Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 32(1): 193–223.
DOI: 10.3934/era.2024010
Received: 15 September 2023
Revised: 11 December 2023
Accepted: 13 December 2023
Published: 20 December 2023

Research article

A novel method for mobile application recognition in encrypted channels

Jiangtao Zhai[1,2,*], **Zihao Wang**[1,2], **Kun Duan**[1,2] **and Tao Wang**[1,2]

[1] School of Electronic & Information Engineering, Nanjing University of Information Science & Technology, Nanjing 210044, China

[2] Key Laboratory of Intelligent Support Technology for Complex Environments, Ministry of Education, Nanjing 210044, China

* **Correspondence:** Email: jiangtaozhai@nuist.edu.cn.

Abstract: In the field of mobile application traffic analysis, existing methods for accurately identifying encrypted traffic often encounter challenges due to the widespread adoption of encryption channels and the presence of background traffic. Consequently, this study presents a novel mobile application traffic identification model that is in encrypted channels. The proposed model utilizes an adaptive feature extraction technique that combines Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs) to effectively extract abstract features from encrypted mobile application traffic. Additionally, by employing a probability-based comprehensive analysis to filter out low-confidence background traffic interference, the reliability of recognition is further enhanced. Experimental comparisons are conducted to validate the efficacy of the proposed approach. The results demonstrate that the proposed method achieves a remarkable classification accuracy of 95.4% when confronted with background traffic interference, surpassing existing techniques by over 15% in terms of anti-interference performance.

Keywords: mobile application traffic analysis; encryption channels; Convolutional Neural Networks; Gated Recurrent Neural Networks; background traffic interference

Abbreviations: CNN: Convolutional Neural Network; GRU: Gated Recurrent Unit; DPI: Deep Packet Inspection; TCP: Transmission Control Protocol; IP: Internet Protocol; RNN: Recurrent Neural Network; LSTM: Long Short-Term Memory; SSR: ShadowSoksR; XAI: Explainable Artificial Intelligence

Math symbols

z_t	The update gate of GRU
r_t	The reset gate of GRU
x_t	The input of GRU
h	The output of GRU
h_t	The candidate output of GRU
z_j	The raw score of the j_{th} neuron
e^{z_j}	The exponential value for each element
p_j	The probability that the input sample belongs to the j_{th} class
$P_{\max}$	The node with the highest probability
M	The number of mobile applications
TP	Positive samples are correctly identified as positive samples
TN	Negative samples are correctly identified as negative samples
FP	Negative samples are misidentified as positive samples
FN	Positive samples are misidentified as negative samples

1. Introduction

In recent years, there have been significant changes in the way users access the Internet, and the traffic generated by mobile devices has exploded. Mobile applications have become an indispensable part of people's daily lives, and user behavior recognition in mobile applications has become one of the hot research directions in the field of mobile Internet [1]. By monitoring and recognizing the flow of mobile application traffic, network security issues such as malicious behavior, network attacks and privacy leaks can be detected and protecting user data and privacy is of great significance for network security control [2]. In addition, the research results can also be used in fields such as mobile application recommendation, advertising delivery and user behavior analysis. Therefore, the research on the recognition and management of user behavior flow in mobile applications has important research value and practical significance.

Although mobile application traffic identification study is similar to PC traffic identification work, the uniqueness of mobile application traffic, such as the fast iteration speed of application versions and data transmission through encryption protocols, has posed greater challenges to traditional identification methods [3, 4]. Currently, mobile traffic identification can be classified into four categories: port-based classification methods, Deep Packet Inspection (DPI) based classification methods, traditional machine learning-based classification methods and deep learning-based classification methods.

Initially, the major means of classifying network traffic were based on TCP/UDP packet port numbers. However, since many applications do not use registered port numbers or disguise port numbers, this method is often not effective. DPI-based classification methods are a typical rule-based method that requires manual rule-making and matching to achieve traffic identification. This method is time-consuming and becomes less applicable with more encrypted traffic [5]. Traditional machine learning-based classification methods require complex feature engineering techniques to achieve better accuracy, and this method may become ineffective after mobile application updates, which is also one of the bottlenecks faced by machine learning development [6, 7]. Deep learning-based classification methods can optimize feature engineering on their own [8, 9], solving the problem of over-reliance on manual feature extraction accuracy.

With the increasing number of installed applications on mobile devices, these applications may run automatically in the background and generate traffic even when the user has not opened them. This poses significant challenges for identifying mobile application traffic. The problem becomes even more severe when communicating through an encrypted channel. Therefore, we propose mobile application traffic identification model that is resilient to background traffic interference under encrypted channels. The model aims to achieve accurate classification of mobile application traffic in encrypted channels and address the issue of background traffic interference. Furthermore, multiple sets of experiments are conducted to evaluate the performance of the model in classifying mobile application traffic in encrypted channels. The main contributions of this work are as follows:

1) The adoption of a burst-flow diversion method, where traffic is grouped based on the time intervals between packet arrivals, source-destination IP addresses and port numbers. Different thresholds (0.005 s, 0.05 s, 0.5 s and 1 s) for grouping are compared to assess their impact on the classifier, aiming to find the optimal threshold that enhances the accuracy and robustness of the model's classification.

2) A neural network which combines Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs) has been designed. This approach utilizes CNNs to extract spatial features from network traffic and employs GRUs to model the temporal sequence, enabling the capturing of dynamic variations in mobile application traffic. Extensive comparative experiments have been conducted to determine the optimal model parameters. This enables the model to achieve superior performance in traffic recognition under encrypted channels.

3) In scenarios where unknown applications generate background traffic interference, the proposed model not only accurately detects the traffic of known applications under encrypted channels but also incorporates a filtering module to filter out background traffic. The paper explores and analyzes the background traffic detection rate and false positive rate under different confidence thresholds, selecting an appropriate confidence threshold that achieves an optimal balance between background traffic detection and false positives. This leads to an improved recognition accuracy of the model and a stronger anti-interference capability compared to existing methods.

The organization of the paper is as follows. Section 2 provides a comprehensive review of related work in the field of mobile application traffic identification. Section 3 presents the framework of the proposed method. Section 4 describes the details of the proposed method. Section 5 presents the experimental setup and discuss the results and analysis. Finally, Section 6 concludes the paper with a summary of the findings and suggestions for future research.

2. Related works

The paragraph describes several studies that explore the effectiveness of various machine learning methods for network traffic classification [10,11]. For instance, reference [12] successfully identified thousands of applications using the traffic features generated during application launch. However, when the training and testing datasets are collected from different devices, the accuracy drops by as much as 26%. Taylor et al. partitioned the network traffic using adjacent packet time interval thresholds, then performed finer-grained partitioning based on four-tuple information and extracted 18 features related to packet length from packets sequences in different directions. They used support vector machines and random forests to establish classification models, achieving application classification accuracy rates of up to 98% [13].

