Packet Representation Learning for Traffic Classification

Xuying Meng
ICT, CAS
Purple Mountain Laboratories
mengxuying@ict.ac.cn

Yequan Wang
Beijing Academy of Artificial Intelligence
tshwangyequan@gmail.com

Runxin Ma
ICT, CAS
UCAS
marunxin@ict.ac.cn

Haitong Luo
ICT, CAS
UCAS
luohaitong@ict.ac.cn

Xiang Li
Alibaba Group
yuanye.lx@alibaba-inc.com

Yujun Zhang*
ICT, CAS
UCAS
nrcyujun@ict.ac.cn

ABSTRACT

With the surging development of information technology, to provide a high quality of network services, there are increasing demands and challenges for network analysis. As all data on the Internet are encapsulated and transferred by network packets, packets are widely used for various network traffic analysis tasks, from application identification to intrusion detection. Considering the choice of features and how to represent them can greatly affect the performance of downstream tasks, it is critical to learn high-quality packet representations. In addition, existing packet-level works ignore packet representations but focus on trying to get good performance with independent analysis of different classification tasks. In the real world, although a packet may have different class labels for different tasks, the packet representation learned from one task can also help understand its complex packet patterns in other tasks, while existing works omit to leverage them.

Taking advantage of this potential, in this work, we propose a novel framework to tackle the problem of packet representation learning for various traffic classification tasks. We learn packet representation, preserving both semantic and byte patterns of each packet, and utilize contrastive loss with a sample selector to optimize the learned representations so that similar packets are closer in the latent semantic space. In addition, the representations are further jointly optimized by class labels of multiple tasks with loss of reconstructed representations and of class probabilities. Evaluations demonstrate that the learned packet representation of our proposed framework can outperform the state-of-the-art baseline methods on extensive popular downstream classification tasks by a wide margin in both the close-world and open-world scenario.

CCS CONCEPTS

• **Computing methodologies → Knowledge representation and reasoning**; • **Networks → Network properties**.

*Corresponding Author

KDD '22, August 14–18, 2022, Washington, DC, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9385-0/22/08.
https://doi.org/10.1145/3534678.3539085

KEYWORDS

Representation Learning, Traffic Classification, Packet Representation, Contrastive Learning

ACM Reference Format:
Xuying Meng, Yequan Wang, Runxin Ma, Haitong Luo, Xiang Li, and Yujun Zhang. 2022. Packet Representation Learning for Traffic Classification. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534678.3539085

1 INTRODUCTION

With the surging development of information technology, even in the local area network of an Internet company (e.g., in the data center, office Intranet, et. al.), there can be various computers, IoT devices, routers, and even their own domain name systems. To provide high-quality network services, there are increasing demands and challenges for traffic analysis. Traffic classification, characterizing network traffic with appropriate class labels, is important to many analysis demands, such as quality of service (QoS) control, security control, pricing, resource usage planning, and access control [19]. As all data on the Internet are encapsulated and transferred by network packets, packets are the fundamental elements to offer rich information for these downstream tasks. It is important to learn packet representations to capture latent information gleaned from datasets of diverse classification tasks, which will also help further studies on network traffics, such as zero-day attack detection [15].

Existing works learn the representations independently for different traffic data formats (e.g., plaintext, encrypted, compressed) and different classification tasks (e.g., intrusion detection, application classification). We roughly categorize them into *feature-based* and *byte-based* ones. For *feature-based* models, Ashfaq et al. [2] construct packet representations with header features and utilize unlabeled samples to improve the performance of intrusion detection. Hypolite et al. [8] utilize the deep packet inspection (DPI) with both header and payload information to match the specific regex of each class. However, these works are only applicable to plaintext traffic and require extra cost and labor for feature engineering. To tackle these problems, recent *byte-based* methods have aroused great attention. Lotfollahi et al. [14] and Wang et al. [25] make the initial trials to learn packet representations with the hex number of each byte, and utilize CNN for application classification. Casino et al. [4] utilize hex bytes of each packet, and further distinguish between encrypted and compressed traffic. As there are always new packet patterns in the real world (i.e., open-world problem),

```
<IP  version=4 ihl=5 tos=0x8 len=1360 id=14677 flags=DF
frag=0 ttl=37 proto=tcp chksum=0x4928 src=204.236.238.164
dst=10.8.8.138 |<TCP  sport=1639 dport=41005
seq=4157736792 ack=86048704 dataofs=8 reserved=0 flags=A
window=235 chksum=0x255f urgptr=0 options=[('NOP', None),
('NOP', None), ('Timestamp', (447078848, 2304838))] |<Raw
load='\xc4\x95\x94\xd8\x02\xaa\xa62\xcf\xf4\xae\x0c\x9c\x9a\
x97\xeb\x06\xe8\x1f\x98\x7f\x13\x9e\x0f\x8d\xc7\xf0\x0f\xe9\x
ca\x82\xbe\xa7\x81\x07\xe7\xc9\x15\x85\x0e\xb5\x85\xca\xc8
\xdf\xd5\x1a\xd4\x92\x8a\xd5\x16\xf7\x14\xa4\xeb\x92\xf9\xc
a\xf1\xb9\xd8\x80\xad\x1f\xa7\xb7…'|>>>
```

Figure 1: A packet example, where the raw payload data are packed with TCP and IP headers at the corresponding protocol layers. It contains both unencrypted and encrypted data, and has class labels like "benign", "FTP", "file transfer" and "VPN" for different classification tasks.

Zhang et al. [28] add a step of filtering unknown packets after the hex byte representation learning for application classification.

Despite the remarkable progress made towards traffic classification, we argue that although each packet has different characteristics (e.g., class labels) from the perspectives of different tasks, the packet representation learned from one task can help understand its complex packet patterns in the other tasks, while existing works ignore to leverage them. For example, as illustrated in Figure 1, a "benign" packet of "file transfer" traffic by " FTP" through the "VPN" connection has four labels for classification tasks of "intrusion detection", "application classification", "service identification" and "VPN detection", respectively. These labels of diverse classification tasks will help learn the complex packet patterns and even help find new packet patterns, which will not only benefit the existing tasks but also the further studies on traffic. As such, in this work, we propose to investigate the novel problem of learning packet representations for diverse traffic classification tasks.

However, learning packet representations for diverse classification tasks remains non-trivial, mainly owing to the following reasons: **(1)** From the perspectives of *learning packet representations*, it is difficult to generally encode different types of packets. As illustrated in Figure 1, packet data are encapsulated at each protocol layer from down to top [12]. With different protocols, the packets can be in different header structures, packet lengths, and data formats. **(2)** From the perspectives of *diverse traffic classification tasks*, it is hard to utilize the uniformed packet representations to adapt to different classes and tasks. Even for the same packet, different classes and different tasks pay attention to different parts of its representation. For example, for protocol identification, header information will make more contributions; while for attack detection, header information may be more important for land attacks while payload information may be more vital for XSS attacks.

