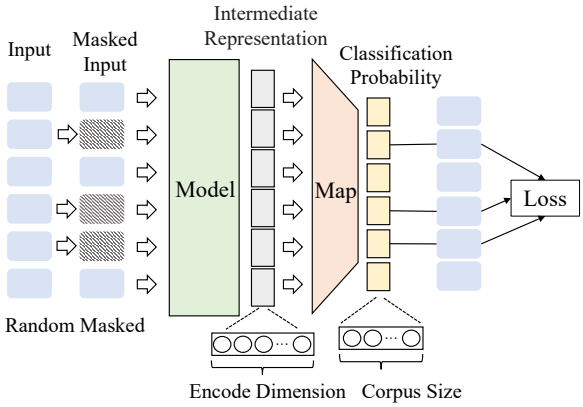


Figure 2: An illustration of the MBM task.

3.2. The Pre-training Stage

In the pre-training stage, the model is trained on a massive publicly unlabeled traffic dataset. The traffic data is first converted into tokens (tokenization), and then constructed into the input for the pre-training tasks. The pre-training tasks include MBM and SODF multi-classification tasks.

3.2.1. Data Preprocessing. A flow is defined by the 5-tuple, *i.e.,* the same source/destination IP address, source/destination port, and protocol, which includes packets in both directions. The definition of a flow is not limited to the 5-tuple. And a burst is defined as a sequence of consecutive packets transmitted along the same direction [59]. The traffic dataset is first split into multiple flows, and each flow is divided into multiple bursts.

Each packet is a string of hexadecimal numbers, *e.g.,* 4504008*bd*0. TrafficFormer converts packet in the form of bigrams, where each byte is connected with the following byte to obtain a 4-digit hexadecimal string. For the above example, after conversion, we get 4504 0400 008b 8bd0. Then, TrafficFormer uses the Byte Pair Encoding (BPE) algorithm to build a corpus with a maximum size of 65535. The BPE algorithm breaks down all the words in the training corpus into individual characters, and then continuously merges characters to form new words until the corpus reaches the preset size. Applying BPE results in a more fine-grained feature. Specifically, applying BPE algorithm in Bigram results in a subword with a minimum of 1 hexadecimal digit, which is less than most field lengths. In addition, special tokens such as [CLS], [SEP], [PAD], [MASK], and [UNK] are included in the corpus. [CLS] is used for classification tasks. [SEP] is used to separate different sequences. [PAD] is used to pad the input to the maximum length. [MASK] serves the masked modeling task, used to replace the masked words. [UNK] is used to represent words not in the corpus.

3.2.2. Pre-training Tasks. We first present the two pre-training tasks (MBM and SODF) separately and then the overall pre-training task loss.

MBM Task. Traffic data is sequential data as the natural language. For the capture of sequence information, the masked modeling task is a classical approach, such as Masked Language Modeling (MLM) and Masked Image Modeling (MIM). TrafficFormer retains the masked modeling task.

MBM task masks some tokens in the input and needs the model to predict them, and the input of MBM is bursts which are continuous packets in the same direction.

As shown in Figure 2, the original input is first randomly masked to obtain the masked input. Then, the model obtains relevant information from the context to learn the intermediate representations of tokens with the dimension being the set encoding dimension. The intermediate representation goes through a mapping network to obtain the prediction probabilities, with the dimension being the size of the corpus. Finally, the classification probabilities at the masked positions are combined with the true probabilities to calculate the loss. The loss function is cross-entropy loss, as shown in Equation 1. n is the number of masked tokens. t_i is the true probability for the i-th token, which uses one-hot encoding. $\hat{t_i}$ is the predicted probability for the i-th token, which is the classification probability in Figure 2, and the sum of the values in $\hat{t_i}$ is 1.

$$loss_{MBM} = -\sum_{i=1}^{n} t_i log(\hat{t_i}) \tag{1}$$

SODF Task. The direction and order of packets in traffic data are more important compared to words in text. However, previous works have not focused on the characteristics of traffic. ET-BERT also has two pre-training tasks and the second pre-training task is the Same-origin Burst Prediction (SBP) task similar to the NSP task. SBP splits a burst into two segments, and then with a certain probability, replaces the latter segment with a segment from other bursts. The SBP task predicts whether the given two segments originate from the same burst, *i.e.,* a binary classification task. There are two issues with this task:

1) The task is relatively simple. Two segments from the same burst have many consistent and similar fields, such as IP, IPID, and sequence number. Although some fields in the two segments are not the same, they are very close. For example, the first 16 bits of the sequence number are often the same, while the field in a randomly replaced segment differs significantly. Therefore, predicting whether two segments originate from the same burst is relatively simple.
2) The information learned is limited. The burst only contains packets in the same direction, while the input to the model during fine-tuning stage will contain consecutive packets, which generally have different directions. This causes an inconsistency issue between pre-training stage and fine-tuning stage. Since there are only packets in one order, the SBP's cognition of the packet order information is also limited.

Therefore, the SBP task is hard to capture the directional and sequential information in the traffic. We design the Same Origin-Direction-Flow (SODF) multi-classification task to achieve this goal. The SODF task combines the split burst segments to form five categories. Figure 3 shows an example of the combination of different categories.

1) Category 1: A normal burst, with the two split segments of the burst separated by a [SEP] token. The segment identifiers of tokens in the two split segments are 1 and 2, respectively.
2) Category 2: Similar to Category 1, the difference is that the two segments of a burst are swapped after being split.
3) Category 3: Two consecutive bursts from one flow, with a [SEP] token between the two bursts. The segment identifiers of tokens in the two bursts are 1 and 2, respectively.
4) Category 4: Similar to Category 3, the difference is that the two bursts are swapped.
5) Category 5: Arbitrarily combine bursts from two flows, with a [SEP] token added between bursts for separation.

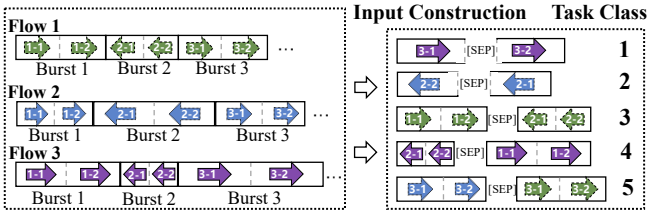


Figure 3: An example of the SODF task. There are three flows and we show the first three bursts of each flow. These bursts vary in length and together form five categories.

Categories 1 and 2 represent the normal and disordered packets within a burst. Categories 3 and 4 further contain two bursts in two directions. All four above categories are associated with a single flow, while category 5 includes bursts of different flows. Thus, the SODF task allows the model to learn the packet order, the packet direction, and the packet slave flow (flow that the packet belongs to) together. Each burst is processed into each category with a probability of 20%, meaning that the sample size is the same for each category. The SODF task also uses cross-entropy loss, as shown in Equation 2. Here, b is the batch size. d_i is the true probability distribution of sample i, and $\hat{d_i}$ is the predicted probability distribution of sample i, both of which have a dimension of 5, corresponding to the number of categories in the SODF task.

$$loss_{SODF} = -\sum_{i=1}^{b} d_i log(\hat{d_i}) \tag{2}$$

The pre-training stage adopts the multi-task learning mode, and the total loss function of pre-training is shown as Equation 3. Here, λ is a hyperparameter used to balance the loss values of the two tasks and is set to $\lambda = 0.1$ in the experiments. Due to the adoption of multi-task learning, the MBM task is also more difficult because of the influence of various inputs (bursts of different flows, directions, and orders) in the SODF task compared with ET-BERT, which enhances TrafficFormer's learning of the basic semantics of the traffic.

$$loss = \lambda * loss_{MBM} + loss_{SODF} \tag{3}$$

3.3. The Fine-tuning Stage

In the fine-tuning stage, the model is initialized with the parameters of the pre-trained model and then continues to train on the data of the specific task. To maintain consistency with pre-training, the fine-tuning data is processed into the same input form as the pre-training data. First, the data packets are converted into hexadecimal strings, then transformed into bigram form, and finally tokenized using the corpus generated in the pre-training stage. The tokens of packets are directly concatenated without adding a [SEP] token for separation, *i.e.,* the segment identifiers of all tokens are the same.