Similarly, Park et al. [14] used machine learning techniques to identify traffic patterns generated by instant messaging application Kakao Talk. Its method selected packet length as the feature sequence and achieved a recognition accuracy rate of 99.7% for Kakao Talk encrypted traffic, albeit with poor scalability. Saltaformaggio et al. proposed a user behavior recognition system called NetScop, which can be deployed on Wi-Fi access points or other network devices. The system had an identification accuracy rate of 78% [15]. Reference [16] found that by analyzing side information such as the size of Apple's iMessage network packets, it is possible to identify message length and language type, as well as to distinguish between five user behaviors such as sending messages, inputting and reading, with classification accuracy rates over 90%.

Conti et al. extracted features from the relevant information in the data stream, such as packet length and transmission direction, and proposed a data stream analysis method based on hierarchical clustering to solve the problem of multiple data streams generated by each operation in the application [17]. They applied clustering to the data stream and performed label operation, followed by classification of the behavior operation using random forests. The experimental results show that its identification accuracy rate can up to 95%. These methods rely heavily on the accuracy of feature engineering by domain experts, which is time-consuming and of limited generality and can become ineffective after mobile application upgrades. Deep learning avoids the need for feature engineering by domain experts and has stronger capabilities in learning complex patterns than traditional machine learning methods. The following works demonstrate the effectiveness of deep learning methods for network traffic classification. For example, Wang et al. built a stacked autoencoder classifier model which is used to classify 58 common protocols with accuracy and recall rates exceeding 90% [18]. Hu et al. proposed a CLD-Net model that combines CNN and LSTM to distinguish network encrypted traffic and accurately recognize Facebook and Skype application traffic (chat, audio, or file) on the ISCX public dataset [19]. Aceto et al. proposed two frameworks called MIMETIC and DISTILLER respectively, where MIMETIC requires two inputs for model training, payload information and protocol/time series features, to classify traffic [20]. Multitask and multimode deep learning models are suitable for mobile application classification using the DISTILLER framework. Wang et al. proposed an end-to-end encrypted traffic classification model based on 1D-CNN, which extracts features and selects them before performing classification [21]. This model was validated using the ISCX public dataset and showed that the classification performance of 1D-CNN is superior to that of 2D-CNN.

Table 1 illustrates the categorized reviewed works based on their primary features. Despite the satisfactory identification performance of existing deep learning-based methods for mobile application traffic identification, there exists a problem of insufficient feature extraction in current works. Additionally, these methods fail to consider the scenario where unknown applications generate background traffic interference in real-world usage,indicating their limitations. Existing works focus on closed testing scenarios for traffic identification, where the training and testing datasets consist of the same traffic classes. This causes a significant decrease in classification accuracy when facing background traffic interference. The interference issue becomes more severe when users communicate through encrypted channels, as both mobile application traffic and background traffic are encrypted, leading to confusion between background traffic and the desired mobile application traffic, thereby affecting the accuracy of mobile application traffic identification. We present a mobile application traffic identification model that is resilient to background traffic interference under encrypted channels. We utilize a neural network approach that combines CNNs and GRUs because CNNs excel in extracting spatial features, while GRUs are adept at handling temporal features among samples. By integrating these two, we can effectively extract latent spatial features from the data and capture temporal characteristics among samples. This methodology empowers the model to maximize feature extraction from mobile application traffic within encrypted channels, consequently enhancing recognition accuracy. Following this, we conducted multiple sets of experiments to evaluate the model's performance in classifying mobile application traffic within encrypted channels.

Table 1. Characteristics of reviewed works.

Author	Primary feature
Alan et al. [12]	Using traffic features generated during application launch
Taylor et al. [13]	Extracted 18 features related to packet length from packet sequences in different directions
Park et al. [14]	The packet length as a feature sequence and its scalability is poor
Saltaformaggio et al. [15]	User behavior recognition system deployed on Wi-Fi devices
Coull et al. [16]	Analyzing additional information to identify user behavior
Conti et al. [17]	Hierarchical clustering-based data stream analysis method
Wang et al. [18]	Stacked autoencoder classification model
Hu et al. [19]	CLD-Net model combining CNN and LSTM for distinguishing network encrypted traffic
Aceto et al. [20]	MIMETIC and DISTILLER frameworks
Wang et al. [21]	End-to-end encrypted traffic classification model based on 1D-CNN

3. The framework of the proposed method

The overall framework of the mobile application user behavior classification model proposed in this paper is shown in Figure 1, which mostly includes three modules: The data preprocessing module, feature extraction module and background traffic filtering module.

In the data preprocessing module, this paper employs three steps to process the experimental dataset: initial screening, traffic grouping, and image transformation. In the traffic grouping step, a Burst-flow diversion method is utilized to divide the traffic samples. The grouping is performed based on the time intervals between data packet arrivals, source and destination IP addresses and port numbers. After grouping, the byte length of the flow samples is standardized to facilitate subsequent model input and processing.

In the feature extraction module, a neural network design combining CNN and GRU is utilized. This design incorporates the advantages of both CNN and GRU models. It not only accurately and efficiently extracts potential spatial features from the data but also captures temporal features between samples. This allows the model to maximize the extraction of characteristics from mobile application traffic under encrypted channels, thereby enhancing the recognition accuracy.

In the background traffic filtering module, the model's output probability distribution is compared with a preset confidence threshold. If the predicted probability of a certain class is greater than or equal to the confidence threshold, that class is considered the final prediction result. If the predicted probability of a certain class is below the confidence threshold, that class is identified as background traffic and filtered out, not included in the final prediction result. The introduction of the background traffic filtering module effectively reduces interference from background traffic and abnormal data, thereby improving the stability and generalization ability of the model.

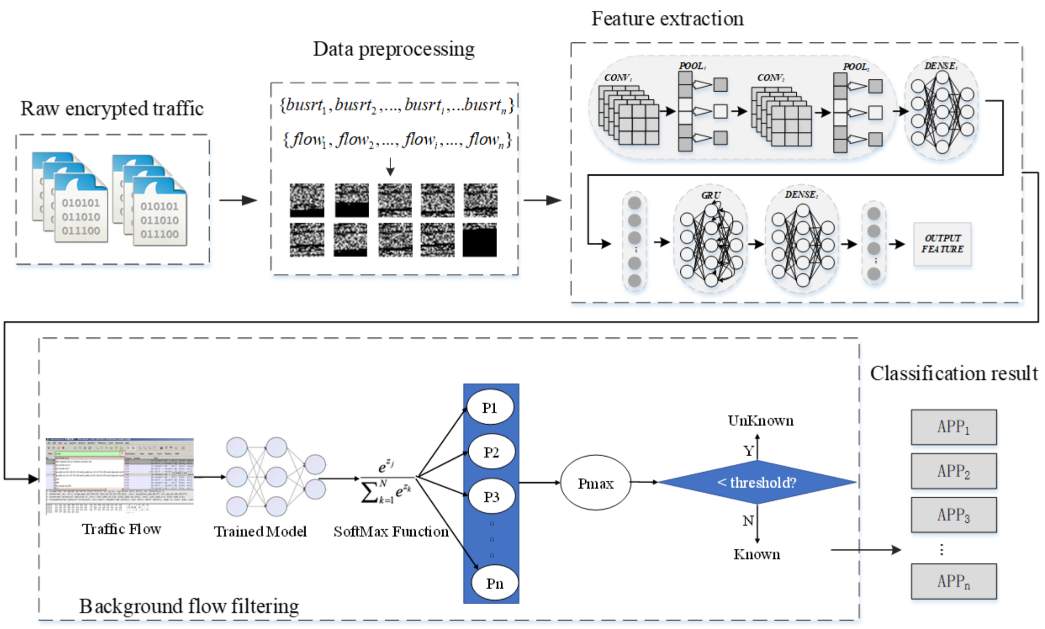


Figure 1. Overall framework of the proposed method.