To address the aforementioned challenges, in this work we propose **PacRep**, a novel framework to learn **Pac**ket **Rep**resentations for traffic classification. PacRep is composed of two modules, i.e., a *packet encoding* module and a *joint tuning* module. In the *packet encoding* module, we regard each packet as a text with tokens, and learn an effective encoder regularized from contrastive loss of selected positive and negative samples. In the *joint tuning* module, we optimize the model by the adaptive hidden representations and the predicted probability for each class. Also, since packets can be classified into only one class for each task, to keep the effectiveness among diverse downstream classification tasks, we also jointly regularize the predicted probability for multiple tasks, which will also further optimize the encoder in turn. To summarize, our main contributions are three-fold:

- **Problem**: To the best of our knowledge, we are the first to investigate the novel problem of general packet representation learning for diverse traffic classification tasks with both encrypted and unencrypted information.
- **Algorithm**: We propose a novel PacRep framework, which learns effective packet representations optimized by contrastive loss from positive and negative samples, and further tuned by labels from multiple classes and tasks.
- **Evaluation**: We perform extensive experiments to demonstrate the effectiveness of our PacRep on six downstream tasks, showing that PacRep outperforms state-of-the-art baselines on all these classification tasks by a wide margin.

2 RELATED WORK

In this section, we briefly review prior works on traffic classification and contrastive representation learning methods.

2.1 Traffic Classification

Traffic classification is important to many applications [19], and recent years have witnessed increasing deep learning approaches [4, 14–20, 27, 28, 30]. We roughly categorize them into feature-based and byte-based ones.

For the feature-based approaches, they extract representative features as input data for the models [15, 16, 18, 20, 27, 30]. As features of individual packets are limited, most feature-based models collect statistical features of multiple packets from perspectives of flows, sessions and time series. For example, Shen et al. [20] learn a fusion feature selection method to construct a strong classifier for encrypted flows of decentralized applications. Also, as classification is a traditional machine learning task, some general classification models can be directly used for traffic classification based on features [18, 27]. For example, Pang et al. [18] leverage a few labels to learn anomaly scores based on a feature representation learner. However, except for the difficulty of feature engineering, for network traffic, the statistic features of nodes, time series, flows and sessions are relatively hard to obtain, and the traffic encryption further increases the difficulty of finding representative features.

For the byte-based approaches, these works directly use bytes in each raw packet, thus do not need complicated feature processing [4, 14, 17, 28]. However, most of them translate bytes into fixed lengths of hex numbers, treating them as a pixel in an image [1, 9], thus the rich semantic information (especially the plaintext of the unencrypted parts) is wasted. For example, Lotfollahi et al. [14] identify different application traffic by CNN with packet contents in hexadecimal format. To better adapt to the real world with new classes, Zhang et al. [28] add a step of filtering unknown packets before the application classification. Specially, some of them notice the similarity of packets and texts, and utilize natural language processing techniques to learn the semantic meaning of the hex data. Min et al. [17] make an initial trial to treat each hex as a token and utilize the textCNN [11] to learn token embeddings. However, it can make limited improvement with a small vocabulary space of the hex number.

Most of these feature-based and byte-based approaches are trained to achieve good performance for a specific task. Huang et al. [7] notice the mutual improvement of different tasks, and train a multi-task learning model to obtain labels for these tasks. However, it is still limited to specific multiple tasks. Although we can produce labels like multi-task learning, we have two main differences: (1) Our target is to learn an encoder for better packet representation. The tuning components for these tasks are only used to help learn a more accurate representation encoder; (2) With the well-trained encoder, we can train different tuning components for new tasks and new labels, and does not need to be limited by the tasks we used for training. These can not be accomplished by multi-task learning. It is still challenging to learn packet representations for these diverse downstream tasks.

2.2 Contrastive Representation Learning

Contrastive representation learning learns discriminative representations by contrastive positive and negative samples, and has been successfully applied for computer vision (CV) and natural language processing (NLP) in recent years [24]. The high-level idea is to encourage the model to maximize the mutual information of positive instance pairs so that two views or augmentations of the similar samples are close to the source in latent semantic space [13]. For comparison, it is important to get suitable positive and negative samples. For example, there are many works that learn image representations with generated negative samples using a multiple-state augmentation pipeline, consisting of color jitter, random flop, and cropping [3]. Zhu et al. [31] learn graph representations with designed augmentation schemes of selecting positive samples from both topological and semantic perspectives. Kang et al. [10] learn balanced feature spaces for self-supervised learning by randomly selecting k instances with the same labels as the positive samples.

However, for network data, there are different criteria of importance for different traffic tasks for both positive and negative samples. The uniformed strategy of sample selection can not adapt to the diverse downstream tasks. For example, even in the same class, there are different packet patterns (e.g., there are different kinds of attacks in the abnormal class in malware detection), we can not simply randomly utilize k samples with the same class as positive samples. Thus, except for the data format, the existing strategies of sample selection can not work well for network data.

3 PROBLEM DEFINITION

In this paper, we aim to learn high-quality packet representations for downstream tasks. In detail, our objective is to learn a packet encoder that can be used to produce packet representations and hidden representations of input tokens for any unknown packet, and lay a good foundation for downstream tasks, such as application classification, intrusion detection, and malware detection.

Suppose we have a packet set $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$, where a_i is the i-th packet and $|\mathcal{A}| = n$. As shown in Figure 1, in each packet, there are headers and payloads information, where both can be encrypted or unencrypted. The unencrypted is retained as text and the encrypted is kept as hexadecimal bytes. To utilize both the encrypted and unencrypted data, we represent each packet a_i as an input sequence $\mathbf{x}_i = [w_{i,1}, w_{i,2}, \dots]$, where $w_{i,j}$ is j-th token in $\mathbf{x}_i$. For the unencrypted part, the plaintext is split by the tokenizer to generate tokens. Also, for the encrypted part, the hexadecimal bytes split by the same tokenizer are treated as tokens. Since packet length is unfixed, the vector $\mathbf{x}_i$ has unfixed numbers of tokens.