Traffic Data Augmentation. Considering the small amount of data in the downstream fine-tuning task, TrafficFormer proposes a traffic data augmentation method.

Protocol	Randomly Initialized Fields
IP	IPID
TCP	source port, sequence number, acknowledge number, timestamp in the timestamp option
UDP	source port
TLS	the random number in the client hello and server hello message

TABLE 2: Randomly initialized fields in common protocols

Some fields in the network protocol are initialized randomly, and the field values have no meaning, so they have no effect on classification. The Table 2 lists the random initialization fields in common protocols. In the IP protocol, the IPID field is used to label each packet. When an IP packet is fragmented, the IPID fields of the multiple fragments of the same packet are consistent. Random initialization of the IPID increases the difficulty of guessing to prevent malicious attacks. In the TCP and UDP protocols, the source port number identifies the client's sending program. To prevent port scanning or denial-of-service attacks, the source port number is also randomly initialized. Since the TCP protocol is based on byte streams, the sequence number in the TCP protocol indicates the starting position of the bytes to be sent, and the acknowledgment number indicates the position of the next expected byte to be received. Guessing the TCP source port number, sequence number, and acknowledgment number is the basis for TCP hijacking attacks [60], [61], and random initialization helps to increase the difficulty of hijacking attacks. The timestamp in the TCP timestamp option is used to calculate the round-trip time delay and to prevent sequence number wraparound, and it may be a value increased by a random offset [62]. In the TLS protocol, the random numbers in the client hello and server hello messages are used to generate encryption keys, and randomization enhances the strength of the session keys and helps to prevent replay attacks.

Based on the above insights, TrafficFormer proposes the traffic data augmentation method, *i.e.,* Random Initialization Field Augmentation (RIFA). In a nutshell, RIFA creates multiple copies of traffic data by randomly varying randomly initialized fields in a packet while retaining the original label since RIFA doesn't change the original semantics.

Datasets	Size	Flows	Included Protocols [§]
ISCX-NonVPN [63]	4.9GB	219076	TLS1.2, SFTP, SSDP, SNMP, NTP, MDNS, HTTP, GQUIC...
CICMalAnal-2017 [64]	6.5GB	232627	TLS1.2, GQUIC, SSDP, MDNS...
Browser [1]	7.4GB	149527	TLS1.3, GQUIC...

[§] Only partial special protocols are listed, and common protocols such as IP, TCP, UDP, and DNS are not included.

TABLE 3: Pre-training datasets.

Considering that the change pattern of fields is often more important than the field values, RIFA maintains the change pattern of a field in subsequent packets after changing the initial value. For example, in the TCP protocol, RIFA replaces the sequence number of the first packet with a random number and then assigns the sequence numbers of subsequent packets as the random number plus the original difference. After data augmentation, the model can focus less on the values of these fields and more on the changes in values or other fields, thereby quickly extracting valuable information from a large amount of data. Compared to deep learning-based methods, RIFA is based on domain knowledge and modifies the original data directly, not in the feature space.

In traffic data, the source/destination IP and source/destination ports may introduce shortcuts in classification. ET-BERT has chosen to select the packet content after the port number to avoid this issue. Although this method can solve the problem, it loses a lot of information. For example, in the TCP protocol, there is only the size of the TCP header, and without inputting the entire TCP payload, information about the TCP payload size may be missing. On the contrary, TrafficFormer still utilizes all the packet header content, while randomly varying the IP and port number fields. This not only retains more information but also prevents shortcuts.

4. Evaluation

In this section, we evaluate TrafficFormer extensively on multiple datasets to demonstrate:

1) TrafficFormer outperforms previous methods in traffic classification tasks, including machine learning methods, deep learning methods, and pre-training methods.
2) TrafficFormer outperforms previous pre-training methods in protocol understanding tasks.
3) The impact of key components on the classification result.

4.1. Experiment Setup

We build TrafficFormer based on Pytorch 2.0.1, and all experiments are run on NVIDIA A100 GPUs. The following introduces specific experimental settings, including pre-training datasets and settings, metrics, and baselines.

Pre-training Datasets. As shown in Table 3, we select three datasets from different sources for pre-training, including ISCX-NonVPN [63] (2016), CICMalAnal2017 [64] (2017), and Browser [1] (2020). The ISCX-NonVPN dataset contains traffic from various application types, such as browsers, email, audio, video, file transfer, etc., and we choose the non-VPN part of the traffic as training data. The CICMalAnal2017 contains traffic from normal software and malware, and we choose the normal software traffic in 2017 as training data. The Browser dataset contains traffic data obtained by using a Samsung phone to access the top 1000 websites provided by Alexa with Google Chrome, Firefox, UC, and Samsung browsers.

We first use SplitCap to split the dataset by flow. After splitting, the dataset no longer includes protocols such as ARP, ICMP, etc., because these protocols are connectionless and irrelevant to actual data transmission. Table 3 shows detailed information about the three datasets. The total volume of the pre-training dataset is about 20GB, including more than 600,000 flows. Table 3 also shows the special protocols contained in the datasets. ISCX-NonVPN and CICMalAnal2017 datasets contain software data, which includes a variety of network protocols, with ISCX-NonVPN having a more diverse range of protocols. However, these two datasets were collected earlier and lack data for the TLS 1.3 protocol, while the Browser dataset compensates for this. Therefore, the pre-training dataset is sufficient in volume and rich in protocols.

Pre-training Settings. Ethernet addresses often relate to the location of traffic collection and are unrelated to the communicating parties, thus the content in the Ethernet packet header is useless for classification. Additionally, most packets are encrypted nowadays, and the content of the encrypted payload is also useless. Therefore, during pre-training, each packet is taken 64 bytes of data after the Ethernet layer. The model's encoding dimension is 768, with a total of 12 layers, each multi-head attention having 12 heads, and the maximum sequence length is 512. The SODF task is based on the representation of [CLS] token in the last layer of the model for classification. We set the batch size to 64, and the number of GPUs to 3, resulting in an actual batch size of 192. The optimizer is Adam, with a learning rate of 2e-5, a linear decay scheduling strategy, and a warm-up ratio of 0.1. The total number of training steps is set to 500,000. The loss tends to stabilize at 120,000 steps, so we selected the model at 120,000 steps as the initial model for fine-tuning downstream tasks.

Evaluation Metrics. We use classification accuracy (AC), precision (PR), recall (RC), and F1 score as the evaluation metrics. For multi-class tasks, when calculating the above metrics for each class, that class is considered the positive class, and all other classes are considered the negative class. When calculating the above metrics for the entire dataset, we average the metric value of each class as the metric value for the overall classification. This balances the problem of different sample sizes across classes.

Baselines. We select six baselines, including two machine learning methods (Appscanner and BIND), two deep learning methods (DeepFP and GraphDapp), and two pre-training methods (ET-BERT and YaTC). ML-based/DL-based baselines use TrafficFormer's fine-tuning datasets for training. Pre-training model baselines use the same pre-training datasets and fine-tuning datasets with TrafficFormer.