4. The proposed method

4.1. The data pre-processing module

Traffic datasets are typically stored and distributed in .pcap or .pcapng format, and they cannot be directly classified in most cases. Preprocessing of such files is necessary to convert the raw network traffic into a data format suitable for inputting into a classification model. In this paper, the processing of the dataset mainly involves three steps: screening of traffic, traffic grouping, and image transformation. The specific process is as follows.

4.1.1. Initial screening of traffic

In practical network environments, there can be a certain number of TCP retransmissions and corrupt packets. The occurrence and frequency of TCP retransmissions and corrupt packets are primarily dependent on the current network conditions. If a large number of TCP retransmissions and corrupt packets are saved along with the data packets generated by normal communication of mobile applications, it can interfere with the training of the classifier model. Therefore, it is necessary to filter them out.

One common approach is to filter based on packet characteristics, such as examining packet sequence numbers, checksums and acknowledgment numbers to identify and eliminate packets that might be retransmitted or damaged. Another method involves utilizing the mechanisms within the TCP protocol to filter retransmitted packets, for instance, inspecting the flags in the TCP header to identify retransmitted packets. Additionally, using information such as the order of packet arrival, along with network status and protocol specifications, can help filter out abnormal data that might interfere with training. Here, we employ a method based on packet characteristics for filtering.

4.1.2. Traffic grouping

We adopt the Burst-flow approach to partition the traffic based on the time intervals between packets, source and destination IP addresses and port numbers. The reason we use the burst-flow method to divide the data is twofold. First, the content of traffic within encrypted channels is challenging to analyze directly. By analyzing the bursts in traffic, it becomes possible to make certain inferences about the transmission patterns, frequency, or size of the encrypted data, aiding in understanding the characteristics of data transmission. Second, within the same encrypted channel, different applications may be carried, and from an observer's perspective, these applications share identical quintuples. Employing clustering provides a more convenient way to distinguish between them. For a collection of data packets corresponding to a specific mobile application, the packets are sorted based on timestamp labeling to ensure that packets with the same timestamp are grouped together in the same burst group. Next, a threshold value for bursts needs to be determined. If the time difference between the current packet and the previous packet is less than the burst threshold, the current packet is assigned to the burst group where the previous packet belongs. Otherwise, it is considered as the first packet of a new burst group. As described in Section 5.4.1, this section presents comparative experiments with different burst thresholds (0.005 s, 0.05 s, 0.5 s and 1 s) to evaluate the impact of different burst thresholds on classification accuracy. Ultimately, 0.5 s is selected as the burst threshold for the experimental setup in this paper.

After this step, the burst dataset is obtained, denoted as $BurstDat = \{busrt_1, busrt_2, ..., busrt_i, ..., busrt_n\}$. $busrt_i$ represents the $i_t h$ burst group, and n represents the total number of burst groups. The current burst group may contain data packets generated by more than one device, so a more detailed subdivision of the burst groups is required based on the source and destination IP addresses.

By traversing the burst dataset, for each burst group $busrt_i$, packets with the same or opposite source and destination IP addresses, as well as the same or opposite source and destination port numbers, are combined to form a flow data group. The packets within each flow group are sorted in ascending order based on their timestamps. After this step, the flow dataset is obtained, denoted as $FlowData = \{flow_1, flow_2, ..., flow_i, ..., flow_n\}$. $flow_i$ represents the $i_t h$ data flow group, and n represents the total number of data flow groups. Refers to the data object that will be further transformed into grayscale images.

4.1.3. Image transformation

In order to facilitate the input and processing of subsequent models, the post-partitioned data samples are processed to have a unified byte length. During the establishment of communication, there is frequent interaction between the sender and receiver, and the headers contain more valuable information. Therefore, the original byte sequences of the data packets are extracted starting from the header of the flow sample. The Maximum Transmission Unit (MTU) defines the maximum packet length as 1500 bytes in the network. As described in Section 5.4.2, after practical testing, a standard length of 784 bytes is chosen in this study. If the byte count of a flow sample exceeds the standard length, the first bytes up to the standard length are extracted from the header. If the byte count is less than the standard length, it is padded with 0×00 bytes to reach the standard length. The processed data samples are then transformed into grayscale images. Figure 2 displays the traffic of different applications in grayscale format, representing the flow through encrypted channels.

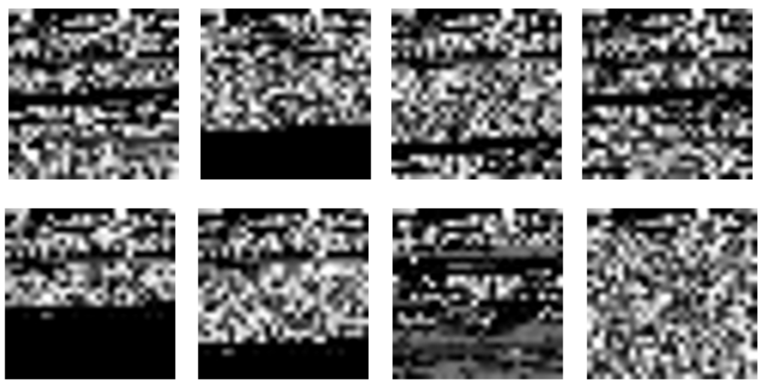


Figure 2. Gray-scale representation of mobile application traffic under an encrypted channel.

In a grayscale image, each pixel can have 256 shades, with 0×00 representing black and $0 \times \mathrm{ff}$ representing white. One byte consists of 8 bits, and the shade of each pixel in the grayscale image is determined by the value of each byte. The standard-length bytes are transformed into grayscale images of size 28×28. The varying interaction behaviors of different types of traffic result in different compositions of the original byte sequences of the data packets. As a result, the generated grayscale images exhibit distinct texture styles, which possess strong representational capabilities.