What's more, each packet a_i has its label vector $\mathbf{y}_i$. Specially, in different tasks, each packet has different labels. With labels for S downstream tasks, we have $\mathbf{y}_i \in \mathbb{R}^S$ for each packet $\mathbf{x}_i$. For each task, the numbers of classes are unfixed, and we define $\mathbf{y}_{i,j} \in \{1, 2, \dots, |\mathcal{T}_j|\}$, where the classification task j has class set $\mathcal{T}_j$ and the number of classes in this task is $|\mathcal{T}_j|$. Also, with limited labels, not all tasks will be available for all packets, and we set $\mathbf{y}_{i,j} = \text{None}$ if we do not have a label for packet a_i in task j. What's more, as classes in the downstream tasks can be overlapped, we do not need to separate them based on specific tasks, and the total number of classes is denoted as $C = |\mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_j \cup \cdots \cup \mathcal{T}_S|$. In summary, along with the packet set $\mathcal{A}$, we have the input sequence set $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, and the label set $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$, where the i-th packet a_i has $\mathbf{x}_i$ and $\mathbf{y}_i$. In addition, we denote the latent vector of packet a_i as $\mathbf{v}_i \in \mathbb{R}^d$, where d is the hidden dimension of packet representations. Suppose there are L tokens in a_i, the latent matrix is presented as $\mathbf{H}_i \in \mathbb{R}^{d \times L}$, and each token is in d dimension.

With these notations, we can formally define the target of packet representation learning as follows:

Given the input set $\mathcal{X}$ and the label set $\mathcal{Y}$ of multiple tasks, the goal of this work is to: (1) learn a packet representation encoder $f : \mathbf{x}_i \rightarrow \mathbf{H}_i, \mathbf{v}_i$; and (2) obtain accurate $\mathbf{y}_i$ on downstream classification tasks by a probability function $g : \mathbf{H}_i, \mathbf{v}_i \rightarrow \mathbf{y}_i$.

4 PROPOSED FRAMEWORK

In this section, we will present our proposed PacRep. The overall framework is introduced first.

4.1 Framework Overview

The overall architecture for the proposed model is shown in Figure 2. PacRep consists of two novel modules: *packet encoding* module and *joint tuning* module. Aiming at learning packet representations for diverse downstream classification tasks, the solution is two-fold. On one hand, we learn the encoder to provide effective representations of each packet by the *packet encoding* module. On the other hand, we tune the representations to jointly adapt to each class and each task by the *joint tuning* module.

For the *packet encoding* module, the packet data can be regarded as a text with different kinds of tokens. Given packets with their input sequences, the transformer-style [23] encoder will produce the latent representation, with regularization from contrastive loss of selected positive and negative samples. For each packet a_i, the latent representation $\mathbf{H}_i$ and $\mathbf{v}_i$ will serve as the input to following classes in the *joint tuning* module, which can also be regarded as a pre-training procedure for downstream tasks.

For the *joint tuning* module, we tune the encoder for classes and tasks, inspired by the capsule design in [26]. To tune for classification, we learn the adaptive representation vector $\mathbf{v}_{i,c}$ for each class c

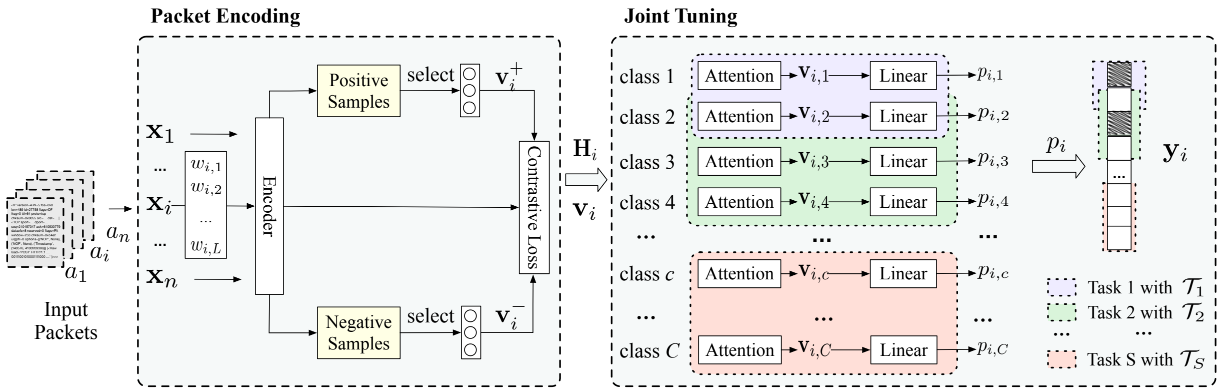


Figure 2: Our proposed Packet Representation framework PacRep. With n packets, C classes and S tasks, we take the packet a_i with L tokens as an example. We first learn to encode the packet into H_i and v_i with help of selected positive and negative samples. To joint tune for downstream classes and tasks, we learn to predict $\mathbf{y}_i$ with each class set $\mathcal{T}_j$ in each task j.

by attention mechanism, and output the corresponding probability $p_{i,c}$. These adaptive representations $\mathbf{v}_{i,c}$ and corresponding $p_{i,c}$ will also in turn help optimize the encoder. With C classes and S tasks, although not all tasks will be active for all packets, each packet can have several labels for the active tasks. Among the active tasks, the packet can be classified into only one class for each task.

In addition, to tune for tasks, we adjust to downstream tasks with the probability $p_{i,c}$ and the adaptive importance of the class sets $\{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$. During training, the objective is to maximize the probability corresponding to the ground truth class labels, and to minimize the probability of the rest classes for all tasks. As different tasks have different convergence rates, the importance of each task will be weighted based on the corresponding class sets. Also, in the testing process, each packet can be classified into several classes for the corresponding active tasks, and in each task, only one class with the highest probability will be retained.

4.2 Packet Encoding

As packet data can be regraded as text, we can utilize the natural language processing techniques to encode packets. In essence, the packet encoding module is composed of three key building blocks, including an initial encoder, a sample selector, and a contrastive loss. The details are as follows.

4.2.1 Initial Encoder. As shown in Figure 1, packet data can be both encrypted and unencrypted. The unencrypted plaintext can be directly separated by tokenizer to generate tokens. While for the encrypted data, we utilize the straightforward way to translate each byte into a hex number, and then use tokenizer to generate tokens. However, with hex format, some tokens are just numbers, which may make it hard to encode them into semantic latent vectors. Also, each byte can only represent 255 kinds of tokens in the vocabulary space (i.e., the maximal hex number is "ff"), which may be not big enough for their underlying semantic information. To learn effective packet representations, it requires further analysis, and we also conduct experiments on the token strategies in Section 5.4.

