CapsuleFormer: A Capsule and Transformer combined model for Decentralized Application encrypted traffic classification

Xiang Zhou
x-zhou21@mails.tsinghua.edu.cn
Shenzhen International Graduate School, Tsinghua University
China

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

Qing Li
andyliqing@gmail.com
Peng Cheng Laboratory
China

Bin Zhang
bin.zhang@pcl.ac.cn
Pengcheng Laboratory
China

Guangwu Hu
hugw@sziit.edu.cn
School of Computer Science, Shenzhen Institute of Information Technology
China

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

Tianwei Zhang
tianwei.zhang@ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

Network traffic classification plays a crucial role in both network management and monitoring. Recently, an increasing number of Decentralized Applications (DApps) are appearing on various blockchain platforms. DApps employ encryption techniques such as SSL/TLS to safeguard the data transmitted over the network, making it more challenging to do traffic classification. In this paper, to tackle the challenge of insufficient classification accuracy in the existing classification of encrypted DApp traffic, we present CapsuleFormer, a novel encrypted traffic classification model for DApps. CapsuleFormer utilizes capsule neurons instead of traditional scalar neurons, where the neurons within the capsule embody various attributes of particular entities. Furthermore, Transformer blocks are adopted to generate a high-dimensional representation of the capsule activation vector. Thus, CapsuleFormer has the capability to extract potential features from the encrypted traffic patterns of DApps. Moreover, we collect and open a dataset of more than 700,000 encrypted traffic flows from 10 different types of DApps. The results of the experiments on the dataset demonstrate that CapsuleFormer is superior to the current methods, with an accuracy rate of 98.7%.

*Corresponding author.

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07…$15.00
https://doi.org/10.1145/3634737.3637664

CCS CONCEPTS

• **Security and privacy → Network security**.

KEYWORDS

Blockchain, Decentralized Applications (DApps), Traffic Classification

ACM Reference Format:
Xiang Zhou, Xi Xiao, Qing Li, Bin Zhang, Guangwu Hu, Xiapu Luo, and Tianwei Zhang. 2024. CapsuleFormer: A Capsule and Transformer combined model for Decentralized Application encrypted traffic classification. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore*, 12 pages. https://doi.org/10.1145/3634737.3637664

1 INTRODUCTION

Network traffic classification plays a crucial role in both network management and monitoring. The blockchain network adopts a decentralized and distributed ledger technology. Its inherent features, namely decentralization, transparency, and immutability, enable its wide-ranging applications in digital currencies, copyright management, digital identity verification, and other areas. The majority of these applications are developed on the basis of decentralized applications, called Decentralized Applications (DApps).

DApps are a type of software that runs on a decentralized blockchain network. Until November 2022, there had been over 5,800 DApps deployed across various blockchain platforms, including Ethereum (28.07%), Binance Coin(BNB) chain (18.39%), and Polygon (7.64%) [1]. These applications control business logic and deal with data via smart contracts. Thanks to these contracts, DApps work in a predictable manner without relying on trust in a central organization. Unlike traditional centralized applications, DApps are not controlled by a single entity. DApps employ encryption techniques

such as SSL/TLS to safeguard the data transmitted over the network, making it more challenging to do traffic classification.

Traditional methods for network traffic classification, such as using payload-based deep packet inspection (DPI) [35], are not effective in the case of DApps deployed on blockchains as they utilize SSL/TLS encryption to secure data. While there has been some research on traffic classification for general applications [14, 23, 24, 29], few studies have focused specifically on DApps. Among these studies, Wang et al. [32–34] only considered packet length and time-related features, neglecting other potentially valuable information in byte features. Shen et al. tried to address this issue in [25, 26], but the results were not satisfactory. In [25], a graph neural network approach was proposed, but the information for model training was limited due to using only the first 25 packets in a flow.

Capsule networks outperform traditional neural networks in terms of feature extraction ability [11, 22]. Especially, unlike scalar-based neural units, each component of the vector neuron in capsule networks can capture the properties of traffic features, such as the position of byte information and the direction between packets. Moreover, capsule networks do not rely on pooling operations to address the issue of information loss, thereby preserving more fine-grained information.

Inspired by the above idea, we present the CapsuleFormer model based on capsule networks to do traffic classification of DApps. In order to get the input of the model, a novel feature selection algorithm is proposed to evaluate the significance of each feature under diverse sample conditions. The architecture of CapsuleFormer consists of a PrimaryCapsule layer, a Transformer block, and a DigitalCapsule layer. The output of each capsule is a vector, with coordinate values indicating position, size, orientation, and other factors. We incorporate the self-attention mechanism in Transformer to enhance the feature representation, which calculates attention weights between features to capture the contextual information of global features. The DigitalCapsule layer finally obtains the embedding vector for classification. We thoroughly evaluate the performance of CapsuleFormer on real-world datasets. The main contributions of this paper are as follows:

- Based on literature [14], we put forward an enhanced feature selection algorithm based on the Laplacian method. The improved algorithm calculates the score of each feature utilizing an indefinite number of samples and its efficacy is confirmed through experiments.
- We build a deep learning model, CapsuleFormer, to classify DApp encrypted traffic. CapsuleFormer leverages both capsule network and Transformer to extract latent features from the encrypted traffic. This process empowers CapsuleFormer to learn and comprehend the underlying characteristics of the traffic. To the best of our knowledge, this approach is novel and has not been explored before.
- We collect and open[1] a real-world traffic dataset from 10 DApps on the Ethereum and BNB chains, consisting of approximately 200G of data and over 700,000 flows. The experiments on this dataset demonstrate that the proposed method outperforms state-of-the-art approaches, achieving a classification accuracy of 98.7%.

[1]https://github.com/Jsontorch/CapsuleFormer

2 THREAT MODEL AND RELATED WORK

2.1 THREAT MODEL

The DApp classification is a form of traffic attack method aimed at identifying the DApps users access. For this attack to succeed, attackers need to acquire the traffic generated when users access DApps. Therefore, we assume that DApps and the blockchain network will not voluntarily leak traffic information (or other sensitive metadata) to attackers. Attackers are unable to execute any attacks other than traffic collection within the user's local network. This is detailed in Fig. 1.

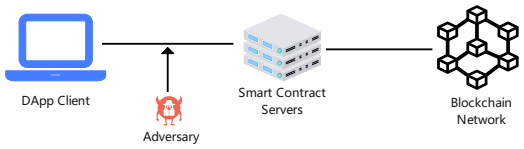

Figure 1: The system model of DApp classification.

2.2 RELATED WORK

As cryptographic protocols have advanced, traditional techniques such as port-based and payload-based methods have become obsolete in this field. Therefore, we will not be discussing traffic classification using these traditional methods in our survey. Instead, we will focus on the following areas: Mobile application traffic classification, Web application traffic classification, and DApps application traffic classification.

2.2.1 Mobile Application Traffic Classification. This category can be broadly divided into two main types. The first aims to directly classify mobile application types, while the second aims to differentiate user actions within mobile applications.

1) In the first category, Tang et al.[28] introduced Trigger Relationship Aware Traffic Classification (TRAC), which utilizes a trigger relationship graph model to depict the relationships between applications. A trigger relationship analyzer is then used to construct the graph, and the traffic is classified using the application labels identified by the DPI engine. To overcome the "forgetting problem" faced by current traffic classification classifiers, Chen et al.[4] proposed an incremental learning framework that combines the One vs Rest (OvR) strategy with neural network classifiers. This framework allows for the incremental addition of new applications to the classification system while preserving the knowledge acquired by existing classifiers. Furthermore, certain traffic classification methods incorporate conventional machine learning techniques, such as random forest [29] and logistic regression [7], recent research has predominantly leveraged deep learning approaches such as Recurrent Neural Networks (RNNs) [17], Capsule Networks [6], and Graph Neural Networks (GNNs) [2, 12] for traffic classification.