4.2. Feature extraction module

The neural network structure designed for feature extraction in the mobile app user behavior classification model in this paper is a combination of Convolutional Neural Network (CNN) and Gated Recurrent Unit (GRU), which combines the advantages of CNN and GRU. It can not only accurately and efficiently mine potential spatial features from data, but also extract temporal features between samples. CNN accurately extracts the features of traffic samples through the convolutional layer, and then reduces the dimension of the extracted feature information through the pooling layer to reduce the computational complexity of the network. Considering that network traffic data is structured sequential data, this paper uses CNN as part of the neural network structure to learn the spatial features of traffic data. In addition, this paper also uses recurrent neural networks to learn the temporal features of network traffic sequences. LSTM and GRU are two widely used recurrent neural networks to eliminate the gradient vanishing and explosion problems of traditional RNN. Compared with LSTM, GRU can better capture the dependencies with larger intervals in the temporal data, and has fewer parameters and shorter training time in the training process, which is why we choose GRU as the neural network for this paper.

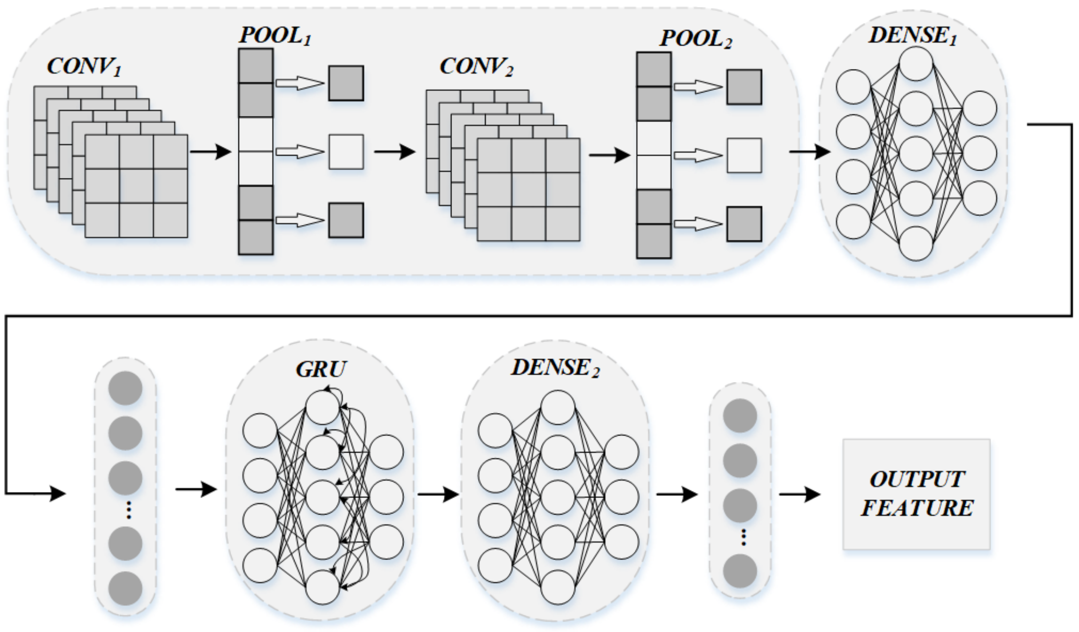


Figure 3. Feature extraction network model structure.

In the feature extraction module, the designed neural network is used for model training. First, the preprocessed samples are input into the CNN network to extract spatial features. Then, the extracted features are integrated and sent to the GRU network to extract the temporal features of traffic. Finally, the predicted results are output through the fully connected layer and Softmax layer. The specific structure of the feature extraction network model in this paper is shown in Figure 3. We select hyperparameters, such as the number of convolutional layers, the number of fully connected layers, the stride size and the activation function, through comprehensive testing involving numerous parameter combinations. The depth of CNN should neither be too large nor too small, so that it can learn complex relationships while keeping the model converging.

4.2.1. Spatial feature learning

The spatial feature learning module of this network framework consists of the input layer, convolutional layer, pooling layer and fully connected layer of traditional CNN [22, 23]. The specific design model architecture parameters are shown in Table 2. The input format of the input layer matches the output format of the preprocessing, which is a fixed format of M × N. To select the best input format, we first set the initial range of M to 10–35 and the range of N to 10–35, and then randomly selects different sizes for a series of comparative experiments. After considering all parameters, using a 28 × 28 input format can achieve the best balance between classification performance and computational efficiency.

The proposed model adopts a two-layer convolutional structure for feature extraction, each convolutional layer using 32 convolution kernels of size 3 × 3 to process input data, generating 32 feature maps. In the first convolutional layer, the size of the generated feature map is 28 × 28, while in the second convolutional layer, the size of the generated feature map is 14 × 14. To introduce non-linear transformation, the activation function after the two convolutional operations is the Rectified Linear Units (ReLU). Compared with traditional Sigmoid and Tanh functions, ReLU does not require exponential and derivative calculations, making it more computationally efficient and faster. In addition, ReLU can alleviate the problem of gradient vanishing in deep neural networks, thus, it is chosen as the activation function in this paper [24].

In the pooling layer, the maximum pooling function with a kernel size of 2×2 is used for pooling, which takes the maximum value within the adjacent matrix region as the pooling result. To avoid overfitting, L2 regularization and Dropout techniques are used for training optimization. L2 regularization can penalize high weight values to avoid overfitting to the training data, while Dropout can randomly set neuron outputs to 0, forcing the model to learn more robust features and preventing severe co-adaptation between neurons [25].

Table 2. CNN model architecture parameter list.

Layer number	Operation	Input format	Output format
1	Input	28×28	28×28
2	Convolution	28×28	$32 \times (28 \times 28)$
3	Pooling	$32 \times (28 \times 28)$	$32 \times (14 \times 14)$
4	Convolution	$32 \times (14 \times 14)$	$32 \times (14 \times 14)$
5	Pooling	$32 \times (14 \times 14)$	$32 \times (7 \times 7)$
6	FC	$32 \times (7 \times 7)$	128×1

4.2.2. Time series feature learning

The GRU (Gated Recurrent Unit) is an improved version of the LSTM (Long Short-Term Memory) with fewer parameters and only two gates, but with similar functionality [26]. Considering the hardware computational capacity and time cost, this paper uses the GRU to learn the time features of the mobile app user behavior data flow. The feature vector obtained by the Convolutional Neural Network is input into the GRU module for learning to obtain the time features of the data flow. The detailed internal structure of the GRU is shown in Figure 4.

GRU combines the input gate and forget gate of LSTM into a single gate called the update gate, denoted as z_t in the diagram. The update gate determines how much past and new information to retain at the current time step. It controls a combination of the input and forget gates, dictating the extent to which the new candidate value will be incorporated into the current cell state. Specifically, the update gate's computation involves utilizing a sigmoid activation function based on the current input and the hidden state from the previous time step to generate a value between 0 and 1. This value decides how much information gets updated into the state at the current time step.

GRU also has another gate called the reset gate, denoted as r_t in the diagram, which controls how much past information should be forgotten. The reset gate determines how much of the previous hidden state should be ignored at the current time step. It assists the network in deciding whether to retain past information and which information to discard. The computation of the reset gate is similar to that of the update gate, employing a sigmoid activation function based on the current input and the hidden state from the previous time step to generate a value between 0 and 1.