1) In Appscanner [41], statistical features are constructed using packet size and direction, totaling 54 features. The model is random forest, and the validation set is used to select the appropriate classification threshold.
2) In BIND [40], distribution features are constructed using the size and interval of packets and bursts, totaling 700 features. The model is random forest, and the validation set is used to select the appropriate classification threshold.
3) In DeepFP [43], the size and direction information of packets are input into a CNN model. The validation set is used to select the number of channels, the dimension of the fully connected layer, and the learning rate.
4) In GraphDapp [46], the relationships of packets are constructed in the form of a graph, and the constructed graph is trained through a GNN model. The validation set is used to select the number of layers in the GNN model, the size of the hidden layer, and the input length.
5) ET-BERT [13] and TrafficFormer use the same pre-training data and hyperparameters, but the pre-training tasks are different. We select the model at step 120,000 for fine-tuning, and the input data during fine-tuning is also consistent with TrafficFormer.
6) YaTC [14] processes each flow as an image, which also uses the same pre-training data and hyperparameters as TrafficFormer. We choose the model at step 400,000 for fine-tuning downstream tasks.

Datasets		Tasks	Flow Number[¶]	Class Number[¶]
Cross-Platform (Android)	[65]	Application Fingerprinting	32149	197
Cross-Platform (iOS)	[65]	Application Fingerprinting	19736	190
CSTNET-TLS 1.3	[13]	Website Fingerprinting	46372	120
ISCX-VPN (Service)	[63]	Service Type Identification	1457	6
ISCX-VPN (App)	[63]	Application Fingerprinting	1444	11
USTC-TFC	[66]	Malware Detection	6049	14 [§]

[¶] The number of flows and classes refers to the actual number of flows and classes used for classification after processing.
[§] Normal software classes: Malware classes = 5:9

TABLE 4: Fine-tuning datasets.

4.2. Traffic Classification Task

In this section, we demonstrate the performance of TrafficFormer on six fine-tuning tasks for traffic classification.

Fine-tuning Datasets. As shown in Table 4, six datasets are selected as the fine-tuning datasets in this section, includ-

Datasets	Cross-Platform(Android)				Cross-Platform(iOS)				CSTNET-TLS 1.3			
Approaches	AC	PR	RC	F1	AC	PR	RC	F1	AC	PR	RC	F1
Appscanner [41]	0.5185	0.4897	0.3538	0.3982	0.4058	0.3267	0.2943	0.3014	0.7182	0.8205	0.6771	0.7305
BIND [40]	0.4025	0.2987	0.2705	0.2774	0.3607	0.2456	0.2407	0.2330	0.8014	0.7842	0.7445	0.7510
DeepFP [43]	0.2669	0.1542	0.1588	0.1525	0.2138	0.1235	0.1201	0.1169	0.6345	0.5868	0.5870	0.5835
GraphDapp [46]	0.4806	0.3676	0.3429	0.3396	0.2605	0.2430	0.2533	0.2460	0.7945	0.7690	0.7669	0.7622
ET-BERT [13]	0.6952	0.5446	0.5163	0.5162	0.4721	0.4059	0.3634	0.3680	0.8120	0.7986	0.7888	0.7884
YaTC [14]	0.6233	0.4324	0.4077	0.4088	0.3931	0.2975	0.2902	0.2815	0.8407	0.8165	0.8170	0.8133
TrafficFormer	0.7369	0.5900	0.5726	0.5644	0.4807	0.4003	0.372	0.3699	0.8197	0.8064	0.8027	0.8014
TrafficFormer w/ EA	**0.7664**	**0.6435**	**0.6204**	**0.6167**	**0.5679**	**0.4966**	**0.4697**	**0.4689**	**0.8484**	**0.8371**	**0.8351**	**0.8338**

TABLE 5: The results of different approaches on the Cross-Platform dataset and the CSTNET-TLS 1.3 dataset.

ing Cross-Platform (Android), Cross-Platform (iOS), ISCX-VPN (Service), ISCX -VPN(App), CSTNET-TLS 1.3, and USTC-TFC, which include four specific fine-tuning tasks: application fingerprinting, service type identification, website fingerprinting, and malware detection. Cross-Platform (Android) and Cross-Platform (iOS) collect traffic data of the 100 most popular applications on Android and iOS phones in China, the United States, and India, respectively. CSTNET-TLS 1.3 contains traffic data collected on 120 websites using the TLS 1.3 protocol. The ISCX-VPN(Service) and ISCX-VPN(App) contain the traffic of different behaviors of multiple applications in the Virtual Private Network (VPN), and the traffic can be classified into different applications and services. For example, there are three service types of traffic in the Skype application, *i.e.,* text chat, file transfer, and voice chat. The three types of traffic belong to different classes in ISCX-VPN(Service), while they all belong to the Skype class in ISCX-VPN(App). The USTC-TFC dataset contains traffic generated by 10 normal software and 10 malware.

Same as the pre-training processing, we first split the fine-tuning dataset into flows, then remove the flows that are less than 2KB or less than 3 packets, and then remove the classes that have less than 10 flows. For the classes with more than 500 flows, we randomly select 500 flows from them, so that the number of flows of each class is not more than 500. The number of flows and classes remaining in each dataset after processing are shown in Table 4. In the malware detection task, the ratio of the number of normal software classes to the number of malware classes is 5:9. We divide the training set, validation set, and test set in the ratio of 8:1:1 for each class.

Fine-tuning Settings. For machine learning and deep learning methods, all packets of a flow are taken as inputs. For pre-training methods, the first 5 packets of each flow are selected. For ET-BERT and TrafficFormer, each packet takes 64 bytes of data after the Ethernet layer as input to the model and a total of 20 rounds of training are performed. For YaTC, each packet takes 80 bytes for the header and 240 bytes for the payload (data after IP/TCP) and a total of 300 rounds of training are performed. For pre-training methods, we select the optimal model among multiple learning rates and multiple rounds based on the validation set.

In addition, for pre-training methods, we process IP, port, timestamps in the TCP protocol, and timestamps and server name indication (SNI) in the TLS protocol via the data enhancement method in Sec. 3.3 to avoid classification shortcuts. For data enhancement in TrafficFormer, we randomly initial the IPID, TCP sequence number, and acknowledge the number five times to produce five times more data. To keep the total amount of data consistent, only 4 rounds of training are applied to the data-enhanced TrafficFormer (TrafficFormer w/ EA).

Cross-Platform Dataset. The results of different approaches on the Cross-Platform dataset are shown in Table 5. The pre-training approaches largely achieve better results compared to both deep learning and machine learning approaches, the reason is that the pre-training approaches input more packet information and extract valuable information from the complex information. For pre-training approaches, YaTC has the worst performance. TrafficFormer outperforms ET-BERT in all metrics, with an improvement of 4.82% and 0.19% in the F1 value, respectively. After using the data enhancement, TrafficFormer w/ EA improves 10.05% and 10.09% on the F1 value compared to ET-BERT. This proves the effectiveness of the data enhancement, which reduces the model's dependence on irrelevant information. The classification accuracies of the Cross-Platform dataset are not very high, which may be related to the following factors: (i) there are too many categories in the dataset, with 197 and 190 categories respectively; (ii) there are multiple applications of the same type in the dataset, and these applications have similar network interaction patterns; (iii) different applications in the dataset often access the same third-party domain names. For example, 78% of Android applications in the United States will access google.com, meaning these applications have the same access records [65]. In addition, the iOS system is more closed compared to the Android system, and the traffic patterns on iOS are more uniform, making it more difficult to identify application fingerprints on the iOS system. However, TrafficFormer still shows significant improvement over the best methods, proving its efficient traffic detection capabilities.