2) In the second category, Coull et al.[5] proposed a method that utilized the size of encrypted packets to infer user actions,

the language of messages, and even the length of these messages with an accuracy of over 96%. This approach is particularly effective for instant messaging services. Yan et al.[36] used a two-step approach for classifying user actions within mobile applications. Firstly, the original data is partitioned into "bursts" to represent different actions. Next, features such as packet length and inbound and outbound statistics are extracted. Finally, a random forest classifier is used for classification. Liu et al.[18] presented a prompt approach for recognizing application activity based on encrypted traffic. The authors extracted trend features from the traffic generated by application activities and subsequently employed a random forest algorithm for the classification process. Li et al.[15] proposed a multi-label dataset of Internet traffic (MLDIT) and constructed four behavior classifiers using Multi-layer Perceptron (MLP).

2.2.2 Web Application Traffic Classification. Cai et al. [3] employed Support Vector Machines (SVMs) with kernel functions based on the Damerau-Levenshtein edit distance and Hidden Markov models as classifiers in the Webpage and Website Fingerprinting (WF) attacks, respectively. Panchenko et al. [20] introduced a WF attack on the Tor network, named CUMUL, which utilized the cumulative sum of packet lengths as a feature and classified it using an SVM classifier. Hayes et al.[9] introduced a random decision forest-based fingerprinting technique for websites called k-fingerprinting. This method achieved accuracy comparable to that obtained by CUMUL. Sirinam et al. [27] proposed Deep Fingerprinting (DF), a WF attack against the Tor network. DF utilizes Convolutional Neural Networks (CNNs) for feature extraction and classification. Wang et al.[31] introduced a cross-platform WF attack method based on multiple similarity loss, utilizing the K-nearest neighbor algorithm with cosine similarity as the primary classification model. Lu et al.[19] proposed a Graph Attention Pooling network (GAP-WF) for precise WF. Khajehpour et al.[13] employed encrypted payloads of network packets to fingerprint Tor traffic and utilized MLP, CNN, and Random Forest (RF) as classifiers. Ling et al.[16] introduced a genetic programming-based variant covert traffic search technique to identify misleading coverage traffic for deep learning-based WF classifiers (CNN, DF, and Stacked Denoising Autoencoders (SDAE)[21]). The results showed that in the open world, the detection rate was 0.4% with a bandwidth overhead of only 8.1% compared to the DF model. In the closed world, the detection rate was only 1.7% and the bandwidth overhead was 12.0% compared to the DF model. Gong et al. [8] introduced a defense technique against WF named Suraka. It utilizes Generative Adversarial Networks (GANs) to generate transmission patterns and adjusts the buffered data accordingly.

2.2.3 DApps application traffic classification. Wang et al. explored the issue of classifying encrypted traffic for DApps in three studies [32–34]. In [32], the authors employed statistical features (e.g. packet length, time series) to build a dataset and used Gradient Boosting Decision Tree (GBDT), Decision Tree (DT), and RF as classifiers. [34] proposed a quadratic network based on clustering, reducing the training set's redundancy and using a quadratic structure to capture more constrained relationships. [33] utilized time- and sequence-based features and employed an RF classifier. Shen et al. also studied DApps classification in [25, 26]. In [26], the authors fused different dimensional features through kernel functions and used KNN, SVM, and RF as classifiers. [25] proposed the Traffic Interaction Graph (TIG) as an info-rich representation of encrypted DApp flows, transforming the DApps fingerprinting into a graph classification problem via bidirectional client-server interactions. In [37], Yang et al. perform flow classification. The authors utilized time series and packet length sequences as primary raw features. Furthermore, the flow data was segmented into bursts at specific time intervals. Subsequently, these time series and packet length sequences were used to create feature matrices for each burst. Finally, CNN and BiGRU were harnessed as classifiers for the classification task.

2.3 Summary

The limitations of conventional techniques are predominantly evident in two facets. Firstly, there is inadequate utilization of the raw data(Pcap files) from DApps, as models are constructed using only packet length sequences or time series-related features (e.g., [7, 17, 25, 29]) as inputs. Secondly, DApps deployed on the same blockchain type commonly share similar SSL/TLS protocols, run backend code on the identical blockchain network, and manage relevant data, potentially causing distinct DApps to exhibit analogous characteristics. This feature of DApps could lead to a decline in the classification accuracy of existing methods. Furthermore, as noted in [25], the author's utilization of DApp data is suboptimal, confined to utilizing merely the initial 25 packets from the flow to construct the graph. Compounding this issue, the graph construction process itself is quite time-consuming. Additionally, the approach presented in [34] demonstrates potential by suggesting the division of the dataset into easy and hard datasets to potentially augment training efficiency. However, due to the intricate nature of the workflow stemming from the author's choice not to employ an end-to-end training methodology for model creation, the anticipated substantial reduction in overall processing time remains unrealized.

3 METHODOLOGY

In this section, we present our novel method, CapsuleFormer. The overall architecture of the model is depicted in Fig. 2, composed of four distinct components: Data Preprocessing, Feature Generation, Model Training, and DApps Classification.

3.1 Data Preprocessing

The data preprocessing in this subsection involves several steps to clean and prepare the raw traffic data for constructing a dataset of features. These include packet processing, segmenting the data into flows using packet capture (Pcap), and further filtering of the flows.

1) Packet processing: To ensure the purity and integrity of the raw traffic data, invalid packets must be eliminated. This includes identifying and removing retransmitted packets, packets with multiplexed IPs and ports, acknowledged packets, packets with missing TCP answers, and packets with non-consecutive sequence numbers. This step helps to filter out any irrelevant or inaccurate information in the dataset.

2) Pcap-Flows segmentation: The original Pcap data can be segmented into a sequence of flows. Each flow is then defined as a series of packets that share the same 5-tuple, which includes the

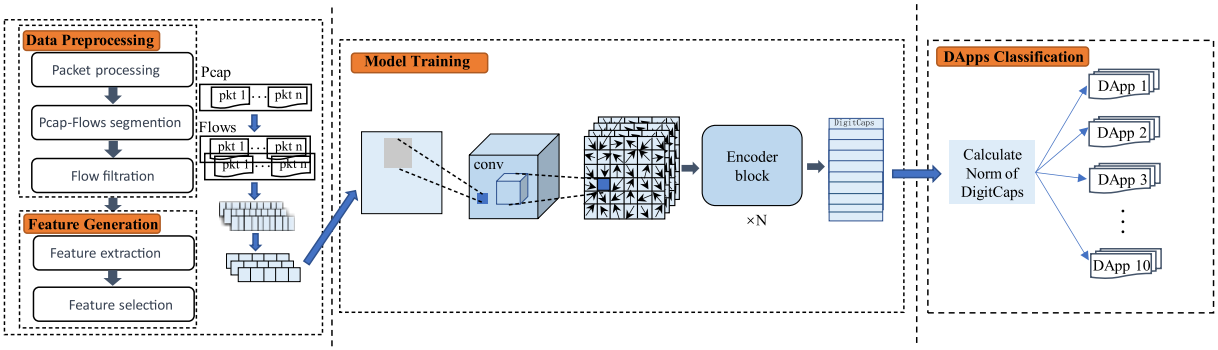

Figure 2: The architecture overview of CapsuleFormer.

source/destination IP address, source/destination port, and protocol.

3) Flow filtration: After pcap-flows segmentation, some of the smaller flows, which have a smaller size, contain less information related to the category. To ensure data integrity, we retain the maximum flow after segmenting the traffic, where the maximum flow refers to the one containing the highest number of packets.