Given the input sequence $\mathbf{x}_i = \{w_{i,1}, w_{i,2}, \ldots, w_{i,L}\}$ for each packet a_i with token length L, we utilize a pre-trained transformer as the text encoder to obtain the initial representations, as shown in the left of Figure 2. With a pre-trained transformer encoder such as BERT [5], we can obtain

$$\mathbf{H}_i, \mathbf{v}_i = \text{Encoder}(\mathbf{x}_i), \tag{1}$$

where in the latent matrix $\mathbf{H}_i \in \mathbb{R}^{d \times L}$, each token is encoded in d dimension, and the packet is encoded as latent vector $\mathbf{v}_i \in \mathbb{R}^d$.

4.2.2 Sample Selector. We utilize triplets for encoder training, to semantically regularize it with positive and negative samples to pull close neighbors together and push apart non-neighbors [6]. Since the sampling strategy directly affects the regularization performance, we present the sample selector first.

For each packet a_i, we have the positive samples $\mathcal{D}_i^+$ and the negative samples $\mathcal{D}_i^-$. For $\mathcal{D}_i^+$, each $a_j \in \mathcal{D}_i^+$ should have the same labels as the anchor sample a_i among all the S tasks. For each $a_j \in \mathcal{D}_i^-$, none of their labels should be the same as the anchor sample a_i among all the S tasks. However, with insufficient labels, we can not ensure all packets have labels of all S tasks, and it might lead to mistakes with the existing labels. For example, for an anchor packet with the label "malicious", its positive sample may be "malicious" from different Malware. It will affect the accuracy of the representation, and make bad effects on the corresponding tasks. To tackle this problem, we separately regularize the encoder for different tasks. For each packet a_i, we randomly select k positive and negative samples based on its labels in $\mathbf{y}_i$. For tasks satisfying $\mathbf{y}_{i,j} \neq$ None, all packets in $\mathcal{D}_i^+$ have the same labels as a_i, and packets in $\mathcal{D}_i^-$ do not. Also, we do not utilize packet a_i for task j satisfying $\mathbf{y}_{i,j} =$ None. In this way, we utilize the mutual impacts of available labels while avoiding bad impacts of unavailable ones.

4.2.3 Contrastive Learning. With the selected k positive and negative samples, we expect a well-trained encoder to capture similarity in the latent semantic space, which can be achieved by

$$\ell_c = \sum_{i=1}^{n} \max(0, 1 - \sum_{j=1}^{k} \text{sim}(\mathbf{v}_i, \mathbf{v}_j^+)) + \sum_{i=1}^{n} \max(0, \tau + \sum_{j=1}^{k} \text{sim}(\mathbf{v}_i, \mathbf{v}_j^-)), \tag{2}$$

where τ is a hyper-parameter for margin, and $\mathbf{v}_j^+$ and $\mathbf{v}_j^-$ represent the packet representation of the j-th sample from the $\mathcal{D}_i^+$ and $\mathcal{D}_i^-$ respectively. $sim(\mathbf{v}_i, \mathbf{v}_j^+)$ is the similarity between $\mathbf{v}_i$ and $\mathbf{v}_j^+$, and we use cosine similarity here. For simplification, we set $k = 1$ in this paper.

4.3 Joint Tuning

With the optimization by the contrastive learning objective, we semantically regularize the packet encoder to provide the representations $\mathbf{H}_i$ and $\mathbf{v}_i$ for each packet a_i. To adapt to classes and tasks, we further tune the encoder accordingly.

4.3.1 Tuning for Classes. Since each token has different importance for recognizing different classes, directly utilizing the same $\mathbf{v}_i$ for all the classes cannot utilize this characteristic. To effectively weight the importance of each class, we utilize the attention mechanism [26] based on $\mathbf{H}_i$ by

$$\begin{aligned} z_{i,l} &= \mathbf{h}_{i,l}\mathbf{w}_{\gamma,c}, \\ \gamma_{i,l} &= \frac{exp(z_{i,l})}{\sum_{k=1}^{L} exp(z_{i,k})}, \\ \mathbf{v}_{i,c} &= \sum_{l=1}^{L} \gamma_{i,l}\mathbf{h}_{i,l}, \end{aligned} \tag{3}$$

where $\mathbf{h}_{i,l}$ is the embedding vector of the l-th token in $\mathbf{H}_i$, and $\mathbf{w}_{\gamma,c}$ is the projection parameter of class c. The attention score $\gamma_{i,l}$ evaluates the importance of the l-th token. The adaptive update of the packet representation $\mathbf{v}_i$ for class c is a weighted summation over all the positions using the attention importance scores as weights. Also, other attention mechanisms such as [5] can be utilized here.

After getting the adaptive representation vector $\mathbf{v}_{i,c}$, we can get the probability $p_{i,c}$ for each class by the linear layer and sigmoid activation function:

$$p_{i,c} = \sigma(\mathbf{W}_c\mathbf{v}_{i,c} + \mathbf{b}_c), \tag{4}$$

$$\mathbf{r}_{i,c} = p_{i,c}\mathbf{v}_{i,c}, \tag{5}$$

where $\mathbf{W}_c$ and $\mathbf{b}_c$ are the parameters of the sigmoid layer $\sigma(\cdot)$ for class c. The reconstruction representation is denoted as $\mathbf{r}_{i,c}$ for the i-th packet.

To evaluate the similarity of the packet representation from the encoder and the tuned one from classes, we include the reconstruction loss ℓ_r based on the Eq. (1) and Eq. (3) by

$$\ell_r = \sum_{i=1}^{n} \max(0, 1 - \sum_{c=1}^{C} q_{i,c}\mathbf{v}_i\mathbf{r}_{i,c}), \tag{6}$$

where $q_{i,c}$ serves as an indicator of the ground truth of class c. For any task j, if $\mathbf{y}_{i,j} = c$, i.e., the i-th packet can be classified into class c in this task, we have $q_{i,c} = 1$, and the remaining $q_{i,c}(s)$ in this task are -1; or if $\mathbf{y}_{i,j} =$ None, then all $q_{i,c}$(s) for this task are 0. Hinge loss is used to pull close the general representation $\mathbf{v}_i$ and the reconstructed representation $\mathbf{r}_{i,c}$ when $q_{i,c} = 1$. Also, it pushes them away when $q_{i,c} = -1$.