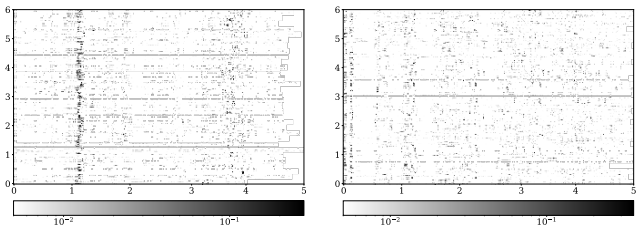
CSTNET-TLS 1.3 Dataset. The rightmost part of Table 5 shows the results of website fingerprinting on the CSTNET-TLS 1.3 dataset. TrafficFormer achieves the optimum in all metrics, with a 3.92% improvement in F1 value compared to the optimum results in the machine learning and deep learning approaches, and a 1.3% improvement in F1

Datasets	ISCX-VPN(Service)				ISCX-VPN(App)			
Approaches	AC	PR	RC	F1	AC	PR	RC	F1
Appscanner [41]	0.9178	0.9036	0.9212	0.9113	0.7724 (0.6207)[¶]	0.7146 (0.5968)	0.6906 (0.5712)	0.6918 (0.5640)
BIND [40]	0.8562	0.8741	0.8433	0.8543	0.7724 (0.6000)	0.6664 (0.5154)	0.6624 (0.5367)	0.6547 (0.5147)
DeepFP [43]	0.6781	0.6845	0.6678	0.6723	0.7310 (0.5034)	0.6413 (0.3224)	0.6901 (0.4119)	0.6599 (0.3559)
GraphDapp [46]	0.8906	0.8903	0.9038	0.8956	0.7969 (0.5938)	0.7351 (0.5637)	**0.7813** (0.5154)	**0.7419** (0.4826)
ET-BERT [13]	0.9452	0.9496	**0.9450**	0.9454	0.7586	0.6255	0.6236	0.6042
YaTC [14]	0.8356	0.8163	0.8116	0.8122	0.7310	0.6004	0.6199	0.6034
TrafficFormer	0.9247	0.9268	0.9167	0.9205	**0.8000**	**0.7675**	0.7036	0.6959
TrafficFormer w/ EA	**0.9589**	**0.9621**	**0.9450**	**0.9580**	0.7931	0.7544	0.7044	0.7129

[¶] The contents in parentheses are the results of the method using only the first five packets of information.

TABLE 6: The results of different approaches on the ISCX-VPN dataset.

value compared to the optimum pre-training method. After data enhancement, TrafficFormer w/ EA improves 2.87%, 3.07%, 3.24%, and 3.24% compared to TrafficFormer in terms of accuracy, precision, recall, and F1 value respectively. This indicates that data enhancement is very helpful for the website fingerprinting task, making the model focus on the valuable information in the packet.



(a) Cross-Platform(Android) dataset (b) CSTNET-TLS 1.3 dataset

Figure 4: The illustrations of attentions of the last layer of [CLS] tokens on the Cross-Platform(Android) dataset and CSTNET-TLS 1.3 dataset in TrafficFormer. The horizontal axis is 5 packets, and within each interval are the 64 bytes of the packet. The vertical axis is the first 6 of the 12 attention headers, and within each interval is a batch of data with a batch size of 32.

We show the illustrations of attention in TrafficFormer on the CSTNET-TLS 1.3 dataset and Cross-Platform (Android) dataset in Figure 4. As we can see from Figure 4a, the attention is more concentrated in two parts in the Cross-Platform (Android) dataset. In contrast, the attention of CSTNET-TLS 1.3 dataset is more dispersed as shown in Figure 4b, with multiple bytes in the packet having some impact on the final classification. The two illustrations of attention have a lot in common. For the same data, there is some disparity between each attention head, suggesting that they are focusing on different parts of the data. For a given attention head, its focus varies across different samples in a batch. This suggests that the pre-trained model learns diverse knowledge from complex data, which allows for the recognition of different classes.

ISCX-VPN Dataset. The results of different approaches on the ISCX-VPN dataset are shown in Table 6, including the results for service type identification and application fingerprinting. In the task of service type identification, the pre-training approaches demonstrate better results compared to machine learning approaches and deep learning approaches. TrafficFormer w/ EA performs best in the service type identification task, with an improvement of 1.26% in F1 value compared to the optimal baseline. Compared with TrafficFormer, TrafficFormer w/ EA improves 3.42%, 3.53%, 2.83%, and 3.75% in terms of accuracy, precision, recall, and F1 value respectively. This suggests that the traffic data augmentation is helpful for the service type identification task.

In the task of application fingerprinting, TrafficFormer is optimal in terms of accuracy and precision, and GraphDapp is optimal in terms of recall and F1 value. The pre-training approaches do not show better results compared to machine learning and deep learning approaches. This may be related to the fact that the pre-training approaches only use information from the first 5 packets for fast traffic processing, while machine learning and deep learning approaches use information from all packets. We report the results using only the information from the first 5 packets in the parentheses. It can be seen that when only the information from the first 5 packets is used, the machine learning and deep learning approaches are not as effective as the pre-training approaches. The optimal F1 value in the machine learning and deep learning approaches is 56.4%, which is lower than the worst F1 value of 60.34% in the pre-training approaches. TrafficFormer and TrafficFormer w/ EA improve 9.17% and 10.87% in F1 value respectively compared to the best baseline. In other words, TrafficFormer can make more accurate decisions when only a few packets have appeared in a flow, which is very important in scenarios where quick decisions are needed. TrafficFormer w/ EA does not show better results compared to TrafficFormer in accuracy and precision, which is limited by the enhancement mode in this section. We augment the data eight times for 10 rounds of training and the four metrics are 83.45%, 73.31%, 74.96%, and 73.94% respectively. The results significantly outperform that of TrafficFormer.

USTC-TFC Dataset. Table 7 shows the results of the malware detection task on the USTC-TFC dataset. In this task, the pre-training approaches are also superior to those of machine learning and deep learning. For pre-training approaches, YaTC has the worst performance. TrafficFormer

Approaches	AC	PR	RC	F1
Appscanner [41]	0.8942	0.9137	0.9050	0.8984
BIND [40]	0.8926	0.8995	0.9043	0.9013
DeepFP [43]	0.8496	0.8613	0.8543	0.8548
GraphDapp [46]	0.8633	0.8787	0.8779	0.8738
ET-BERT [13]	0.9699	0.9741	0.9724	0.9727
YaTC [14]	0.9672	0.9679	0.9659	0.9667
TrafficFormer	0.9766	0.9795	0.9777	0.9784
TrafficFormer w/ EA	**0.9816**	**0.9837**	**0.9826**	**0.9830**

TABLE 7: The results of different approaches on the USTC-TFC dataset.

improves by 0.67%, 0.54%, 0.53%, and 0.57% in accuracy, precision, recall, and F1 score, respectively, compared to ET-BERT. After data augmentation, TrafficFormer w/ EA improves by 1.17%, 0.96%, 1.02%, and 1.03% in accuracy, precision, recall, and F1 score, respectively, compared to ET-BERT. So TrafficFormer has better accuracy in detecting malicious traffic and the traffic data augmentation is also helpful for the malware detection task.

4.3. Protocol Understanding Task

Traffic not only reflects the behavioral information of network entities but more fundamentally, contains the interaction information of network protocols. In this section, we propose new tasks to evaluate the protocol interaction logic understanding ability of the pre-trained traffic model, which helps to more comprehensively evaluate the capabilities of a pre-trained model. Specifically, we propose four tasks, including packet direction judgement, packet loss detection, packet out-of-order detection, and packet prediction.

Packet Direction Judgement. Randomly take two packets from a flow. If the direction of the two packets is the same, the label is 1. Conversely, the label is 0. This binary classification task evaluates the model's cognitive ability regarding the direction of packets.