3.2 Feature Generation

1) Feature extraction: A detailed description of all features used in this analysis is provided below:

- General flow-level descriptive statistical features: For a flow, various statistical measures are calculated including mean, maximum, minimum, standard deviation, variance, median, skewness, kurtosis, and a ratio of the standard deviation to the square root of the flow length with respect to its time series and packet length, respectively.
- Port number: Including source and destination ports.
- The rate of packets and bytes: Number of packets per second in the flow, number of bytes per second in the flow.
- Time and protocol: The average arrival time of packets in the flow, timestamp of the first outbound packet and the first inbound packet in the flow, timestamps of the first 100 packets in the flow, duration of the flow, and the used transport protocol.
- Statistical features of bidirectional packets in the flow: Including the proportion of bidirectional packets, proportion of bidirectional bytes, total number of bidirectional packets, total number of bidirectional bytes, and average packet length.
- Packet length distribution: Binning into 150 bins, resulting in a list of length 150.
- Binning statistics features: Dividing the flow's packet sequence into groups of 20 and tallying the number of outgoing packets in each group to create a sequence of outgoing group totals. Calculating the standard deviation, mean, median, maximum, and minimum of this sequence.
- Inbound/outbound packets: Obtaining the number of outbound and inbound packets in the first 30 and last 30 packets of a flow.
- Payload features: Calculating the mean and variance of the payload in a sequence of packets in a flow. Distribution of bytes of payload in the flow.
- Percentage of elements of a flow: Dividing the packet length and delay sequence into 10 equal parts and calculating the percentage of elements in each bin over the total elements as a feature. This feature is a list of lengths 10.
- Length of the longest monotonically increasing sub-sequence in the flow: Including both increasing and decreasing, calculated separately for time and packet length sequences.
- Variance of the forward and backward direction of the flow: Given a sequence, first, three representative positions (i.e., the first, second, and third quartiles) are selected from the sequence. Then, for each selected position, the sequence is divided into two sub-sequences in the forward and backward directions. Finally, the variance of these sub-sequences is calculated as a feature.
- Jump count in the flow: Calculating the number of elements in a sequence whose value is significantly greater than the mean difference of the adjacent elements in the sequence.

2) Feature selection: After the calculations in Feature extraction, the total dimension of the features is 599. In order to prevent overfitting, we propose a feature selection algorithm based on the Laplacian algorithm [10]. In the Laplacian algorithm, all samples are used to construct the similarity matrix. Intuitively, it is understood that the number of samples used to construct the similarity matrix affects the experimental performance and the overall experimental duration. Therefore, considering this aspect, we are reworking the original algorithm to facilitate the determination of the optimal number of samples to be used when constructing the similarity matrix. The algorithm's specific steps are described in Algorithm 1. It takes the results obtained from feature extraction and the number of samples used to construct the similarity matrix as inputs. Since we're utilizing supervised learning and have known target labels, we don't need to compute the Euclidean distances between samples in all categories, thus reducing the time cost.

Initially, for all samples within the same category, we calculate the Euclidean distances among them and store these values in a list E (lines 6-7). For samples not within the same category, we set the corresponding elements in the similarity matrix to 0 (line 9). Following this, we sort the list E in ascending order (line 13), and

select the top a elements into list G, aiming to obtain the nearest a samples to a specific sample. Then, for the nearest a samples within the same category based on Euclidean distance, we set the corresponding elements in the similarity matrix to 1 (line 17) and set the remaining elements to 0 (line 19) in the similarity matrix. At this point, we obtain the similarity matrix constructed based on the entire dataset. Subsequently, we obtain the degree matrix D and Laplacian matrix L from the similarity matrix (line 23). Finally, we utilize $\mathbf{f}_r$, D, L to compute the score for each feature (lines 24-30).

Furthermore, we employ the improved algorithm to calculate the score of each feature using an indeterminate number of samples. This approach demonstrates highly effective for feature extraction in DApps, as illustrated by the results obtained in the experimental section.

Algorithm 1 Feature Selection Algorithm

Input: $X \in R^{n\times m}$, n: Number of flows, m:Number of features, a:Number of flows in constructing matrix S.
Output: L_r: The Score of the r-th feature.

1: f_{ri} denotes the i-th flow of the r-th feature, $i = 1, \cdots, m$. $\mathbf{x}_i$ denotes the i-th flow.
2: Generate the similarity matrix S.
3: **for** $\mathbf{x}_i \in X$ **do**
4: **for** $\mathbf{x}_j \in X$ **do**
5: **if** $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same category. **then**
6: $d\left(\mathbf{x}_i, \mathbf{x}_j\right) = \left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2$
7: put $d\left(\mathbf{x}_i, \mathbf{x}_j\right)$ into a list E.
8: **else**
9: $S_{ij} = 0$
10: **end if**
11: **end for**
12: **end for**
13: Sort E in ascending order. Put the first a elements of E into list G.
14: **for** $\mathbf{x}_i \in X$ **do**
15: **for** $\mathbf{x}_j \in X$ **do**
16: **if** $d\left(\mathbf{x}_i, \mathbf{x}_j\right) \in G$ **then**
17: $S_{ij} = 1$
18: **else**
19: $S_{ij} = 0$
20: **end if**
21: **end for**
22: **end for**
23: Define $\mathbf{f}_r = [f_{r1}, f_{r2}, \cdots, f_{rm}]^T, D = \text{diag}(S\mathbf{1}), \mathbf{1} = [1, \cdots, 1]^T, L = D - S$
24: **for** $1 \le r \le m$ **do**
25: $\widetilde{\mathbf{f}}_r = \mathbf{f}_r - \frac{\mathbf{f}_r^T D\mathbf{1}}{\mathbf{1}^T D\mathbf{1}}\mathbf{1}$
26: $\overline{L} = \widetilde{\mathbf{f}}_r^T L \widetilde{\mathbf{f}}_r$
27: $\overline{D} = \widetilde{\mathbf{f}}_r^T D \widetilde{\mathbf{f}}_r$
28: $L_r = \frac{\overline{L}}{\overline{D}}$
29: **end for**
30: **return** L_r

3.3 The CapsuleFormer Classifier

In this subsection, we propose CapsuleFormer classifier, as shown in Fig. 3, which can effectively classify encrypted traffic in DApps. The size of the input matrix to the CapsuleFormer is 22×22. The CapsuleFormer consists of three parts: a convolutional layer, a Transformer block, and a capsule layer. In a capsule network, a capsule is comprised of a set of neurons with specific parameters. The activation vector of a capsule neuron carries information about multiple attributes, such as position, size, and direction, through its coordinate values. The length of the vector represents the probability of the feature's existence [22].

1) Convolutional layer: The input data dimension to the convolutional layer is $1 \times 22 \times 22$. To enhance the model's learning capabilities, we normalize the input data so that the values are between 0 and 1. Initially, the convolutional kernel size is 3×3, with a stride of 1 and 256 kernels. After convolution, we apply the ReLU activation function to increase the model's non-linearity.

After the first convolutional layer, we obtain a feature map of size $256 \times 20 \times 20$. Then, it goes through the second convolutional layer, containing 256 kernels with the kernel size of 9×9 and a stride of 2. The output dimension is $256 \times 6 \times 6$. Next, we crop the channel dimension to 32 eight-dimensional vectors, where 8 is the length of the capsule neurons. The second convolutional layer with the cropping operation is also called the primary capsule layer. At this point, the primary capsule layer has 1152 capsules with a dimension of 8.