4.3.2 Tuning for Tasks. As each packet can only have one class label in each task, the parameters are learned based on the ground-truth labels. For task j with class set $\mathcal{T}_j$, we have

$$\ell_{p,j} = \sum_{i=1}^{n} \max(0, 1 - \sum_{c\in\mathcal{T}_j} q_{i,c}p_{i,c}). \tag{7}$$

Algorithm 1 PacRep Framework

```
Input: input sequence set $\mathcal{X}$, label set $\mathcal{Y}$
Output: encoding function $f$, probability function $g$
1: Initialize $\mathcal{D}_i^+$ and $\mathcal{D}_i^-$ for each packet $a_i$ based on $\mathcal{Y}$;
2: Initialize $\mathbf{H}_i$, $\mathbf{v}_i$ for each $a_i$ by Eq. (1);
3: for each iteration $t = 1, 2, ...,$ do
4:     Update parameters of $f$ and $g$ by Eq. (9);
5: end for
6: Obtain $f$ which can encode each packet $a_i$ to latent representation $\mathbf{H}_i$ and $\mathbf{v}_i$ for diverse downstream tasks
7: Obtain $g$ which can attach each $a_i$ with labels of trained tasks
```

In the testing, a packet can be classified into a class c if $p_{i,c}$ is the largest among other classes of the corresponding task.

As there are plenty of tasks, and not all packets have labels for all the S tasks, we also need to jointly tune for the tasks, thus we have the probability loss ℓ_p with $\ell_{p,j}$ in Eq. (7) by

$$\ell_p = \sum_{j=1}^{S}\sum_{i=1}^{n} w_j \max(0, 1 - \sum_{c\in\mathcal{T}_j} q_{i,c}p_{i,c}), \tag{8}$$

where the contribution of each task is regularized by w_j. Here, we define w_j as $\frac{1}{|\mathcal{T}_j|}$.

4.4 Training Objective

Based on the contrastive loss ℓ_c for the encoder in Eq. (2), the reconstruction loss in Eq. (6) and the probability loss in Eq. (8), the complete loss function of PacRep can be formulated as follows:

$$\ell = \lambda_1\ell_c + \lambda_2\ell_r + \lambda_3\ell_p, \tag{9}$$

where λ_1, λ_2 and λ_3 balance the contributions of three losses. The joint training algorithm for PacRep is summarized in Algorithm 1. In the training process, we update parameters for each module of our PacRep. In the testing process, for each packet a_i, the representations $\mathbf{H}_i$ and $\mathbf{v}_i$ can be obtained by the encoder based on updated parameters of Eq. (1), i.e., the encoding function f. The classification results $\mathbf{y}_i$ of all the tasks can be obtained by the probability $p_{i,c}$ based on the learned parameters of Eq. (4), i.e., the probability function g. If $p_{i,c}$ is the largest among other classes of the corresponding task, and the packet a_i can be classified into a class c.

Specially, among the tasks we provide, the representations will help predict labels not only for those with abnormal patterns (or classes) which have appeared in the training set (i.e., close-world problem), but also for those which have not appeared in the training set (i.e., open-world problem [28]). We will provide classification results for both close and open world in Section 5.2. Note that, inspired by the big success of pre-trained encoders like BERT in various NLP tasks, with our learned encoder, we can also improve the performance of more downstream tasks (even traffic clustering tasks, such as the protocol reverse analysis [29]), and are not limited to those we provide in this paper.

5 EXPERIMENTS

In this section, we aim to answer the following questions:

- **RQ1**: Compared to the state-of-the-art models, can the proposed PacRep achieve better classification performance for downstream tasks in the close world and the open world with the learned packet representations?
- **RQ2**: How does each part (e.g., ℓ_c, ℓ_r and ℓ_p) in PacRep contribute to the performance?
- **RQ3**: How does the format of tokens (e.g., hex, base64, plaintext) affect the performance of PacRep? and
- **RQ4**: How does the text length affect the performance?

5.1 Experimental Setup

5.1.1 Datasets. We utilize the following three publicly available datasets in our experiments.

- **ISXW2016**[1]: Each Packet in this dataset has two labels, i.e., the VPN label and the application label. For the VPN label, there are two classes (VPN and NonVPN), and all packets over VPN are encrypted. For the application label, there are 12 classes (Youtube, Facebook, et al.).
- **DoHBrw2020**[2]: This dataset is made up of traffic from DoH (DNS over HTTPS), thus all packets are encrypted. Each packet has two labels, i.e., the query abnormality label and the query generator label. For the query abnormality label, there are two classes (malicious and benign). For the query generator label, there are 5 classes (DNSCat2, Iodine, et al.).
- **USTCTFC2016**[3]: In this dataset, each packet has two labels, i.e., the software abnormality label and the software label. For the software abnormality label, there are two classes (malware and normal). For the software label, there are 19 classes (Cridex, Geodo, et al.).

Based on the labels, we have six tasks and each dataset can contribute more than one task, i.e., with ISXW2016, we have VPN detection task (*task 1*, binary classification) and application classification task (*task 2*, multi-class classification); with DoHBrw2020, we have malicious DoH query detection task (*task 3*, binary classification) and DoH query generator identification task (*task 4*, multi-class classification); and with USTCTFC2016, there are abnormality detection task (*task 5*, binary classification) and software identification task (*task 6*, multi-class classification). Although there are overlapped classes in task 2 and task 6, for simplification, we train them independently among different tasks in the experiments, which can still affect the encoder together. Also, we can record the indexes of overlapped classes, and share the same parameters to avoid redundant training.

Note that, for each packet, we delete the strong indicators like IP addresses, MAC addresses and port numbers, since some of these datasets utilize the same IP/MAC address or port number to produce some specific classes of traffic, which are easy to change in the real world. Table 1 summarizes the datasets we use in these experiments.

5.1.2 Comparison Methods. We compare our proposed framework PacRep with two categories of traffic classification methods in the six tasks, including: (1) feature-based methods (i.e., APPS, SMT, Meta-AAD, DevNet) where only features in the unencrypted packet headers (e.g., header information like protocol, packet length, flags, et al.) are considered; and (2) byte-based methods (i.e., Deep Packet, TR-IDS, HEDGE, 3D-CNN) where raw bytes are involved. Details of these compared baseline methods are as follows:

- **APPS** [21]: This method learns to select representative features, and utilize them with a random tree classifier for application classification.
- **SMT** [20]: This method fuses features of different dimensions by a kernel function and utilizes them with classifiers like SVM for application classification.
- **Meta-AAD** [27]: This method utilizes reinforcement learning to adaptively detect anomalies from different distributions of features.
- **DevNet** [18]: This method synthesizes neural network, Gaussian prior and Z-Score-based deviation loss to obtain anomaly scores based on instance features.
- **Deep Packet** [14]: This method utilizes raw data of packets to train the autoencoder and Convolution Neural Network (CNN) for application classification.
- **TR-IDS** [17]: This method treats each byte as a token in the text, and utilizes the Text-CNN to get payload features. With features from headers and payloads, a random tree classifier is trained for intrusion detection.
- **HEDGE** [4]: This threshold-based method is based on the evaluation of the randomness of the data bitstreams and can classify each packet without the need to have access to the entire stream.
- **3D-CNN** [28]: This method utilizes the byte data as input for 3D-CNN and can classify packets from both known and unknown patterns.