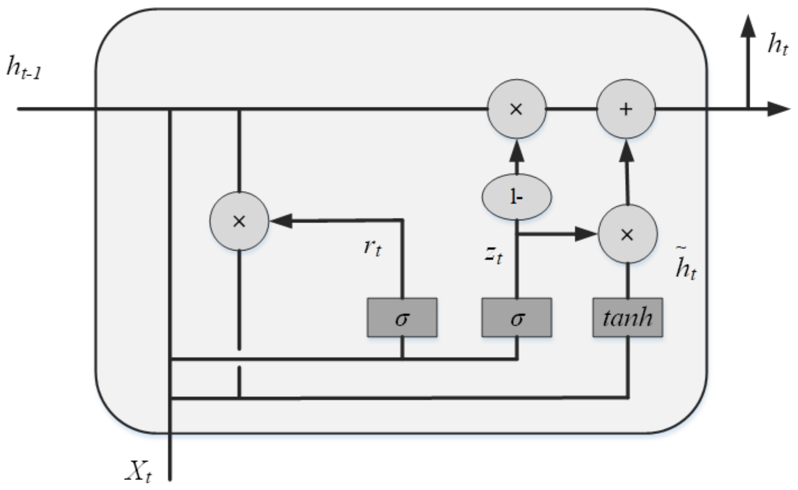


Figure 4. The structure of GRU.

In the diagram, x_t represents the input, h represents the output and h_t represents the candidate output. Subscripts and denote the current and previous time steps, respectively. The amount of past memory information that can continue to be retained at the current time step is controlled by z_t, or in other words, it determines how much information from the previous time step and the current time step should be passed on to the future. The expressions for each parameter are shown in Eqs (4.1)–(4.4).

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \tag{4.1}$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \tag{4.2}$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t \times h_{t-1}, x_t]\right) \tag{4.3}$$

$$h_t = (1 - z_t) \times h_{t-1} + z_t \times \tilde{h}_t \tag{4.4}$$

4.2.3. Background traffic filter module

With the increasing number of installed applications on mobile devices, these applications may run automatically in the background and generate traffic even when the user has not opened them. This poses significant challenges for identifying mobile application traffic. The problem becomes even more severe when communicating through an encrypted channel. Under an encrypted channel, both mobile application traffic and background traffic are encrypted, making it difficult to distinguish between them. This confusion hampers the accuracy of mobile application traffic identification, as background traffic gets mixed with the desired mobile application traffic, making it challenging to differentiate them accurately. The confusion between background traffic and mobile application traffic complicates the model's learning process, making it challenging to accurately differentiate and understand genuine behavioral patterns of mobile applications. This confusion leads to unstable predictive outcomes, impacting the model's generalization ability and its capability to identify unknown data. Additionally, the presence of background traffic interference may compel the model to handle a considerable amount of noisy data during training, elevating the complexity and time costs of the training process. Hence, to address the issue of confusion between background traffic and mobile application traffic, there is a need for a background traffic filtering method.

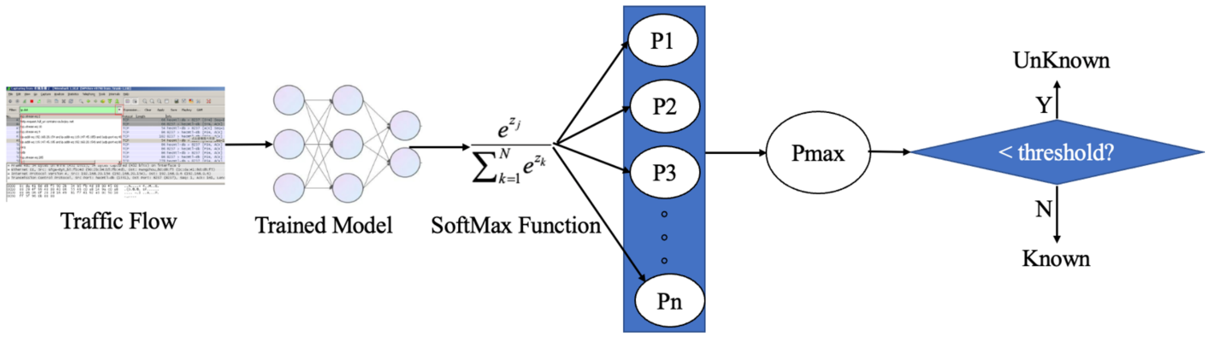


Figure 5. The filter principle of the background traffic.

The model proposed in this paper effectively reduces the interference of background traffic and abnormal data by constructing a background traffic filtering module to identify and filter untrained application behavior as unknown traffic, thus improving the stability and generalization ability of the model. The principle of background traffic filtering is shown in Figure 5. First, the data stream is input to the trained model presented in Section 4.2. When a data stream is input to the model, the output layer of the model converts the raw scores of each class into probabilities using the Softmax function. Specifically, for a classification problem with N classes, the output layer of the model usually has N neurons, each of which corresponds to a class, and outputs a real number as the raw score for that class. Then, the Softmax function is used to convert these raw scores into probabilities for each class. The definition of the Softmax function is as follows:

$$\operatorname{softmax}\left(z_j\right) = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} \tag{4.5}$$

where z_j is the raw score of the $j_t h$ neuron, and is the total number of neurons. The Softmax function transforms each raw score into a non-negative real number p_j representing the probability that the input sample belongs to the $j_t h$ class, and satisfies:

$$\sum_{j=1}^{N} p_j = p_1 + p_2 + \cdots + p_N = 1 \tag{4.6}$$

In general, for any given input, the value of one node should be higher than the values of other output nodes. As shown in Figure 5, the node with the highest probability is denoted as P_{max}. The decision to convert the predicted probabilities into class labels is controlled by a parameter called the confidence threshold. If P_{max} is less than the threshold, the input traffic is treated as an unknown sample. Among all predicted results, the samples with confidence above the threshold are retained, while the samples below the threshold are filtered out. Using the confidence threshold to filter samples, the interference of background traffic and abnormal data can be effectively reduced, and the stability and generalization ability of the model can be improved. Furthermore, training efficiency can be improved, and training costs can be reduced. In order to test the impact of the threshold on the model performance, a series of thresholds were selected and tested, and the threshold parameter with the highest classification accuracy was ultimately selected, as described in Section 5.5.2.

4.2.4. Background traffic filter module

The algorithm first utilizes CNN to extract spatial features from traffic samples and then employs GRU to learn the temporal features of network traffic data. Finally, it integrates these features for user behavior classification. Through this combined approach, the algorithm accurately extracts spatial and temporal features from the data to better classify user behavior. Next, we will assess the algorithm's complexity from the perspectives of time complexity and model parameter count.