Packet Loss Detection. Take N consecutive packets from a flow, randomly drop any packet from 2 to N-1 to form a packet loss sample, labeled as 0. To keep the number of packets the same, take packets from 2 to N or 1 to N-1 as a no packet loss sample, labeled as 1. This binary classification task evaluates the model's cognitive ability regarding the order of packets.

Packet Out-of-order Detection. Take N consecutive packets from a flow, randomly select any packet from 1 to N-1, and insert it into any position after it to form an out-of-order sample, labeled as 0. The original N packets are a non-out-of-order sample, labeled as 1. This binary classification task evaluates the model's cognitive ability regarding the order of packets.

Packet Prediction. Packet prediction is to predict the fields in the header of the packet. We meticulously categorize the packet header fields. Taking the TCP protocol as an example, some fields can be predicted, such as the TCP sequence number. Some fields are difficult to predict, such as the window size, because it is related to the available resources in the host and is hard to predict based solely on the protocol interaction. The prediction of window size is more reliant on experience. Some fields are more challenging to predict, such as the checksum field, which is calculated based on all the contents of the packet, requiring the model to learn the complex logic of checksum calculation. The packet prediction task focuses on predictable fields because they reflect the logic of protocol interaction. The task takes N consecutive packets from a flow, masks tokens related to the field to be predicted in the last packet, and expects the model to give the correct token for that position. The task is similar to the masked modeling task, but the masked modeling task predicts masked token based on the context, while this task is based on the previous packets, *i.e.,* there is only preceding context, making the prediction more challenging. The task is also a multi-classification task, with the number of categories being the size of the corpus. This task evaluates the model's comprehensive cognitive ability regarding the logic of protocol interaction.

We construct protocol understanding tasks based on two datasets: CSTNET-TLS 1.3 and CICMalAnal2017. To avoid overlap with pre-training data, we select benign data in 2016 from the CICMalAnal2017 dataset. We construct the task data according to the above description, and N=5. For the packet prediction task, we focus the IP and TCP protocols and the fields to be predicted include IPID, source/destination IP, source/destination port, TCP sequence number, TCP acknowledgment number, TCP header length, and TCP flags. We predict the fifth packet based on the first four packets of each flow, meaning that some fields of the fifth packet input into the pre-trained model are masked and waiting to be predicted. To provide the direction of the next packet, the data will randomly include one of the source IP, destination IP, source port, or destination port from the packet to be predicted. In the first three tasks, the number of positive and negative samples is equal, and we show the F1 scores. For the packet prediction task, we show the predictive accuracy of fields.

The baselines in this section are ET-BERT and YaTC. In the first three tasks, each method was trained for a total of 5 rounds. In the fourth task, ET-BERT and TrafficFormer were trained for 50 rounds and YaTC for 150 rounds. In YaTC, the masked modeling task is fitting the target value, not to do multi-classification. So in the packet prediction task, the last dimension is made 256 through the reshape operation and fully connected layers in order to do multi-classification. Table 8 shows the performance of pre-trained models on different datasets and protocol understanding tasks. Here, CSTNET represents the CSTNET-TLS 1.3 dataset, and CIC represents the CICMalAnal2017 dataset.

As we can see from Table 8, both TrafficFormer and ET-BERT achieve close to 100% F1 scores in the task of packet direction judgment. However, YaTC has poor F1 score on the CSTNET dataset, with F1 score about 6% worse compared to the other two methods. In the packet loss detection task, YaTC still has the worst F1 score. TrafficFormer and ET-BERT are similar on the CIC dataset. On the CSTNET

Tasks	Packet direction judgement		Packet loss detection		Packet out-of-order detection		Packet prediction	
Approaches	CSTNET	CIC	CSTNET	CIC	CSTNET	CIC	CSTNET	CIC
ET-BERT [13]	0.9996	1.0000	0.8862	0.9890	0.8622	0.9874	0.7847	0.7446
YaTC [14]	0.9374	0.9931	0.7743	0.9789	0.7141	0.9767	0.7336	**0.7687**
TrafficFormer	**0.9998**	**1.0000**	**0.8923**	**0.9901**	**0.8837**	**0.9892**	**0.8361**	0.7522

TABLE 8: Performance of pre-trained traffic models on different datasets and different protocol understanding tasks.

Datasets	packet 1	packet 2	packet 3	packet 4
CSTNET	164.703	190.869	190.869	192.582
CIC	80.021	126.728	126.728	121.86

TABLE 9: The average edit distance of flows (first four packets) in different datasets.

dataset, TrafficFormer outperforms ET-BERT with a F1 difference of 0.6%. In the packet out-of-order detection task, the results are similar to the packet loss detection task, and TrafficFormer still has the best performance. On the CSTNET dataset, TrafficFormer outperforms ET-BERT with a F1 difference of 2.15%. In the packet prediction task, TrafficFormer performs best on the CSTNET dataset, with an accuracy 5.14% higher than the optimal baseline. YaTC performs best on the CIC dataset, with an accuracy 1.65% higher than TrafficFormer. The performance loss might be caused by TrafficFormer having more categories than YaTC (65536 vs. 256) since they have different packet representations. When YaTC is also trained for the same 50 rounds as TrafficFormer, its accuracy is only 0.6443, 10.79% lower than TrafficFormer. TrafficFormer achieves good performance with less training, indicating that it has already learned the relevant order information of packets in the pre-training stage. In summary, TrafficFormer is stronger than prior arts in protocol understanding capability.

It can be seen that the results on the CIC dataset are higher than those on the CSTNET dataset in the first three tasks. We compare the similarity of different flows in the two datasets, specifically, computing the edit distance of packets in two random flows. Table 9 shows the average edit distance of the first four packets. It can be seen that the edit distance of CIC is smaller than that of CSTNET, indicating that the different flows in the CIC dataset are more similar, and therefore the prediction difficulty is less difficult, resulting in higher results on the CIC dataset. We also compute the edit distance of neighboring packets in a flow. The average distance is 127.6 on the CIC dataset and 123.2 on the CSTNET dataset. This suggests that packets in a flow in the CSTNET dataset are more similar, which may be the reason of higher packet prediction accuracy on the CSTNET dataset.

4.4. TrafficFormer Deep Dive

In this section, we explore the key components of TrafficFormer to understand the impact of each part on the final results. The exploration includes six parts: the impact of pre-training, the impact of the number of pre-training steps, the impact of data augmentation, the impact of input content, the impact of traffic representation, and the impact of sequence representation. We choose representative datasets for evaluation, *i.e.,* Cross-Platform (Android) (the maximum number of categories), CSTNET-TLS (the maximum number of flows), and ISCX-VPN(App) (the small number of flows and categories).

Impact of Pre-training. We evaluate the model without pre-training on traffic classification tasks, *i.e.,* fine-tuning starting directly from a randomly initialized model. The model without pre-training maintains the same hyperparameters as TrafficFormer and selects the best results under multiple learning rate settings.

The performance of the model without pre-training on traffic classification tasks is reported in Table 10. The largest drops in F1 score are ISCX-VPN (Service) and ISCX-VPN (App) datasets, which decrease by 85.17% and 66.05%, respectively. In the four learning rates tried, the model shows equally poor performance, indicating that the model does not learn useful information. The reason may be that both datasets have a relatively small number of samples (about 1500), and the small amount of training data leads to the model not learning the associations between inputs, resulting in poor performance. Since labeling traffic data is difficult, a small sample size is common. In the scenario of data scarcity for downstream tasks, pre-training is very necessary, as pre-training learns the basic traffic semantic information, which can be quickly transferred to downstream tasks. The smallest drop in the F1 score is the USTC-TFC dataset, which decreases by 6.75%. Among the six datasets, CSTNET-TLS 1.3 has the most data, but its F1 score decreases by 14.28%, which is not the least. This indicates that the decrease in performance is not only related to the small amount of data in the fine-tuning task but also that TrafficFormer learns related knowledge in the pre-training stage.