2) Transformer block [30]: The third layer is a Transformer block, whose main purpose is to enhance the feature representation by mapping the vector output from the primary capsule layer to a higher dimensional space. To achieve this, the output from the previous layer is first processed through a fully connected pre-net, resulting in a tensor with a dimension of 1152×10. Then this tensor is input into the Transformer where it calculates the Q, K, V values for the 1152 capsules according to the formula, and then calculates the attention scores for each head according to equation (1)-(2).

$$Q = XW^Q, K = XW^K, V = XW^V \tag{1}$$

where $W^Q, W^K, W^V \in \mathbb{R}^{x_l \times d}$ are trainable parameters, x_l is a dimension of the primary capsule, d is the hidden layer dimension. Subsequently, the outcome of this self-attention head is calculated:

$$head = Atten(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V \tag{2}$$

It should be noted that the trainable parameters W^Q, K^Q, V^Q are different for each head, in order to calculate attention scores from multiple perspectives. Finally, the outputs of multiple headers are concatenated together, and the formula used for this is defined as follows:

$$MultiHead = Concat(head_1, ..., head_h)W^O \tag{3}$$

where $W^O \in \mathbb{R}^{hd \times d_{model}}$. h represents the number of headers, while d_{model} denotes the dimension of the final output.

The final output tensor has a dimension of 1152×10.

3) DigitCaps: The Transformer block produces 1152 capsules, each with a vector length of 10. Initially, each vector u_i passes through a parameter matrix W_{ij} to obtain prediction vectors $\hat{u}_{j|i}$.

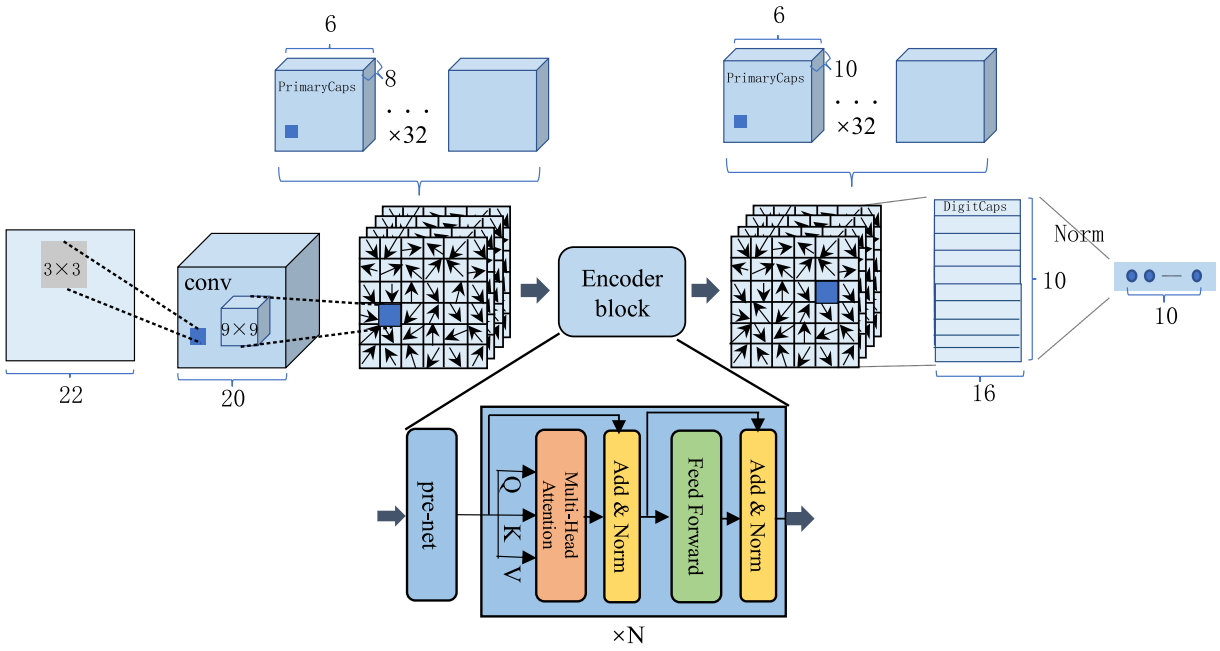

Figure 3: The schematic illustration of CapsuleFormer. An arrow represents the output vector of a capsule.

Then, each prediction vector is multiplied by a coefficient c_{ij} and summed for i to obtain the capsule s_j. The formulas for calculating the prediction vectors and the input vector s_j of the capsule are as follows:

$$s_j = \sum_i c_{ij}\hat{u}_{j|i}, \quad \hat{u}_{j|i} = W_{ij}u_i \tag{4}$$

where c_{ij} is the coupling coefficient, which is calculated by equation (5).

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \tag{5}$$

where b_{ij} is updated through Algorithm 2.

Algorithm 2 Dynamic Routing Algorithm

procedure ROUTING $(\hat{u}_{j|i}, r, l)$
for all capsule i in layer l and capsule j in $(l+1)$: $b_{ij} \leftarrow 0$
for r iterations **do**
  all capsule i in layer l : $c_i \leftarrow \text{softmax}(b_i)$ ▷ Softmax computes c_{ij}
  all capsule j in layer $(l+1)$: $s_j \leftarrow \sum_i c_{ij}\hat{u}_{j|i}$
  all capsule j in layer $(l+1)$: $v_j \leftarrow \text{squash}(s_j)$ ▷ squash computes v_j
  all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$
end for
return v_j

To utilize the length of the capsule output vector as an indication of the presence probability of a category, the input vector s_j of the capsule is compressed to a range between 0 and 1 through a non-linear squashing function. The definition of the squashing function is presented below:

$$v_j = \frac{\|s_j\|^2}{1+\|s_j\|^2}\frac{s_j}{\|s_j\|} \tag{6}$$

where v_j is the output vector of the capsule.

The L2 norm representation of digit capsules is indicative of the probability of entity existence. In order to minimize the loss corresponding to positive samples and maximize the loss of negative samples, the margin loss is employed as the loss function for a single digit capsule. Mathematically, the margin loss is defined as follows:

$$L_k = T_k \max(0, \hat{x} - \|\mathbf{v}_k\|)^2 + \lambda(1-T_k)\max(0, \|\mathbf{v}_k\| - \overline{x})^2 \tag{7}$$

where k represents the predicted class, T_k equals 1 when the class k exists, otherwise, it equals 0. The upper bound $\hat{x}$ and lower bound $\overline{x}$ are respectively set to 0.9 and 0.1. This setting aims to avoid wasting time in further strengthening the L2 norm of the digit capsules when the representation of positive and negative samples is good enough, as reflected in the L2 norm of their corresponding digit capsules being large or small enough. Further training is mainly focused on more difficult-to-classify samples. The parameter λ is set to 0.5 to balance the weights of the two boundaries, thus avoiding the L2 norm of digit capsules being too small at the beginning of training. The total loss for the network is the sum of the losses for all individual capsules. The parameters of each layer and the input and output dimensions in the CapsuleFormer model are shown in Table 1.

Finally, we will use the trained CapsuleFormer model to predict unknown encrypted DApp traffic, and evaluate its performance. And we utilize the test results to guide the selection of hyperparameters.

Table 1: Parameters of each layer in the CapsuleFormer model

Layer	Kernel size	Stride	Number	Input	Output
Conv1	3	1	256	$1 \times 22 \times 22$	$256 \times 20 \times 20$
PrimaryCaps	9	2	1	$256 \times 20 \times 20$	$1152 \times 8 \times 1$
Encoder block	-	-	5	$1152 \times 8 \times 1$	$1152 \times 10 \times 1$
DigitCaps	-	-	1	$1152 \times 10 \times 1$	$10 \times 16 \times 1$
Norm	-	-	1	$10 \times 16 \times 1$	$10 \times 1 \times 1$

4 DATASETS AND METRICS

4.1 Dataset

The initial step involved gathering DApps traffic data from the network environment. The dataset is obtained by selecting DApps based on their user volume ranking on a specific website, and data is being collected in April 2022 for approximately two weeks. A total of 10 categories were collected, five DApps each on the Ethereum and BNB chains. The process involved accessing specific DApps through a Chrome browser on a computer, navigating randomly within the DApps, and then using tcpdump to capture the encrypted traffic data. Each DApp category was visited 1000 times, and the resulting raw data were processed by the data processing method outlined in Section III. The final dataset details can be found in Table 2.