In terms of time complexity, a CNN involves operations like convolution, pooling and activation functions. Typically, for a CNN with N layers and M filters, the time complexity per sample is often denoted as $O(N * M * H * W)$, where H and W represent the height and width of the feature maps. The time complexity of a GRU primarily depends on its matrix multiplication, element-wise operations and non-linear activation functions. For a time series data of length T, the time complexity of a GRU is usually $O(T * D^2)$, where D represents the GRU unit's dimension.

Regarding the model parameter count, the parameter quantity in a CNN model relates to its number of layers, filter count and the number of neurons in each layer. Typically, the parameter count of a CNN model is roughly $O(N * M * L)$, where L represents the number of neurons. The parameter count of a GRU model is determined by the number of units and the input dimension. For a model with N GRU units, the parameter count is approximately $O(N * D^2)$, where D is the input dimension.

5. Experiments and analysis

5.1. Experimental environment

The operating system used in the experiments of this paper was the 64-bit Windows 10 operating system, with an Intel Core i7-9750H/2.60 GHz CPU, 16 GB of memory, Keras as the deep learning platform, TensorFlow 1.8.0 as the deep learning backend and Python 3.6.2 as the development environment. Cross-entropy was used as the loss function during training of the deep neural network, which outputs probability values between 0 and 1 and measures the performance of the classification model, as defined in Eq (5.1).

$$H(y, p) = -\sum_{i=1}^{M} y_i \log(p_i) \tag{5.1}$$

where M is the number of mobile applications, p_i is the probability calculated through regression by the Softmax function. The model was optimized using the gradient descent method, with a mini-batch size of 64 and a learning rate of 0.01, and trained for approximately 50 epochs. Neural networks are often overtrained, so it is important to validate and test the trained model.

The ten-fold cross-validation method was used in this study in order to reduce the influence of randomness and chance on model evaluation results for a single test set. This method divides the original dataset into 10 mutually exclusive subsets, and each time selects one subset as the validation set and the remaining 9 subsets as the training set for model training. This process is repeated 10 times, with each subset taking turns as the validation set. Finally, the results of these 10 runs are averaged to obtain the final result, which effectively evaluates the performance and feasibility of the algorithm. This method can make the evaluation results more statistically significant and reduce the degree of overfitting of the model to specific datasets, thereby better evaluating the performance of the model on unknown datasets.

5.2. Experimental dataset

The experimental dataset in this study was established based on a laboratory environment using the ShadowSocksR (SSR) proxy software to create an encrypted channel. SSR establishes an encrypted channel between the local device and a remote server to encrypt network traffic. This encrypted channel ensures the confidentiality of the data, thereby preventing sensitive information from being intercepted and monitored by third parties. The environment and process for constructing the dataset are illustrated in Figure 6. A laptop was used to create a mobile hotspot, simulating a local area network environment. In this setup, the laptop acts as the gateway listening device, and all traffic generated by smartphones connected to the mobile hotspot passes through the laptop. Wireshark software was used to capture all the traffic in this scenario.

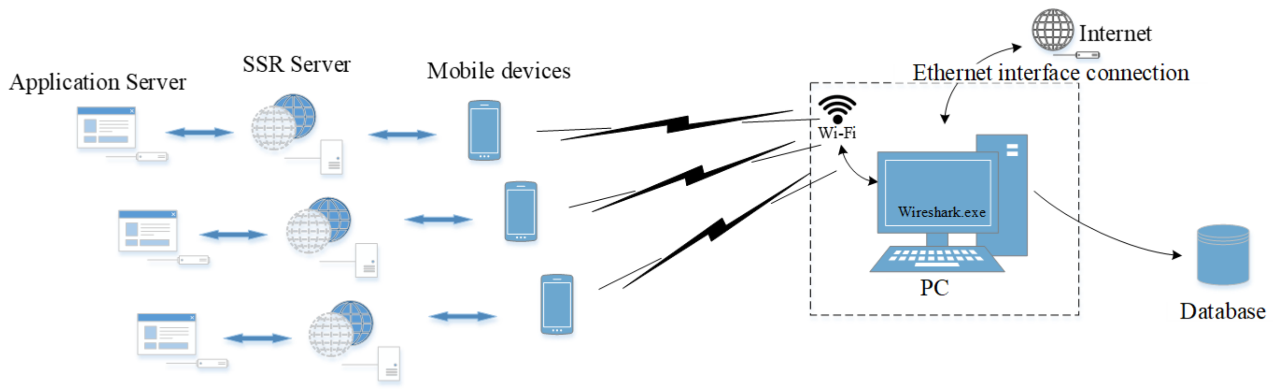


Figure 6. Creation process of the mobile application traffic dataset under an encrypted channel.

The specific information about the experimental devices used in the study is presented in Table 3. Each smartphone device used in the experiments had the SSR client installed (version 3.6.0) and was configured in global proxy mode. It means that all the traffic generated by the Android devices was forwarded through the SSR server. In addition, eight popular mobile applications were selected for the experiments, including IQiyi, Douyin, JD, Toutiao, NetEase Cloud Music, Instagram, Twitter and YouTube.

The data collection process involved manually operating each selected mobile application on the smartphone devices and capturing approximately 30 minutes of traffic data for each application. This process was repeated for a total duration of about 4 hours. During the data collection, only one selected application was installed on each device, and no other applications were running in the background. The internet access permissions of other applications on the Android devices were restricted to minimize background network traffic. This ensured that the captured traffic data represented the pure traffic of a specific mobile application under the encrypted channel, facilitating subsequent experiments.

Table 3. Specific information about experimental devices.

Devices	Devices brand	Devices version	System version
Smartphone 1	vivo	vivo Z6	Android 10.0
Smartphone 2	Xiaomi	Xiaomi 8	Android 8.0
Smartphone 3	OPPO	OPPO A9	Android 8.1
Notebook computer	Dell	Inspiron 5501-R1625D	Windows 10

On the laptop side, Wireshark software (version 3.0.14) was installed to monitor and capture the traffic data packets of each mobile application. During the capturing process, measures were taken to filter out damaged and retransmitted packets. The captured mobile application traffic was saved in the .pcap file format using Wireshark and labeled with the application source. The number of traffic data packets collected for each mobile application is presented in Table 4.

Table 4. Packet count statistics for each application.

APP	Packet quantity	Proportion
IQiyi	74,659	9.96%
Douyin	129,155	17.23%
JD	95,083	12.69%
Toutiao	112,520	15.01%
NetEase Cloud Music	82,535	11.01%
Instagram	79,462	10.61%
Twitter	79,222	10.57%
YouTube	96,843	12.92%
Total	749,479	100.0%

5.3. Evaluation indicators

In this paper, we use Accuracy, Precision, Recall, F1-Score and Confusion Matrix to evaluate classification models. The accuracy rate describes the overall performance of the classifier, and the accuracy rate evaluates the classification effect of each category in the classification problem. The F1-Score is used to evaluate the performance of the classifier. The confusion matrix can be used to observe the classification of each category in detail. TP means that a positive sample is correctly identified as a positive sample. TN means negative samples are correctly identified as negative samples. FP means that negative samples are misidentified as positive samples. FN means that positive samples are misidentified as negative samples.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.2}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.4}$$

$$F1 - \text{Score} = \frac{2\ \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.5}$$

5.4. Experimental parameters

In this paper, we adopt a combined approach of various deep learning models. During the training and testing validation processes, issues related to the selection of hyperparameters such as burst threshold setting, number of bytes for packet truncation, convolutional kernel sizes and Dropout values were addressed. Comparative experiments were designed to determine optimal parameters for these factors.