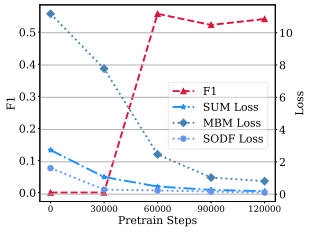
Impact of Pre-training Steps. We evaluate the performance of pre-trained models produced by different rounds in the pre-training stage. The model at the 120,000th step is selected for the experiments in Sec. 4.2 and Sec. 4.3. In this section, we select the models at the 30,000th, 60,000th, and 90,000th steps on the Cross-Platform (Android) and CSTNET-TLS 1.3 datasets, respectively, and the learning rates of the fine-tuning are 1e-4 and 6e-5, respectively.

The fine-tuning results of pre-trained model at different training steps are shown in Figure 5. The left half of Figure 5 (Figure 5a and Figure 5c) shows both the pre-training task

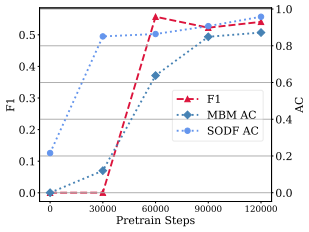
Datasets	AC	PR	RC	F1
Cross-Platform(Android)	0.5026 (-0.2343)[¶]	0.2247(-0.3653)	0.2378(-0.3348)	0.2184(-0.3460)
Cross-Platform(iOS)	0.0253(-0.4554)	0.0001(-0.4002)	0.0053(-0.3667)	0.0003(-0.3696)
CSTNET-TLS 1.3	0.6973(-0.1224)	0.6743(-0.1321)	0.6588(-0.1439)	0.6586(-0.1428)
ISCX-VPN(Service)	0.2603(-0.6644)	0.0434(-0.8834)	0.1667(-0.7500)	0.0688(-0.8517)
ISCX-VPN(App)	0.2414(-0.5586)	0.0219(-0.7456)	0.0909(-0.6127)	0.0354(-0.6605)
USTC-TFC	0.9082(-0.0684)	0.9136(-0.0659)	0.9144(-0.0633)	0.9109(-0.0675)

[¶] The values in parentheses are the metric difference between the model without pre-training and TrafficFormer.

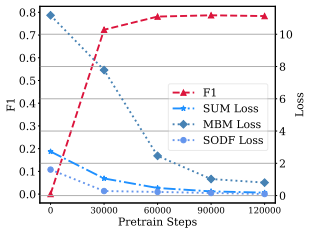
TABLE 10: The performance of model without pre-training on traffic classification tasks.



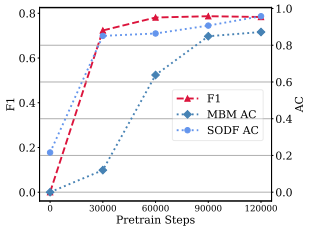
(a) Pre-training task loss and fine-tuning F1 scores on Cross-Platform (Android) dataset.



(b) Pre-training task accuracy and fine-tuning F1 scores on Cross-Platform (Android) dataset.



(c) Pre-training task loss and fine-tuning F1 scores on CSTNET-TLS 1.3 dataset.



(d) Pre-training task accuracy and fine-tuning F1 scores on CSTNET-TLS 1.3 dataset.

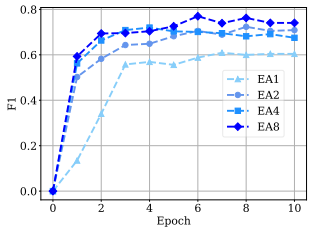
Figure 5: Downstream fine-tuning results of pre-trained models at different training steps.

loss and fine-tuning F1 scores. The right half of Figure 5 (Figure 5b and Figure 5d) shows both the pre-training task accuracy and fine-tuning F1 scores. It can be seen that the accuracy of the SODF task reaches a relatively high level first compared to the MBM task. This indicates that the MBM task is more difficult compared to the SODF task.

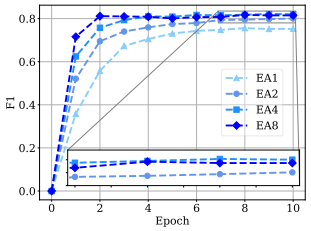
From Figure 5a and Figure 5b, it can be seen that when the number of pre-training steps is less than 30,000 steps, the classification performance of Cross-Platform (Android) is relatively poor. When the number of pre-training steps reaches 60,000, the F1 score improves rapidly. When the number of pre-training steps exceeds 60,000, the F1 score decreases slightly. However, the classification accuracy actually increases gradually, and the accuracies corresponding to 60,000 steps, 90,000 steps and 120,000 steps are 68.43%, 69.83% and 71.35% respectively. Correspondingly, the accuracy of the MBM task is also significantly improved after 30,000 steps. Therefore, the classification performance of Cross-Platform may be more relevant to the MBM task.

From Figure 5c and Figure 5d, it can be seen that when the number of pre-training steps reaches 30,000, the performance of website fingerprinting on the CSTNET-TLS 1.3 dataset is already quite impressive. When the number of pre-training steps reaches 60,000, the F1 score is improved by about 6%. When the number of pre-training steps continues to increase, the F1 score does not improve significantly. Correspondingly, the accuracy of the SODF task is already high at 30,000 steps, and the accuracy does not increase significantly when the number of pre-training steps is increased. This suggests that the classification performance of CSTNET-TLS 1.3 may be more relevant to the SODF task.

Impact of Data Enhancement. We already show the performance of TrafficFormer with data enhancement in Sec. 4.2. However, in order to keep the same amount of training data for a fair comparison, the data is augmented by a factor of 5, and the augmented data is trained for four rounds. In this section, we explore the effect of different data augmentation modes, *i.e.,* different data enhancement factors. For each augmentation mode, the data obtained is trained for 10 rounds.



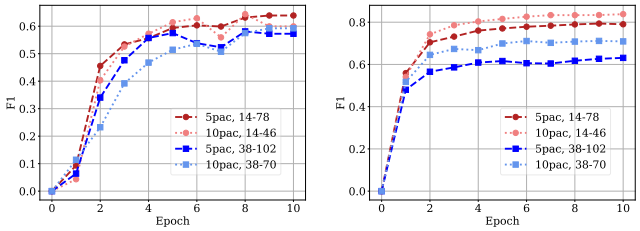
(a) ISCX-VPN(App) dataset



(b) CSTNET-TLS 1.3 dataset

Figure 6: Results of different data enhancement modes.

The performance of different data enhancement modes is shown in Figure 6, with EAN representing data enhanced by N times. From Figure 6a, it can be seen that the larger the data enhancement factor is, the better the results are in each round, and the higher the final F1 score is achieved. After data enhancement by a factor of 8, the classification F1 score on ISCX-VPN(App) reaches 76.98%, which is significantly higher than the result (71.29%) showed in Sec. 4.2. As can be seen in Figure 6b, the larger the data enhancement factor, the better the results in the first few rounds. In the final round, the performance order of different data



(a) ISCX-VPN(App) dataset (b) CSTNET-TLS 1.3 dataset

Figure 7: Results of different input contents.

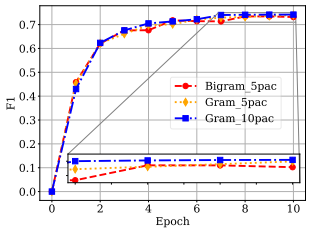


Figure 8: Results of different traffic representations.