Table 2: Details of Encrypted Traffic Data for DApps

DApps	Flow	Size	Label
sushi	10651	15.6G	1
uniswap	95658	29.8G	2
solo.top	29007	2.97G	3
venus	50060	12.7G	4
pancakeswap	76814	15.6G	5
biswap	78707	21.4G	6
alpacafinance	70297	17.4G	7
dodoex	73539	8.29G	8
tokenlon	78566	13.0G	9
curve	67099	64.2G	10

4.2 Performance Metrics

To validate the experimental results, we utilize a set of standard evaluation metrics. Additionally, we conducted a 10-fold cross-validation on the dataset. This procedure involves applying stratified sampling to randomly extract samples from each class, maintaining a 4:1 ratio for the creation of both training and testing sets. Ultimately, the average results are used as the final classification results. Before delving into the evaluation metrics, we will first define several parameters that are necessary for the evaluation metrics. For each category, we have True Positive (TP), which is the number of instances where the sample's true class is positive, and the model correctly identifies it as such; False Negative (FN), which is the number of instances where the sample's true class is positive but the model incorrectly identifies it as negative; False Positive (FP), which is the number of instances where the sample's true class is negative but the model incorrectly identifies it as positive; and True Negative (TN), which is the number of instances where the sample's true class is negative, and the model correctly identifies it. By using these four parameters, we can compute four main evaluation metrics for the classification results: Accuracy, Precision, Recall, and F1 Score.

Accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{8}$$

Precision is defined as:

$$Presion = \frac{TP}{TP + FP} \tag{9}$$

Recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{10}$$

And F1 score is defined as:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{11}$$

4.3 Comparison of Models

To validate the effectiveness of our model, we compare our model with several other methods, as outlined below:

- Appscanner [29]: A random forest classifier is employed to classify outbound, inbound, and bidirectional flows using statistical features of packet length such as minimum and maximum values.
- EDC [15]: This method also employs simple features such as packet length, port number, and packet direction, and utilizes a simple MLP with only two hidden layers as the classifier.
- FAAR [18]: A random forest classifier is employed to classify the data, after extracting features using the packet length and wavelet decomposition.
- GDA [25]: This method constructs a graph for each flow using the first 25 packets of the flow, where the packets are divided into bursts. An edge is then established between packets within the same burst, and an edge is also established between the beginning of the last burst and the last packet. Each packet also has a direction, and the graph is then classified.
- CBFM [37]: This method first divides flows into bursts within specific time intervals. Following this, time series and packet length sequences within each burst are harnessed to construct burst feature matrices. Finally, CNN and BiGRU are employed as classifiers to classify the burst feature matrices derived from the burst sequence of each flow.
- FS-Net [17]: This method utilizes the packet length as an initial feature and then obtains an Embedding representation of the packet length. The Embedding representation is then fed into a model, which is composed of an autoencoder and a Bi-directional Gated Recurrent Unit (Bi-GRU) for classification.
- SPCaps [6]: This method employs the bytes of each packet sequence payload as features. The process begins by extracting the first 784 bytes of each packet sequence, reshaping

them into a 1x28x28 tensor, and then feeding them into a basic capsule network classifier for classification.

- CQNet [34]: The method first extracts byte sequences from pcap files to form the raw dataset. These sequences are then input into a CNN model, previously trained on Fashion-MNIST, to generate embedding features. Following this, the obtained embedding features undergo clustering techniques, which partition the overall dataset into easy dataset and hard datastet. Ultimately, the researchers utilize a quadruplet network designed using a Siamese architecture to conduct training specifically on the hard datastet.

4.4 Experimental Setup

The proposed approach, which is implemented using Python 3.7.13 and PyTorch 1.12.1, is trained on an Ubuntu operating system, utilizing an Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz with 10 CPU cores, 128 GB of RAM, as well as an NVIDIA GeForce RTX 3080 GPU for acceleration.

5 EXPERIMENTAL RESULTS

In this section, we first validate the effectiveness of Algorithm 1. Since variations in parameter a can significantly impact the model's accuracy, which holds crucial implications for real-world applications, we also conduct experiments on parameter a. Moreover, as the input feature dimensions can affect the model's performance and have implications for the actual deployment of the model, we also experiment with input dimensions. Subsequently, to understand the superiority of our method and provide reference benchmarks for comparison, we conduct comparisons with other methods to evaluate the performance of our model. Furthermore, to analyze the importance and effects of each module, we perform ablation experiments on each module to explore their contributions to the overall performance. Finally, to evaluate the model's scalability and its practical application within computational constraints, we study the time complexity of the proposed model in the system. To provide insight into the specifics of the experiments, we present the hyperparameters of the model in Table 3. Our Research Questions (RQ) are as follows:

- **RQ1:** How well does Algorithm 1 perform? And how does the parameter a affect the experimental results?
- **RQ2:** How does the input feature dimension affect overall performance?
- **RQ3:** How does the performance of our model compare to that of other models?
- **RQ4:** What is the contribution of each module in the model to the overall performance?
- **RQ5:** What is the time complexity of the proposed model?

5.1 Examination of Feature Selection (RQ1)

1) In order to demonstrate the benefits of feature selection, we performed experiments in the feature generation step of the CapsuleFormer framework by removing feature selection while keeping all other conditions constant. The resulting confusion matrix is presented in Fig. 4 and Fig. 5. As depicted in Fig. 4 and Fig. 5, the classification performance of classes 0, 1, 4, and 6 was noticeably improved after applying feature selection, thereby demonstrating that the incorporation of feature selection can enhance the feature representation ability of the original model.

2) To demonstrate the influence of the number of samples utilized in constructing the similarity matrix on the effectiveness of the feature selection algorithm. In Algorithm 1, we experimented with adjusting the value of parameter a from 1 to 5, 10, 15, and so on.

Table 3: Hyperparameters of the CapsuleFormer

Hyperparameters	#
Batch Size	128
Optimizer	Adam
Learning Rate	0.001
lr_scheduler	ExponentialLR
lr_decay	0.9
Number of epochs	50
Activation Functions	Relu
Number of MLP layers	1

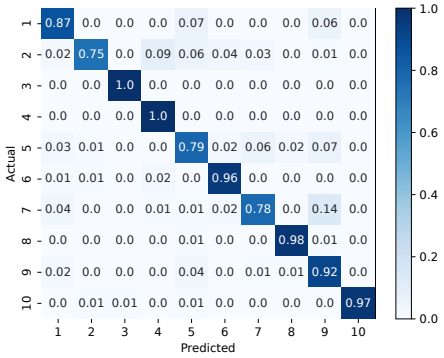

Figure 4: confusion matrices for CapsuleFormer performance without(w/o) feature selection (FS). (Refer to Table 2 for DApp categories)

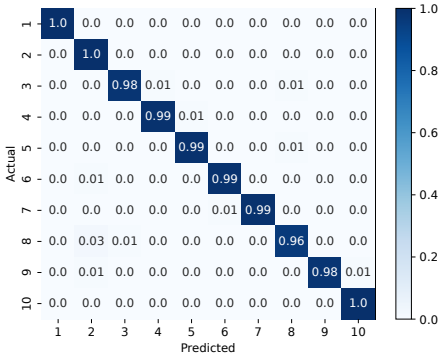

Figure 5: confusion matrices for CapsuleFormer performance with feature selection (FS). (Refer to Table 2 for DApp categories)

As depicted in Fig. 6, it is evident that with the increment of a from 0 to 405, the accuracy of the model exhibits an ascending trend. Nonetheless, there is a minor oscillation present; however, the overall trajectory demonstrates a consistent enhancement. The pinnacle accuracy is attained at a value of 405. However, upon further increase in the value of a, the model's accuracy initiates an oscillation that leads to a decline.