5.4.1. Setting the burst threshold

In this study, the Burst-flow method is employed to process the encrypted mobile application traffic dataset. The traffic data is discretized into burst-form network traffic blocks based on a predefined burst threshold. If the time interval between two data packets exceeds the burst threshold, they are separated into two different bursts. This process helps to prepare the burst-form traffic for further feature extraction. Previous research by Falaki et al. [27] observed that most data packets on smart mobile devices are sent or received within 4.5 s of the previous packet. Similarly, Taylor et al. [13] suggested that setting the burst threshold to 1 s slightly increases the number of bursts in the network, while providing near real-time performance.

In this study, by observing the arrival time intervals of the captured encrypted mobile application traffic data, we found that the majority of data packets have arrival time intervals ranging from 0.005 to 0.5 s. This indicates that network performance (bandwidth and latency) has improved compared to earlier research. Therefore, this study re-explores the setting of the burst threshold in the traffic processing stage. In the experiments, burst thresholds of 0.005 s, 0.05 s, 0.5 s and 1 s are tested, and the impact of different burst thresholds on classification accuracy is evaluated based on experimental results. The number of data samples obtained for each application under different burst threshold settings is shown in Table 5. Additionally, the classification accuracy achieved by the dataset under different burst threshold settings is illustrated in Figure 7.

From the experimental results, it can be observed that the burst threshold has a significant impact on the final model's classification performance. When the burst threshold is set to 0.5 s, the model achieves the highest accuracy in classifying the dataset. Therefore, we choose a burst threshold of 0.5 s as the standard for flow separation in the traffic processing.

Table 5. Packet count statistics for each application.

APP	Sample quantity			
	1 s	0.5 s	0.05 s	0.005 s
IQiyi	2122	2421	3288	7784
Douyin	1352	1517	1680	7520
JD	2458	2651	3111	6018
Toutiao	1120	1166	1269	1567
NetEase Cloud Music	1805	1921	2214	13,561
Instagram	1088	1566	1674	3210
Twitter	960	1025	1136	2876
YouTube	1780	1997	2395	5638
Total	12,685	14,264	16,767	48,174

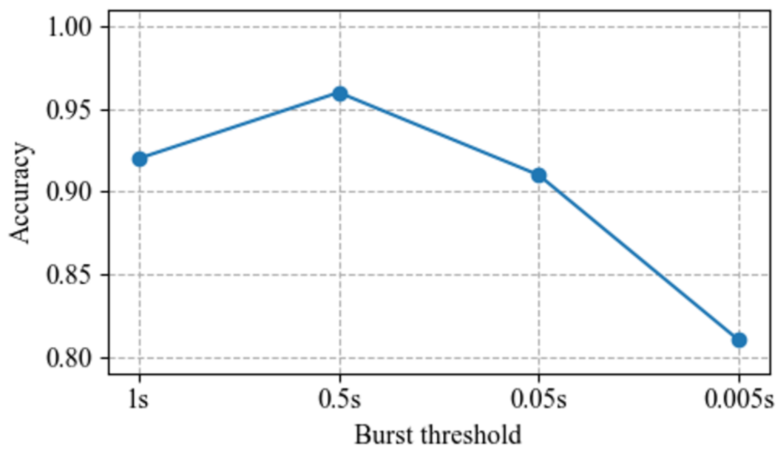


Figure 7. Classification accuracy under different burst thresholds.

5.4.2. Packet byte truncation

To investigate the impact of truncating the original byte length of packets on the classification task of mobile application traffic under encrypted channels, different byte lengths of packets were truncated as input features for the classification model. Figure 8 presents the classification accuracy under different packet lengths.

From Figure 8, it can be observed that as the original byte length increases from 100 to 700, the classification accuracy of the model significantly improves. This indicates that increasing the original byte length can provide more feature information. However, from the curve's change, it can be seen that once the original byte length reaches a certain threshold, the provided feature information tends to saturate. Therefore, we select truncating 784 bytes for further experiments. This approach can provide sufficient information while improving data processing efficiency.

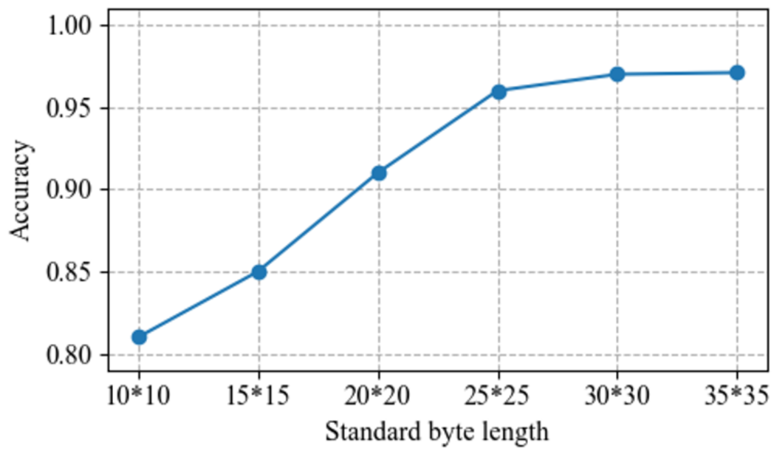


Figure 8. Classification accuracy under different packet lengths.

5.4.3. Convolutional kernel size

The size of the convolutional kernel can affect the ability to extract features. Using a kernel that is too large or too small can have an impact on feature extraction performance. To investigate the impact of convolutional kernel size on the performance of the classification model for mobile application traffic under encrypted channels, this section conducts comparative experiments using five different kernel sizes. The experimental results are shown in Figure 9.

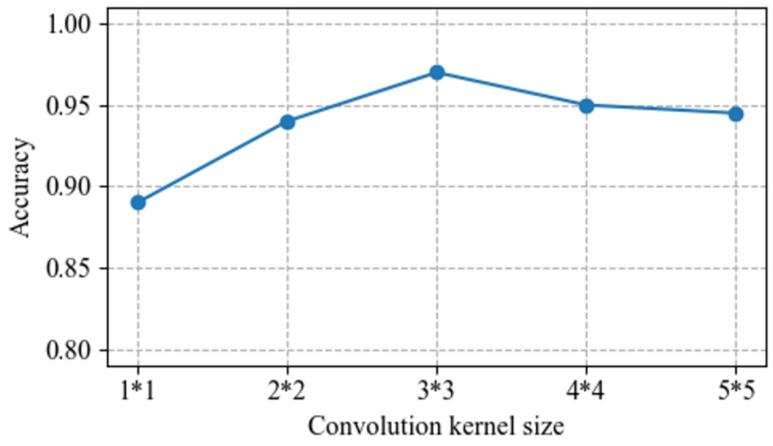


Figure 9. Effect of convolution kernel size on classification performance.