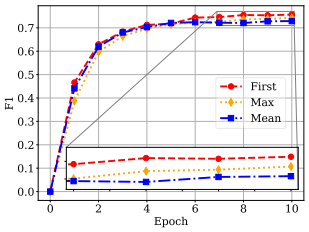


Figure 9: Results of different sequence representations.

enhancement factors is: EA8≈EA4>EA2>EA1. After the data enhancement factor reaches 4, the F1 score is no longer improved. This shows that the data enhancement factor is not as large as the better, after reaching a certain factor, the data enhancement does not bring any more improvement.

Impact of Input Content. In the above sections, we select the first 5 packets, and each packet is taken 64 bytes after the Ethernet layer, *i.e.,* the 14th through 78th bytes of the packet. In this section, we vary the number of packets in the input as well as the bytes selected for each packet to explore the effect of the input content of TrafficFormer on the fine-tuning. Specifically, we construct four different input contents. *5pac,14-78*: the first 5 packets, with the 14th through 78th bytes (64 bytes after the Ethernet layer), *10pac,14-46*: the first 10 packets, with the 14th through 46th bytes (32 bytes after the Ethernet layer), *5pac,38-102*: the first 5 packets, with the 38th through 102nd bytes (64 bytes after the port number), *10pac,38-70*: the first 10 packets, with the 38th through 70th bytes (32 bytes after the port number). The total number of bytes in the four different inputs is the same. The ISCX-VPN (App) and CSTNET-TLS 1.3 datasets are selected for evaluation in this section.

Figure 7 shows the performance of different input contents. As we can see from Figure 7a, the performance of *5pac,14-78* is higher than that of *5pac,38-102*, suggesting that valuable information may lie in the difference set of the two data, *i.e.,* the 14th through 38th bytes. *10pac,38-70* is a little higher than *5pac,38-102*, which may be releated to using more packets. Even though it uses information from 10 packets, its F1 score is still lower than *5pac,14-78*, further indicating that valuable information is within the 14th through 38th bytes. If valuable information is not included in the input, more packets will not necessarily lead to better results. *10pac,14-46* performs the best among the four inputs, which is related to both the inclusion of valuable information and the use of more packets. From Figure 7b, it can be seen that the F1 score of *5pac,14-78* is significantly higher than that of *5pac,38-102*, indicating that valuable information may be in the difference set of the two data, *i.e.,* the 14th through 38th bytes. The performance of *10pac,14-46* is better than *5pac,14-78*, and *10pac,38-70* is better than *5pac,38-102*, showing that a greater number of packets tends to yield better results. The fact that *5pac,14-78* outperforms *10pac,38-70* also shows that valuable information is more important than the number of packets. In summary, (i) the more packets inputted the better the performance achieved, (ii) the valuable information part is different across datasets, (iii) valuable information is more important than the number of packets.

Impact of Traffic Representation. As described in Sec. 3.2.1, hexadecimal packets are transformed into bigram form for input into TrafficFormer. In this section, we explore the effect of different traffic representations of TrafficFormer on the fine-tuning. We define the gram form here for comparison. For example, 450b12 06 are two adjacent fields. Bigram gets 450b 0b12 1206, and Gram gets 450b 1206. The bigram overlaps the bytes and the final input is twice as long as the gram input. Bigram contains richer information compared to gram, *i.e.,* one more word (0b12) belonging to one field. In this section, we perform the pre-training in gram form, and the fine-tuning data is also processed into gram form. The CSTNET-TLS 1.3 dataset is selected for evaluation in this section. *Gram_5pac* represents the input in gram form, and 5 packets are inputted for fine-tuning. As shown in Figure 8, *Gram_5pac* is close to *Bigram_5pac*. The input lengths of *Gram_10pac* and *Bigram_5pac* are the same and they are close in performance. This indicates that the traffic representation has little effect on the downstream fine-tuning traffic classification. We think the good performance of gram is related to the BPE method. For example, when 0b12 is a useful feature, BPE may produce the subwords ##0b and 12##. Then the model learns the relationship between them. Instead, bigram can provide richer information directly in the input, thus reducing the learning burden of model. Longer-sequence overheads are acceptable in the trend of the model's increasing ability to handle long sequences.

Impact of Sequence Representation. In TrafficFormer, the input to the classification layer is the output of the final representation layer, which consists of the representations of all tokens in the input sequence. The representation of the first token, *i.e.,* [CLS], is used as the input to the classification layer in the fine-tuning because it synthesizes the information from the other tokens. In this section, we explore the effect of other sequence representations. The [CLS] representation as the sequence representation is denoted as *First*, the sequence representation formed by the maximum value on each dimension in all token representations is denoted as *Max*, and the sequence representation formed by the average value on each dimension in all token representa-

tions is denoted as *Mean*. The CSTNET-TLS 1.3 dataset is selected for evaluation in this section. As shown in Figure 9, the best result is *First*, followed by *Max* and finally *Mean*. The reason may be that [CLS] representation is also used as a sequence representation for the classification task in the pre-training stage. The reason for the superiority of *Max* over *Mean* may be that it retains more unique information.

5. Discussion

In this section, we discuss the design of the SODF task and the possible limitations of TrafficFormer performing different tasks.

The SODF Task. The SODF task allows the model to learn the packet order, the packet direction, and the packet slave flow (flow that the packet belongs to) together, by classifying different burst segments into multiple categories. These can also be learned separately by three individual pre-training tasks. For example, the packet order can be learned by predicting the word order (e.g., StructBert [67]), and the packet direction and packet slave flow can be learned by two binary classification tasks. However, as the number of pre-training tasks increases, it becomes more difficult to balance multiple pre-training tasks, which would be an interesting future work. Uncertain weighting technique [68] in multi-task learning may be a good solution.

Limititions. (i) Limited flow input length. Although the attention mechanism used in TrafficFormer can handle arbitrarily long data, it may incur memory overflow [69] and distraction [70] when the input length becomes longer. It prevents TrafficFormer from processing too much packet information. We can utilize mechanisms such as sliding windows (i.e., attending only to words within a finite window of surroundings) [71] to mitigate the issue. (ii) Processing flows with only raw packets. TrafficFormer only uses the raw payload of the packet. Some features such as packet timestamps are not part of the packet payload and are therefore not directly used in the TrafficFormer. This may impact the effectiveness of detection tasks that rely on packet intervals (e.g., DDoS detection). These features (e.g., packet interval or other expert features) can be encoded to tokens to add to the payload sequence, or use cross attention technique to build connections with payload sequence [72]. (iii) Single-flow detection. TrafficFormer is a flow-by-flow detection scheme and thus the input only contains one flow information. Under multi-flow scenarios (e.g., accessing a web page will generate multiple flows), the performance of TrafficFormer may decrease. We can extend the input to multidimensional data and then perform multidimensional attention for multi-flow detection (e.g., Space-Time Attention in TimeSformer [73]) .

6. Conclusion

In this paper, we propose an efficient pre-training model TrafficFormer for traffic data, which learns the basic semantics of traffic through a large amount of unlabeled data and generalizes it to the downstream traffic classification task with limited amount of labeled data. Specifically, in the pre-training stage we propose the SODF multi-classification task, so that the pre-trained model can learn the direction and order information of the packets in the traffic, enhancing the representation of traffic data. In the fine-tuning stage, we propose a traffic data augmentation approach RIFA, which reduces the traffic model's reliance on meaningless data, allowing it to find valuable information more quickly. In addition, we propose a new traffic model evaluation task, the protocol understanding task, which evaluates pre-trained models on their cognition of protocol interaction logic. We conduct evaluation on six traffic datasets, and TrafficFormer achieves optimal performance on both the traffic classification task and the protocol understanding task.

References

[1] T. Van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. Choffnes, M. Van Steen, and A. Peter, "Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Network and distributed system security symposium (NDSS)*, vol. 27, 2020.

[2] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.

[3] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2017.