To sum up, the optimal performance on the test set is achieved when building the similarity matrix S using 405 samples. Therefore, 405 samples will be the basis for constructing the similarity matrix S in this experiment.

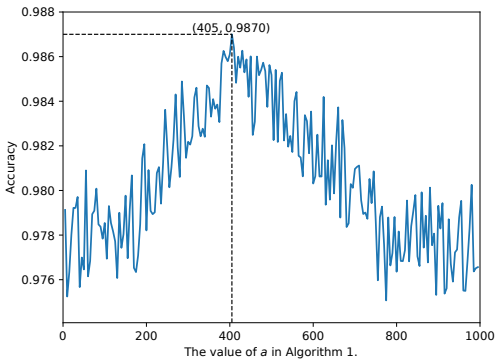

Figure 6: Impact of the value of a in Algorithm 1 on Model Accuracy in Constructing the Similarity Matrix

5.2 Input dimensions of the CapsuleFormer model (RQ2)

Fig. 7 shows the test accuracy of CapsuleFormer across different input dimensions. The graph reveals a clear trend: as input dimensions expand from 10×10 to 22×22, CapsuleFormer's test accuracy consistently improves. The zenith of accuracy is attained at 22×22 input dimensions. Nevertheless, when dimensions surpass 22×22, CapsuleFormer's test accuracy starts diminishing. Concurrently, at 10×10 input dimensions, the test accuracy of CapsuleFormer drops to its lowest point of 0.5127. This can be attributed to the limited utilization of only 100 features from the 599-dimensional input, resulting in insufficient data for robust model training and diminished generalization capabilities. Additionally, as input dimensions surpass 22×22, the model succumbs to overfitting, which consequently leads to a decline in test accuracy.

From the experimental results, it can be observed that the model performs optimally when the input dimensions are 22×22. Both too small and too large values exhibit significant performance degradation. In summary, this study adopts the input dimensions of 22×22 as the selected features for the final CapsuleFormer model.

5.3 Comparison Experiments (RQ3)

Table 4 presents the experimental results of our proposed model and its comparison with other models on the DApps dataset. All the methods utilized for comparison are applicable to the classification

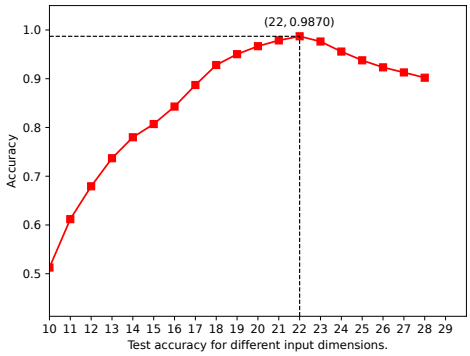

Figure 7: Impact of input dimensions on Model Accuracy

of encrypted traffic. Based on the data presented in the Table 4, the following conclusions can be drawn:

1) Based on Table 4, it is evident that CapsuleFormer demonstrates a favorable performance scale regarding Precision and Recall for the majority of DApp categories, with only minimal deficiencies in a few categories. Additionally, when examining all categories, CapsuleFormer's F1 score and Accuracy surpass those of other models. This analysis indicates that our method exhibits excellent robustness.

2) For Appscanner, FS-Net, EDC, and FAAR methods, they are mainly designed for traditional application traffic classification and behavior classification. As a result, these methods use fewer original features, usually only packet length or time series, which are less informative for classifying DApp encrypted traffic. Additionally, these methods exhibit suboptimal performance because the special characteristics of DApp encrypted traffic were analyzed in introduction and related work.

3) The method of GDA, which uses graph neural networks to classify DApps. However, the GDA method only employs a few initial packets of a flow in graph construction, which may not contain all the necessary information for training a classification model. Furthermore, the primary metadata information for different DApp flows is not always located in the initial packets, which further reduces the effectiveness of the model. As for the CBFM and CQNet methods, despite their specialized design for DApp encrypted traffic classification, their effectiveness remains suboptimal due to their reliance solely on byte information, lacking integration of more nuanced temporal or packet length details that could capture inter-flow relationships. This contributes to their diminished classification accuracy. Additionally, SPCaps, leveraging capsule networks for traffic classification, showcases strong performance attributed to the inherent enhanced information representation capabilities within capsule networks.

4) Through an examination of the principles and experimental results of the aforementioned methods, it can be inferred that the superiority of our proposed CapsuleFormer method in detecting DApp encrypted traffic over other methods is primarily attributed to Firstly, it combines the most common features of traffic in the feature construction process, and we additionally utilize the byte

Table 4: Results of the CapsuleFormer Compared with other Models for the Classification of Blockchain Encrypted Traffic

DApp	Appscanner		EDC		FAAR		GDA		CBFM		FS-Net		SPCaps		CQNet		CapsuleFormer	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
sushi	0.7530	0.9515	0.8931	0.8342	0.9880	0.9986	0.8723	0.8293	0.9112	0.9730	0.9859	0.9841	0.9996	0.8516	0.9761	0.9463	1.0000	1.0000
uniswap	0.5030	0.4918	0.2355	0.9636	0.5014	0.5694	0.8875	0.9233	0.9901	0.9223	0.5264	0.7712	0.9998	0.8507	0.9557	0.9713	1.0000	1.0000
solo.top	0.5458	0.4352	0.0000	0.0000	0.5107	0.3377	0.8952	0.8634	0.9202	0.9625	0.6080	0.4158	0.9996	0.9778	0.9659	0.9527	0.9009	1.0000
venus	0.5609	0.4204	0.6982	0.2504	0.5064	0.4032	0.8571	0.8906	0.9211	0.9532	0.4982	0.5629	1.0000	0.9576	0.9537	0.9670	1.0000	0.9600
pancakeswap	0.6143	0.3382	0.8432	0.1986	0.6438	0.4355	0.9132	0.8725	0.9703	0.9910	0.5636	0.6180	0.9990	0.8198	0.9623	0.9409	0.9901	1.0000
biswap	0.6702	0.5910	0.7203	0.4286	0.5605	0.6277	0.8327	0.8879	0.9055	0.8767	0.7485	0.6307	0.9991	0.9315	0.9677	0.9487	1.0000	1.0000
alpacafinance	0.7731	0.7874	0.9398	0.3928	0.9026	0.7682	0.8995	0.8997	0.9361	0.9616	0.8357	0.8239	1.0000	0.8711	0.9599	0.9780	1.0000	1.0000
dodoex	0.4327	0.7755	0.6988	0.3940	0.4454	0.7588	0.8608	0.9183	0.9407	0.9129	0.7919	0.4810	0.5271	0.9991	0.9789	0.9669	0.9895	0.9400
tokenlon	0.7800	0.6103	0.8045	0.5293	0.7815	0.6153	0.9113	0.8973	0.9321	0.9077	0.8490	0.6221	1.0000	0.9113	0.9779	0.9817	1.0000	0.9700
curve	0.7785	0.7063	0.9933	0.4443	0.7515	0.7156	0.8739	0.9029	0.9818	0.9349	0.6798	0.8244	1.0000	0.9568	0.9593	0.9549	1.0000	1.0000
F1-score	0.6128		0.4779		0.6307		0.8840		0.9398		0.6799		0.9223		0.9632		0.9871	
Accuracy	0.6366		0.5102		0.6607		0.8757		0.9502		0.7069		0.9004		0.9667		0.9870	

feature of packet payload. This allows us to retain more information from the original traffic information for model training. Secondly, our method utilized a feature selection algorithm improved by the Laplacian algorithm. This approach enabled us to effectively capture the optimal number of samples required for constructing the similarity matrix, thereby resulting in the best model performance. Thirdly, we use a capsule network, which has the advantage of using a vector to represent a neuron, which has richer semantic information than the traditional scalar neuron and is better for representing the traffic information during model processing. Finally, we use the Encoder Block, which can better measure the importance of the input vectors, i.e., it allows the traffic information to interact with each other, so that high-dimensional traffic information that is beneficial to the model classification can be filtered out. These advantages are not available in the previous models.