Table 1: Statistics of the datasets.

Dataset	ISXW2016	DoHBrw2020	USTCTFC2016
# tasks	2	2	2
# classes	14	7	21
# packets	192,741	134,195	300,222

5.1.3 Evaluation Metrics. With six tasks, the data distributions of classes are not the same. Some of them are roughly balanced while some are very imbalanced. To have a comprehensive evaluation of the performance of baselines for each task, we use the following metrics in this paper: **(1) macro F1**: It calculates the precision and recall separately for each class, and report the average F1 results of each class as macro F1; and **(2) micro F1**: It calculates the precision and recall of all classes as a whole, and obtain the micro F1 results based on this precision and recall.

Note that although it is important to evaluate the recall for some tasks with imbalanced distribution (e.g., the recall of detected abnormality in task 3 and 5), as we can obtain F1=1 for both abnormal and normal classes in these tasks, we do not show the recall results due to the page limit.

5.1.4 Implementation Details. We follow the criteria of the representation learning models [5, 13], and utilize the majority as the training set. As each dataset has a task with more than two classes, in each dataset, the training set is randomly extracted based on the labels of this task. We have 191541, 600 and 600 packets for training,

[1]https://www.unb.ca/cic/datasets/vpn.html
[2]https://www.unb.ca/cic/datasets/dohbrw-2020.html
[3]https://github.com/yungshenglu/USTC-TFC2016

Table 2: Comparison results in a close world.

Methods	Task 1		Task 2		Taks 3		Task 4		Task 5		Taks 6	
	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1
APPS	0.9349	0.9350	0.8168	0.8133	**1.0000**	**1.0000**	0.3885	0.4960	0.9532	0.9537	0.7780	0.7895
SMT	0.8929	0.8933	0.7631	0.7600	0.9747	0.9760	0.4193	0.4720	0.9532	0.9537	0.7812	0.7937
Meta-AAD	0.7067	0.7067	*	*	**1.0000**	**1.0000**	*	*	0.6622	0.6632	*	*
DevNet	0.7766	0.7767	*	*	**1.0000**	**1.0000**	*	*	0.6309	0.6316	*	*
Deep Packet	0.8550	0.8550	0.6506	0.6670	0.9747	0.9760	0.8301	0.8320	0.9768	0.9768	0.7548	0.7516
TR-IDS	0.9433	0.9433	0.5498	0.5766	0.8850	0.8960	0.8630	0.8720	0.9494	0.9411	0.5276	0.5452
HEDGE	0.8105	0.8133	0.2967	0.3400	0.9832	0.9840	0.6525	0.6480	0.8544	0.8547	0.6376	0.6421
3D-CNN	0.8946	0.8950	0.5434	0.5700	0.9747	0.9760	0.8169	0.8160	0.9958	0.9958	0.5509	0.5305
PacRep	**0.9929**	**0.9929**	**0.9151**	**0.9153**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**0.9002**	**0.9174**

validation and testing on ISXW2016. Similarly, on DoHBrw2020, the numbers are 133945, 125, 125; on USTCTFC2016, the numbers are 299272, 475, 475. We train the model with the training set, tuned hyperparameters with the validation set, and report the performance on the testing set.

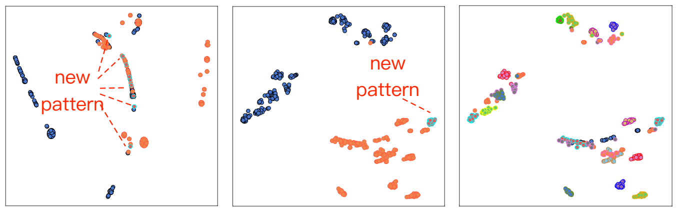
Note that, as the baselines need to learn specific data for specific tasks, six tasks are trained separately with their corresponding datasets. However, our proposed framework jointly tunes the model for these classes and tasks. We train all of the six tasks of the three datasets together, and the numbers of the training, validation and testing sets for our model are the summation of these three datasets, i.e., 624758, 1200 and 1200. To support research, we release our code at https://github.com/ict-net/PacRep.

5.2 Performance Comparison (RQ1)

We conduct comparisons on the six tasks in both the close-world and the open-world scenario.

5.2.1 Close World. For the close-world scenario, all data patterns of all tasks in the testing set have appeared in the training set. To compare with the state-of-the-art methods, we also include some new methods (i.e., Meta-AAD and DevNet) which are designed for specific tasks. However, they can not work for all six tasks, thus we will not report the results of tasks they can not work on. The results of the proposed framework PacRep along with the baseline methods are shown in Table 2.

We can make the following observations. **(1)** Compared to the state-of-the-art baseline methods, in terms of macro F1 and micro F1, our proposed PacRep outperforms all the others by a significant margin in all six tasks, which demonstrates the effectiveness of our PacRep. **(2)** Both feature-based and byte-based methods have their adept tasks. For example, most feature-based methods achieve better performance in task 3; while byte-based methods perform the feature-based ones in task 4. It demonstrates the importance of both header and payload information, considering that feature-based methods fully depend on header features but ignore payload information; while byte-based methods pay more attention to payload information and some of them even remove much header information (e.g., Deep Packet). **(3)** Compared to the byte-based methods, PacRep not only utilizes these encrypted byte data but also maintains those plaintext words, which can further enhance the performance of packet representations.



(a) Based on header features (b) Ours with binary labels (c) Ours with multiple labels

Figure 3: Scatter plots for packet representations with labels of task 1 and task 2 on ISXW2016. Different colors denote different labels (or classes). In (a) and (b), the edge color of packets from new patterns is cyan.

Table 3: Comparison results in an open world.

Methods	Task 1		Taks 3		Task 5	
	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1
APPS	0.9181	0.9183	**1.0000**	**1.0000**	0.9468	0.9474
SMT	0.8683	0.8683	**1.0000**	**1.0000**	0.9489	0.9495
Meta-AAD	0.6867	0.6867	0.6000	0.6160	0.6369	0.6379
DevNet	0.7482	0.7483	0.9312	0.9360	0.6063	0.6063
Deep Packet	0.9044	0.9050	0.9832	0.9840	0.9789	0.9789
TR-IDS	0.7901	0.7966	0.8082	0.8320	0.8222	0.8232
HEDGE	0.7989	0.8017	0.9916	0.9920	0.8459	0.8463
3D-CNN	0.9333	0.9333	0.9916	0.9920	0.9894	0.9895
PacRep	**0.9929**	**0.9929**	**1.0000**	**1.0000**	**0.9979**	**0.9979**

5.2.2 Open World. For the open-world scenario, not all data patterns (or classes) in the testing set have appeared in the training set. In this scenario, the effectiveness of packet representations is important, and it will greatly improve the performance of traffic classification. To analyze why the learned representations from the *packet encoding* module are useful, we take ISXW2016 as an example. To simulate new patterns, in the training set, we remove all "email" packets in the "VPN" class. With the learned packet encoder, we obtain packet representations of packets in the testing set, and we show the low-dimensional representations learned by t-SNE [22] in Figure 3 based on the packet representation $\mathbf{v}_i$ for each packet a_i, where Figure 3(b) and Figure 3(c) denote the same representations colored by different labels. For comparison, we also show the representation distribution based on header features in

Table 4: Ablation study on PacRep and its variants.