[4] T. Shapira and Y. Shavitt, "Flowpic: Encrypted internet traffic classification is as easy as image recognition," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 680–687.

[5] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 2019, pp. 1171–1179.

[6] 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.

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[8] M. E. Peters, W. Ammar, C. Bhagavatula, and R. Power, "Semi-supervised sequence tagging with bidirectional language models," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, R. Barzilay and M.-Y. Kan, Eds. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1756–1765. [Online]. Available: https://aclanthology.org/P17-1161

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:52967399

[10] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[11] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[12] H. Y. He, Z. G. Yang, and X. N. Chen, "Pert: Payload encoding representation from transformer for encrypted traffic classification," in *2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*. IEEE, 2020, pp. 1–8.

[13] 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 *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*. ACM, 2022, pp. 633–642.

[14] R. Zhao, M. Zhan, X. Deng, Y. Wang, Y. Wang, G. Gui, and Z. Xue, "Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 5420–5427.

[15] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 16 000–16 009.

[16] H. Bao, L. Dong, S. Piao, and F. Wei, "Beit: Bert pre-training of image transformers," *arXiv preprint arXiv:2106.08254*, 2021.

[17] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[18] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9729–9738.

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PMLR, 2020, pp. 1597–1607.

[20] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," *arXiv preprint arXiv:2104.08821*, 2021.

[21] 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.

[22] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[24] X. Chen, S. Xie, and K. He, "An empirical study of training self-supervised vision transformers," in *CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9620–9629.

[25] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 07, 2020, pp. 13 001–13 008.

[26] G. Mariani, F. Scheidegger, R. Istrate, C. Bekas, and C. Malossi, "Bagan: Data augmentation with balancing gan," *arXiv preprint arXiv:1803.09655*, 2018.

[27] A. Antoniou, A. Storkey, and H. Edwards, "Data augmentation generative adversarial networks," *arXiv preprint arXiv:1711.04340*, 2017.

[28] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[29] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long text generation via adversarial training with leaked information," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[30] L. Vu, D. Van Tra, and Q. U. Nguyen, "Learning from imbalanced data for encrypted traffic identification problem," in *Proceedings of the 7th Symposium on Information and Communication Technology*, 2016, pp. 147–152.

[31] P. Oeung and F. Shen, "Imbalanced internet traffic classification using ensemble framework," in *2019 International Conference on Information Networking (ICOIN)*. IEEE, 2019, pp. 37–42.

[32] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[33] E. Horowicz, T. Shapira, and Y. Shavitt, "A few shots traffic classification with mini-flowpic augmentations," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 647–654.

[34] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers & Security*, vol. 82, pp. 156–172, 2019.

[35] S. T. Jan, Q. Hao, T. Hu *et al.*, "Throwing Darts in the Dark? Detecting Bots with Limited Data using Neural Data Augmentation," in *Proc. SP*, 2020.

[36] Y. Guo, G. Xiong, Z. Li, J. Shi, M. Cui, and G. Gou, "Ta-gan: Gan based traffic augmentation for imbalanced network traffic classification," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[37] A. W. Moore and D. Zuev, "Internet Traffic Classification using Bayesian Analysis Techniques," in *Proc. SIGMETRICS*, 2005.

[38] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic Classification on the Fly," *SIGCOMM-CCR*, 2006.

[39] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.

[40] K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K. Hamlen, and B. Thuraisingham, "Adaptive encrypted traffic fingerprinting with bi-directional dependence," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 177–188. [Online]. Available: https://doi.org/10.1145/2991079.2991123

[41] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2017.

[42] V. Rimmer, D. Preuveneers, M. Juárez, T. V. Goethem, and W. Joosen, "Automated Website Fingerprinting through Deep Learning," in *Proc. NDSS*, 2018.

[43] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1928–1943.

[44] T. Shapira and Y. Shavitt, "Flowpic: Encrypted internet traffic classification is as easy as image recognition," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 680–687.

[45] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in *Proc. NDSS*, 2018.

[46] 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.

[47] K. Lin, X. Xu, and H. Gao, "Tscrnn: A novel classification scheme of encrypted traffic based on flow spatiotemporal features for efficient management of iiot," *Computer Networks*, vol. 190, p. 107974, 2021.

[48] K. Bartos, M. Sofka, and V. Franc, "Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants," in *Proc. USENIX Security*, 2016.

[49] D. Han, Z. Wang, W. Chen, K. Wang, R. Yu, S. Wang, H. Zhang, Z. Wang, M. Jin, J. Yang *et al.*, "Anomaly detection in the open world: Normality shift detection, explanation, and adaptation," in *30th Annual Network and Distributed System Security Symposium (NDSS)*, 2023.

[50] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New Directions in Automated Traffic Analysis," in *Proc. CCS*, 2021.

[51] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime Robust Malicious Traffic Detection via Frequency Domain Analysis," in *Proc. CCS*, 2021.

[52] ——, "Frequency Domain Feature Based Robust Malicious Traffic Detection," *ToN*, to appear.

[53] C. Fu, Q. Li, and K. Xu, "Detecting Unknown Encrypted Malicious Traffic in Real Time via Flow Interaction Graph Analysis," in *Proc. NDSS*, 2023.

[54] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6203–6220.

[55] J. Li, S. Wu, H. Zhou *et al.*, "Packet-Level Open-World App Fingerprinting on Wireless Traffic," in *Proc. NDSS*, 2022.

[56] X. Deng, Q. Yin, Z. Liu, X. Zhao, Q. Li, M. Xu, K. Xu, and J. Wu, "Robust multi-tab website fingerprinting attacks in the wild," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 1005–1022.

[57] E. Kaiser and J. C. Trueswell, "The role of discourse context in the processing of a flexible word-order language," *Cognition*, vol. 94, no. 2, pp. 113–147, 2004.

[58] Q. Cao, T. Kojima, Y. Matsuo, and Y. Iwasawa, "Unnatural error correction: Gpt-4 can almost perfectly handle unnatural scrambled text," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 8898–8913.

[59] K. Al-Naami, S. Chandra, A. Mustafa, L. Khan, Z. Lin, K. Hamlen, and B. Thuraisingham, "Adaptive encrypted traffic fingerprinting with bi-directional dependence," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016, pp. 177–188.

[60] W. Chen and Z. Qian, "{Off-Path}{TCP} exploit: How wireless routers can jeopardize your secrets," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1581–1598.

[61] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path tcp exploits of the mixed ipid assignment," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1323–1335.

[62] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," Internet Requests for Comments, September 2014. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7323.txt

[63] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Encrypted and VPN Traffic using Time-Related Features," in *Proc. ICISSP*, 2016.

[64] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *2018 International Carnahan conference on security technology (ICCST)*. IEEE, 2018, pp. 1–7.

[65] J. Ren, D. Dubois, and D. Choffnes, "An international view of privacy risks for mobile apps," 2019.

[66] 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.

[67] W. Wang, B. Bi, M. Yan, C. Wu, Z. Bao, J. Xia, L. Peng, and L. Si, "Structbert: Incorporating language structures into pre-training for deep language understanding," *arXiv preprint arXiv:1908.04577*, 2019.

[68] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7482–7491.

[69] W. Xiong, J. Liu, I. Molybog, H. Zhang, P. Bhargava, R. Hou, L. Martin, R. Rungta, K. A. Sankararaman, B. Oguz *et al.*, "Effective long-context scaling of foundation models," *arXiv preprint arXiv:2309.16039*, 2023.

[70] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.

[71] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[72] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4195–4205.

[73] G. Bertasius, H. Wang, and L. Torresani, "Is space-time attention all you need for video understanding?" in *ICML*, vol. 2, no. 3, 2021, p. 4.