5.4 Ablation Study (RQ4)

To assess the validity and efficiency of the model, we conducted an ablation analysis of its components. We evaluated the results of the experiments on the DApps encrypted traffic dataset using precision, recall, F1, and accuracy metrics. The results are displayed in Table 5. We ablate the Transformer Block, Capsule Network, and both modules (using only the pure CNN architecture-based DF [27]) from our approach.

We observed that using the Transformer Block alone in our method led to improved evaluation metrics Precision and Recall for most categories compared to the pure CNN architecture-based DF module. Moreover, the F1 score increased by approximately 0.3 from 0.5259 to 0.8317, while the Accuracy increased by about 0.25 from 0.5850 to 0.8320. Using Capsule Network alone without Transformer Block also resulted in significant improvements in most category evaluation metrics compared to the DF method, with an increase of about 0.4 in F1 score and approximately 0.33 in Accuracy. The CapsuleFormer model that employed both Transformer Block and Capsule Network achieved the best performance, an increase of 0.077 in F1 score and 0.07 in Accuracy compared with the model without Transformer. Therefore, it can be inferred that among these types of models, the feature representation ability of the pure CNN architecture-based DF model is the weakest, while the Transformer Block has better feature representation ability than DF for DApp encrypted traffic. Incorporating both models (Transformer Block, Capsule Network) results in superior performance compared to employing either of the models individually.

5.5 Time Complexity (RQ5)

Feature generation process. In the feature generation phase, Table 6 highlights that EDC has the lowest time consumption per flow, at 2.056E-3 seconds. CBFM, on the other hand, is the method with the highest time consumption, at 1.259 seconds. Other methods such as Appscanner, EDC, FAAR, FS-Net, and others, rely on packet length or a supplementary time series for their initial features, which makes the computation process simpler and less time-intensive. GDA uses graph neural networks but only considers the first 25 packets of a flow when constructing the graph, requiring less information and resulting in shorter processing times. CQNet, as a non-end-to-end framework, initially employs a CNN pre-trained on the Fashion-MNIST dataset for feature extraction. Subsequently, it categorizes data into simpler and more complex subsets, both requiring time-intensive processes, leading to more time overhead. SPCas and CapsuleFormer have similar time consumption, with SPCas requiring more time due to its two-step flow separation procedure. Our method, which uses a wealth of flow information in the feature extraction stage, has a slightly higher time consumption.

Training and testing process. The training time consumption refers to the average time required by the classifier to train a single sample, while the test time consumption represents the average time taken by the trained classifier to classify unknown samples. As shown in Table 6, GDA is the method that consumes the least amount of time during the training process, taking an average of 6.286E-5 seconds to train a single flow. On the other hand, CapsuleFormer consumes the most time, with an average of 4.726E-3 seconds. Despite the similar time consumption for all methods, the

Table 5: Results of Ablation Study on Blockchain Encrypted Traffic dataset

DApp	DF		CapsuleFormer w/o Transformer Block		CapsuleFormer w/o Capsule Network		CapsuleFormer	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
sushi	0.0000	0.0000	0.9252	0.9900	0.8434	0.7000	1.0000	1.0000
uniswap	0.9250	0.7400	0.8571	0.7800	0.9487	0.7400	1.0000	1.0000
solo.top	0.0000	0.0000	1.0000	1.0000	1.0000	1.0000	0.9009	1.0000
venus	0.8761	0.9900	0.8929	1.0000	0.6993	1.0000	1.0000	0.9600
pancakeswap	0.5833	0.0700	0.9600	0.4800	0.7671	0.5600	0.9901	1.0000
biswap	0.9462	0.8800	0.8929	1.0000	0.9552	0.6400	1.0000	1.0000
alpacafinance	0.3550	0.9300	0.7578	0.9700	0.7959	0.7800	1.0000	1.0000
dodoex	0.8085	0.3800	0.9901	1.0000	1.0000	0.9800	0.9895	0.9400
tokenlon	0.3127	0.9100	0.9706	0.9900	0.5928	0.9900	1.0000	0.9700
curve	1.0000	0.9500	0.9897	0.9600	1.0000	0.9300	1.0000	1.0000
F1	0.5259		0.9101		0.8317		0.9871	
Accuracy	0.5850		0.9170		0.8320		0.9870	

number of model parameters is the primary factor that affects the training process, with all models having a similar number of parameters. During the testing process, GDA is the most efficient method, with a time consumption of 6.059E-7 seconds, while CapsuleFormer, with its larger input dimension, is the most time-consuming, taking an average of 1.373E-3 seconds. Despite the fact that CapsuleFormer did not exhibit superior performance in terms of feature generation time, training time, and testing time per sample, it is worth noting that all methods processed each sample in less than one second, and the difference in time consumption among other methods was relatively small. Hence, it can be inferred that the time consumption of CapsuleFormer has minimal impact. Additionally, CapsuleFormer achieved the best classification performance among the methods evaluated.

Table 6: A Comparison of the Duration of the Feature Generation, Training, and Testing Processes for Each Method

Methods	Feature generation process(s)	Training process(s)	Testing process(s)
Appscanner	6.619E-3	1.797E-3	1.793E-4
EDC	2.056E-3	5.144E-4	2.915E-4
FAAR	1.341E-2	1.089E-3	3.747E-5
SPCas	0.110	5.127E-4	2.503E-4
CBFM	1.259	3.619E-4	7.818E-5
FS-Net	5.839E-3	1.623E-4	7.954E-5
GDA	2.465E-3	6.286E-5	6.059E-7
CQNet	0.197	6.805E-4	5.955E-4
CapsuleFormer	0.750	4.726E-3	1.373E-3

6 DISCUSSION

Our method requires a relatively long time for feature extraction from flows as well as during training and testing of the model. Therefore, our system is not suitable for high real-time demands in DApp classification scenarios. Additionally, to enhance the speed of feature extraction, it is possible to consider reducing the time consumed by discarding less important features, while improving training and testing speed through model pruning and compression. The second limitation is that our system faces reduced accuracy when dealing with unmonitored DApps due to feature variations. This issue can be addressed by periodically updating the system model.

7 ETHICAL CONSIDERATIONS & DATA ACCESS

Due to the utilization of data from an actual DApp network for our evaluation, user security remained our foremost concern throughout the research period. The data collection employed proxy IP addresses, ensuring that users' real IP addresses were not exposed. Additionally, the number of accesses to DApp service servers was kept extremely limited in comparison to their actual user volume. Furthermore, we made the data publicly available upon acceptance of this paper, allowing other researchers to evaluate other approaches without having to collect new data samples.