Methods	Task 1		Task 2		Taks 3		Task 4		Task 5		Task 6	
	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1
PacRep	0.9929	0.9929	0.9151	0.9153	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9002	0.9174
w/o ℓ_r	0.9581	0.9582	0.2101	0.2893	1.0000	1.0000	0.5794	0.6371	0.9470	0.9470	0.1298	0.2458
w/o ℓ_c	0.9929	0.9929	0.8693	0.8829	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.8497	0.8898

Table 5: Impacts of different token formats on PacRep.

Methods	Task 1		Task 2		Taks 3		Task 4		Task 5		Task 6	
	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1	macro F1	micro F1
hex	0.9765	0.9765	0.8504	0.8677	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.5876	0.6624
base64	0.9429	0.9430	0.8015	0.8124	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.5703	0.6427
word+hex	0.9929	0.9929	0.9151	0.9153	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9002	0.9174

Figure 3(a). We observe that: **(1)** In Figure 3(a), both two classes (i.e., "VPN" and "NonVPN") are mixed up, while based on our representations, there is a clear boundary in Figure 3(b). **(2)** Although packets of the new pattern are not in the training set, the representations of these packets are much closer to the right class ("VPN") in Figure 3(b). **(3)** Except for the clear boundary for two classes in Figure 3, packets of different patterns (including the new pattern) are separated, and labels for multiple patterns in Figure 3(c) further demonstrate the effectiveness of the packet encoding.

Also, we provide classification results in Table 3. Considering that these need additional strategies to decide if there are new classes [28] and it is not the focus of this paper, we will only simulate new patterns for binary classification in this section, i.e., task 1, task 3 and task 5. In detail, in the training set, we remove all "email" packets in the "VPN" class, all "dns2tcp" packets in the "Malicious" class, and all "Shifu" packets in the "Malware", which are the fewest packet patterns on the ISXW2016, DoHBrw2020 and USTCTFC2016 separately. With the same testing set without removal, we will evaluate if all packets, including those removed patterns from the training set, are accurately classified into the corresponding class. From Table 3, we can observe that **(1)** PacRep still outperforms the baseline methods, and can almost make the perfect classification in these tasks, which demonstrates the robustness of our proposed method. With the success of adapting new data patterns, we can help deal with many difficulties for traffic analysis in the real world. **(2)** Most methods decrease in the open-world scenario, which is easy to understand since some data patterns are missed from the training set but appeared in the testing set. Also, some methods receive better performance in the open world, which may be because the removal data has similarities to data patterns of the opposite class and has a bad impact on original classification results.

5.3 Ablation Study (RQ2)

To evaluate the contribution of each component to the six tasks, we conduct an ablation study. In addition to PacRep, we include the variant "w/o ℓ_r" that only includes the contrastive loss ℓ_c for the encoder and probability loss ℓ_p for class prediction, and the variant "w/o ℓ_c" that only includes the reconstruction loss ℓ_r and probability loss ℓ_p. Since the class prediction results are based on the probability loss, thus we can not remove ℓ_p like the other two objectives.

The results are presented in Table 4, and we can observe that **(1)** PacRep receives the best performance in all the six tasks, showing that each component can make positive contributions to the final results. **(2)** Compared to PacRep without the reconstruction loss (i.e., w/o ℓ_r), PacRep without the contrastive loss (i.e., w/o ℓ_c) performs better, which is because the reconstruction loss utilizes the probability of each class (can be regarded as explicit labels) to regularize the packet representation model of the encoder, while the contrastive loss only optimizes the packet representations based on the positive and negative samples (can be regarded as implicit labels). **(3)** PacRep without the contrastive loss (i.e., w/o ℓ_c) can perform as well as the PacRep in some tasks, which further demonstrates the great positive impacts of tuning for the encoder representation results based on respective classes and tasks.

5.4 Token Format Analysis (RQ3)

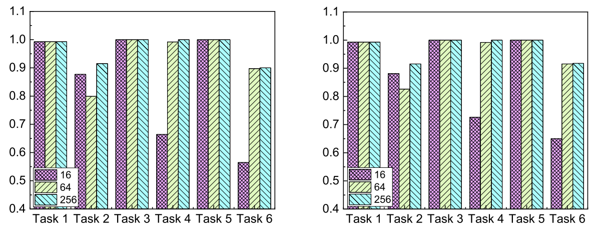
In this section, we further analyze the impacts of token formats on the classification performance in the six tasks. To take advantage of both header and payload information, most existing works utilize the "hex" (or base16) format, where they translate each byte (8 bits) in the packet to be a hex number with two characters (i.e., from \x00 to \xff). In addition to this popular format, we include the "base64" format where we translate every 12 bits into a base64 number with two characters; and the "word+hex" format where we keep those unencrypted words and only translate that encrypted information into hex numbers.

From the results in Table 5, we can find that: **(1)** PacRep with the "word+hex" format achieves the best performance in all these six tasks, which is because words can hold more semantic information and can greatly improve the learned packet representations; and **(2)** PacRep with the "base64" format achieves similar performance to that with the "hex" format, which is because although there are more tokens represented as characters in "base64" format than "hex" format, these characters can not provide extra semantical information. The vocabulary space is not enlarged for NLP-based models with only base changes.

5.5 Text Length Sensitivity (RQ4)

To evaluate the impacts of text length, we conduct experiments on different numbers of words we used (text length) varying in

$\{16, 64, 256\}$. The performance of sensitivity to the text length is presented in Figure 4. We can observe that: **(1)** Generally, longer text length performs better. Specially, for those greatly increasing with text length (task 4 and 6), at first, the performance increases with text length quickly, which is because, with longer text length, more information in the payload will be included to help improve the packet representations. After that, the increase becomes steady, which is because data at the bottom of the packet may make less contribution to the classification performance. With smaller text lengths, there is lower computing complexity. It may help to achieve high performance with relatively small text length in practice. **(2)** For task 2, PacRep with text length 16 achieves better performance to that with 64, which may because extra biased data can have bad impacts on the classification. With more data reducing the bias, PacRep with text length 256 regains the best performance.



(a) macro F1 (b) micro F1

Figure 4: Impacts of text length.

6 CONCLUSION

In this paper, we make the first investigation on the problem of learning packet representations for various downstream classification tasks. To tackle this problem, we propose the PacRep, a novel encoding-based model to learn packet representations. Both semantic and byte information is maintained, and the latent semantic space among similar packets is close via contrastive loss. Furthermore, we jointly optimize the learned representations for different classes of multiple tasks by combining reconstruction loss and probability loss. Through extensive experimental evaluations, we demonstrate the superiority of PacRep over state-of-the-art methods on various classification tasks. For further work, we focus on classification task in this paper. We would like to tune the encoder with other kinds of tasks. Second, this work aims at increasing effectiveness performance. We will work on efficiency with low complexity.

ACKNOWLEDGMENTS

This work is supported in whole or in part, by National Science Foundation of China (61902382, 61972381, 62106249, 61902376), the research program of Network Computing Innovation Research Institute (E061010003), and Project on Cyber Security and Informatization of Chinese Academy of Sciences (CAS-WX2021PY-0501).

REFERENCES

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapè. 2020. Toward effective mobile encrypted traffic classification through deep learning. *Neurocomputing* 409 (2020), 306–315.

[2] Rana Aamir Raza Ashfaq, Xi-Zhao Wang, Joshua Zhexue Huang, Haider Abbas, and Yu-Lin He. 2017. Fuzziness based semi-supervised learning approach for intrusion detection system. *Inf. Sci.* 378 (2017), 484–497.

[3] Philip Bachman, R. Devon Hjelm, and William Buchwalter. 2019. Learning Representations by Maximizing Mutual Information Across Views. In *Neurips*. 15509–15519.

[4] Fran Casino, Kim-Kwang Raymond Choo, and Constantinos Patsakis. 2019. HEDGE: Efficient Traffic Classification of Encrypted and Compressed Packets. *IEEE Trans. Inf. Forensics Secur.* 14, 11 (2019), 2916–2926.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. [n.d.]. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT 2019*. 4171–4186.

[6] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality Reduction by Learning an Invariant Mapping. In *CVPR*. 1735–1742.

[7] He Huang, Haojiang Deng, Jun Chen, Luchao Han, and Wei Wang. 2018. Automatic Multi-task Learning System for Abnormal Network Traffic Detection. *iJET* 13, 4 (2018), 4–20.

[8] Joel Hypolite, John Sonchack, Shlomo Hershkop, Nathan Dautenhahn, André DeHon, and Jonathan M. Smith. 2020. DeepMatch: practical deep packet inspection in the data plane using network processors. In *CoNEXT '20*. ACM, 336–350.

[9] Ines Jemal, Mohamed Amine Haddar, Omar Cheikhrouhou, and Adel Mahfoudhi. 2021. Performance evaluation of Convolutional Neural Network for web security. *Comput. Commun.* 175 (2021), 58–67.

[10] Bingyi Kang, Yu Li, Sa Xie, Zehuan Yuan, and Jiashi Feng. 2021. Exploring Balanced Feature Spaces for Representation Learning. In *ICLR*.

[11] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*. 1746–1751.

[12] James F. Kurose and Keith W. Ross. 2001. *Computer networking - a top-down approach featuring the internet.*

[13] Jialu Liu, Tianqi Liu, and Cong Yu. 2021. NewsEmbed: Modeling News through Pre-trained Document Representations. In *KDD*. 1076–1086.

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

[15] Xuying Meng, Suhang Wang, Zhiming Liang, Di Yao, Jihua Zhou, and Yujun Zhang. 2021. Semi-supervised anomaly detection in dynamic communication networks. *Inf. Sci.* 571 (2021), 527–542.

[16] Xuying Meng, Yequan Wang, Suhang Wang, Di Yao, and Yujun Zhang. 2021. Interactive Anomaly Detection in Dynamic Communication Networks. *IEEE/ACM Trans. Netw.* 29, 6 (2021), 2602–2615.

[17] Erxue Min, Jun Long, Qiang Liu, Jianjing Cui, and Wei Chen. 2018. TR-IDS: Anomaly-Based Intrusion Detection through Text-Convolutional Neural Network and Random Forest. *Secur. Commun. Networks* 2018 (2018), 4943509:1–4943509:9.

[18] Guansong Pang, Chunhua Shen, and Anton van den Hengel. 2019. Deep Anomaly Detection with Deviation Networks. In *KDD*. 353–362.

[19] Shahbaz Rezaei and Xin Liu. 2019. Deep Learning for Encrypted Traffic Classification: An Overview. *IEEE Commun. Mag.* 57, 5 (2019), 76–81.

[20] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, Xiaojiang Du, and Yiting Liu. 2019. Encrypted traffic classification of decentralized applications on ethereum using feature fusion. In *IWQoS*. 18:1–18:10.

[21] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2018. Robust Smartphone App Identification via Encrypted Network Traffic Analysis. *IEEE Trans. Inf. Forensics Secur.* 13, 1 (2018), 63–78.

[22] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Neurips*. 5998–6008.

[24] Tongzhou Wang and Phillip Isola. 2020. Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere. In *ICML*, Vol. 119. 9929–9939.

[25] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. 2017. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *ISI*. 43–48.

[26] Yequan Wang, Aixin Sun, Jialong Han, Ying Liu, and Xiaoyan Zhu. 2018. Sentiment Analysis by Capsules. In *WWW*. 1165–1174.

[27] Daochen Zha, Kwei-Herng Lai, Mingyang Wan, and Xia Hu. 2020. Meta-AAD: Active Anomaly Detection with Deep Reinforcement Learning. In *ICDM*. 771–780.

[28] Jielun Zhang, Fuhao Li, Feng Ye, and Hongyu Wu. 2020. Autonomous Unknown-Application Filtering and Labeling for DL-based Traffic Classifier Update. In *INFOCOM*. 397–405.

[29] Weiyao Zhang, Xuying Meng, and Yujun Zhang. 2022. Dual-track Protocol Reverse Analysis Based on Share Learning. In *INFOCOM*.

[30] Wenbo Zheng, Chao Gou, Lan Yan, and Shaocong Mo. 2020. Learning to Classify: A Flow-Based Relation Network for Encrypted Traffic Classification. In *WWW*. 13–22.

[31] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph Contrastive Learning with Adaptive Augmentation. In *WWW '21*. 2069–2080.