8 CONCLUSION

In this paper, we propose a new feature selection algorithm for constructing efficient encrypted traffic datasets for DApps, and also introduce a novel classification model called CapsuleFormer, which can distinguish encrypted traffic for DApps. Instead of utilizing conventional scalar neurons, CapsuleFormer employs capsule neurons and incorporates Transformer blocks to upgrade feature representation through the mapping of training data to a higher dimensional space. The current study involved conducting experiments on ten diverse DApp dataset categories obtained from the Ethereum and BNB chains. The experimental results demonstrate that CapsuleFormer exhibits superior classification performance when compared to existing methods, surpassing them by a considerable margin. In future endeavours, our goal is to expand the capabilities of CapsuleFormer to provide even finer-grained traffic

classification in DApps and also to optimize its performance by reducing its processing time.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (61972219), the Overseas Research Cooperation Fund of Tsinghua Shenzhen International Graduate School (HW2021013), Singapore Ministry of Education AcRF Tier 2 (MOE-T2EP20121-0006), Guangdong Basic and Applied Basic Research Foundation(2022A1515010417), Key Project of Shenzhen Municipality(JSGG20211029095545002), and the Major Key Project of PCL (PCL2023A05).

REFERENCES

[1] [n. d.]. Dapps. https://www.dapp.com/dapps Accessed on November 20, 2022.
[2] Wei Cai, Gaopeng Gou, Minghao Jiang, Chang Liu, Gang Xiong, and Zhen Li. 2021. MEMG: Mobile Encrypted Traffic Classification With Markov Chains and Graph Neural Network. In *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 478–486.
[3] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 605–616.
[4] Yige Chen, Tianning Zang, Yongzheng Zhang, Yuan Zhou, Linshu Ouyang, and Peng Yang. 2021. Incremental learning for mobile encrypted traffic classification. In *ICC 2021-IEEE International Conference on Communications*. IEEE, 1–6.
[5] Scott E Coull and Kevin P Dyer. 2014. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *ACM SIGCOMM Computer Communication Review* 44, 5 (2014), 5–11.
[6] Susu Cui, Bo Jiang, Zhenzhen Cai, Zhigang Lu, Song Liu, and Jian Liu. 2019. A session-packets-based encrypted traffic classification using capsule neural networks. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 429–436.
[7] Yanjie Fu, Junming Liu, Xiaolin Li, and Hui Xiong. 2018. A multi-label multi-view learning framework for in-app service usage analysis. *ACM Transactions on Intelligent Systems and Technology (TIST)* 9, 4 (2018), 1–24.
[8] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2022. Surakav: Generating Realistic Traces for a Strong Website Fingerprinting Defense. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1558–1573.
[9] Jamie Hayes, George Danezis, et al. 2016. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique.. In *USENIX security symposium*. 1187–1203.
[10] Xiaofei He, Deng Cai, and Partha Niyogi. 2005. Laplacian score for feature selection. *Advances in neural information processing systems* 18 (2005).
[11] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. 2018. Matrix capsules with EM routing. In *International conference on learning representations*.
[12] Ting-Li Huoh, Yan Luo, and Tong Zhang. 2021. Encrypted Network Traffic Classification Using a Geometric Learning Model. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 376–383.
[13] Amirhossein Khajehpour, Farid Zandi, Navid Malekghaini, Mahdi Hemmatyar, Naeimeh Omidvar, and Mahdi Jafari Siavoshani. 2022. Deep Inside Tor: Exploring Website Fingerprinting Attacks on Tor Traffic in Realistic Settings. In *2022 12th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 148–156.
[14] Maciej Korczyński and Andrzej Duda. 2014. Markov chain fingerprinting to classify encrypted traffic. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 781–789.
[15] Wenbin Li and Gaspard Quenard. 2021. Towards a Multi-Label Dataset of Internet Traffic for Digital Behavior Classification. In *2021 3rd International Conference on Computer Communication and the Internet (ICCCI)*. IEEE, 38–46.
[16] Zhen Ling, Gui Xiao, Wenjia Wu, Xiaodan Gu, Ming Yang, and Xinwen Fu. 2022. Towards an efficient defense against deep learning based website fingerprinting. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 310–319.
[17] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. 2019. Fs-net: A flow sequence network for encrypted traffic classification. In *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 1171–1179.
[18] Xue Liu, Shigeng Zhang, Huihui Li, and Weiping Wang. 2021. Fast Application Activity Recognition with Encrypted Traffic. In *Wireless Algorithms, Systems, and Applications: 16th International Conference, WASA 2021, Nanjing, China, June 25–27, 2021, Proceedings, Part II 16*. Springer, 314–325.
[19] Jie Lu, Gaopeng Gou, Majing Su, Dong Song, Chang Liu, Chen Yang, and Yangyang Guan. 2021. GAP-WF: Graph attention pooling network for fine-grained SSL/TLS Website fingerprinting. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.
[20] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale.. In *NDSS*.
[21] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2017. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376* (2017).
[22] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. *Advances in neural information processing systems* 30 (2017).
[23] Meng Shen, Mingwei Wei, Liehuang Zhu, and Mingzhong Wang. 2017. Classification of encrypted traffic with second-order markov chains and application attribute bigrams. *IEEE Transactions on Information Forensics and Security* 12, 8 (2017), 1830–1843.
[24] Meng Shen, Mingwei Wei, Liehuang Zhu, Mingzhong Wang, and Fuliang Li. 2016. Certificate-aware encrypted traffic classification using second-order markov chain. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.
[25] Meng Shen, Jinpeng Zhang, Liehuang Zhu, Ke Xu, and Xiaojiang Du. 2021. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Transactions on Information Forensics and Security* 16 (2021), 2367–2380.
[26] 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 *Proceedings of the International Symposium on Quality of Service*. 1–10.
[27] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1928–1943.
[28] Heyi Tang, Yong Cui, Jianping Wu, Xiaowei Yang, and Zhenjie Yang. 2019. Trigger relationship aware mobile traffic classification. In *Proceedings of the International Symposium on Quality of Service*. 1–10.
[29] Vincent F Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. 2016. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 439–454.
[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[31] Shihao Wang, Liangmin Wang, Shangnan Yin, Hui Zhao, and Hao Shentu. 2020. CPWF: Cross-platform website fingerprinting based on multi-similarity loss. In *2020 International Conference on Networking and Network Applications (NaNA)*. IEEE, 73–80.
[32] Yu Wang, Zhenzhen Li, Gaopeng Gou, Gang Xiong, Chencheng Wang, and Zhen Li. 2020. Identifying DApps and user behaviors on ethereum via encrypted traffic. In *Security and Privacy in Communication Networks: 16th EAI International Conference, SecureComm 2020, Washington, DC, USA, October 21-23, 2020, Proceedings, Part II 16*. Springer, 62–83.
[33] Yu Wang, Chencheng Wang, Gang Xiong, and Zhen Li. 2021. Multi-scene Classification of Blockchain Encrypted Traffic. In *Blockchain and Trustworthy Systems: Third International Conference, BlockSys 2021, Guangzhou, China, August 5–6, 2021, Revised Selected Papers 3*. Springer, 329–337.
[34] Yu Wang, Gang Xiong, Chang Liu, Zhen Li, Mingxin Cui, and Gaopeng Gou. 2021. CQNet: A clustering-based quadruplet network for decentralized application classification via encrypted traffic. In *Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part IV 21*. Springer, 518–534.
[35] Chengcheng Xu, Shuhui Chen, Jinshu Su, Siu-Ming Yiu, and Lucas CK Hui. 2016. A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms. *IEEE Communications Surveys & Tutorials* 18, 4 (2016), 2991–3029.
[36] Feipeng Yan, Ming Xu, Tong Qiao, Ting Wu, Xue Yang, Ning Zheng, and Kim-Kwang Raymond Choo. 2018. Identifying wechat red packets and fund transfers via analyzing encrypted network traffic. In *2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*. IEEE, 1426–1432.
[37] Chen Yang, Can Wang, Weidong Zhang, Huiyi Zhang, and Xuangou Wu. 2023. Decentralized Application Identification via Burst Feature Aggregation. In *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 1551–1556.