From Figure 9, When using different kernel sizes, it was observed that they indeed affect the network's classification performance. Specifically, as the kernel size gradually increased from smaller values to 3 × 3, the model's classification accuracy improved consistently. However, once the size exceeded 3 × 3, the accuracy of the model started to decline. This fluctuation indicates a complex non-linear relationship between the kernel size and model performance. Selecting an appropriate kernel size involves balancing the capability to extract features with maintaining model simplicity. Consequently, the research findings suggest that a 3 × 3 kernel size is the optimal choice, allowing for high performance while effectively managing model complexity.

5.4.4. The value of dropout

As mentioned in Section 4.2, to address the issue of overfitting during model training, the Dropout method is used to randomly drop out a fraction of neurons during the training process. This is a regularization technique that randomly drops a certain proportion of neurons during the training process to reduce interdependency among neurons and prevent the model from overfitting to the training data. In this process, a portion of neurons is randomly deactivated during each training iteration, setting their output to zero with a certain probability. Consequently, each neuron learns to become more robust during training, not relying heavily on the presence of specific other neurons. This method helps improve the model's generalization by forcing the network to learn more robust features rather than relying on particular neurons, thereby reducing overfitting to the training data. In order to determine the appropriate value for Dropout, this section conducts comparative experiments with different Dropout values (0.3, 0.4, 0.5 and 0.6) in the model. The experimental results, as shown in Figure 10, indicate that the model achieves the highest classification accuracy when the Dropout value is set to 0.5. Therefore, in this study, the Dropout parameter is set to 0.5 for the model.

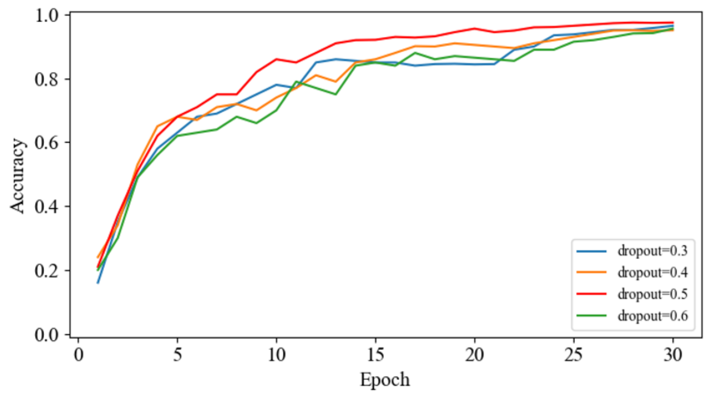
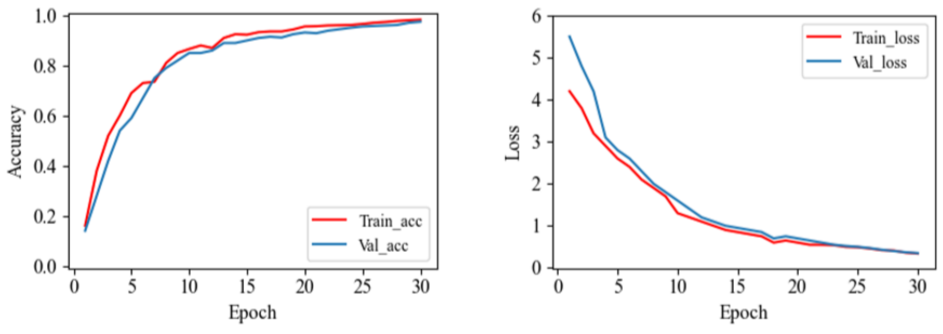


Figure 10. Effect of different dropout settings on model performance.

Given that the model does not suffer from overfitting or underfitting, it can achieve good classification performance on encrypted mobile application traffic. In this case, the accuracy and loss curve of the model training are shown in Figure 11. Regularization and the integration of Dropout algorithm help alleviate the issue of overfitting and improve the model's generalization ability. From Figure 11, it can be observed that the optimized model continuously improves its accuracy on the validation set as the neural network is trained step by step. After 30 epochs of training, the model achieves an accuracy of 98.12% on the training set and 97.56% on the validation set.



(a) Accuracy using Dropout and L2 regularization (b) Loss function using Dropout and L2 regularization

Figure 11. Performance of the model after optimization.

5.5. *Experimental results and analysis*

The experiments in this section include performance evaluation of classification, evaluation of resistance to background traffic interference, ablation experiments and comparative experiments. The specific experimental procedures and results analysis are described as follows.

5.5.1. Classification performance evaluation

After selecting the experimental parameters in Section 5.4, to evaluate the classification performance of the model on mobile application traffic under encryption, this section conducted experiments on the experimental dataset. First, the classification performance of the proposed model without introducing background traffic interference was validated. The experimental results are shown in Figure 12 and Table 6.

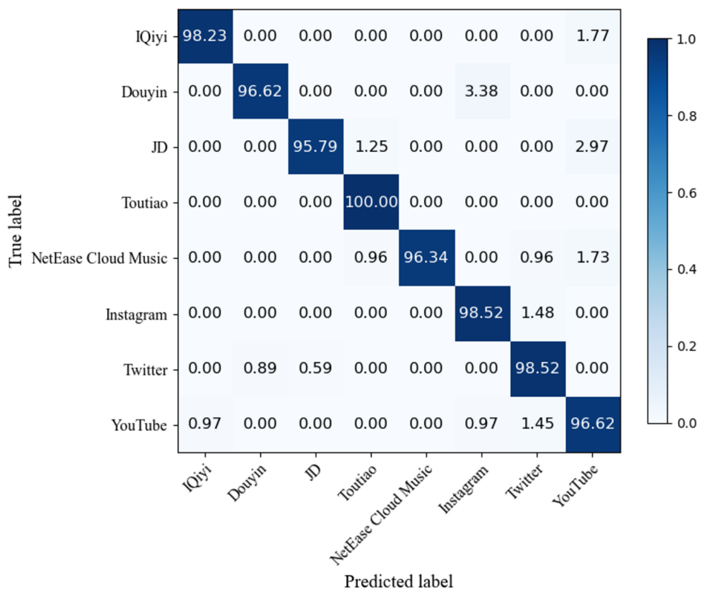


Figure 12. Confusion matrix distribution diagram.

Table 6. The performance of the model on the evaluation index.

APP	Precision	Recall	F1-Score
IQiyi	0.990	0.982	0.986
Douyin	0.991	0.962	0.978
JD	0.994	0.957	0.975
Toutiao	0.977	1.000	0.987
NetEase Cloud Music	1.000	0.963	0.981
Instagram	0.957	0.985	0.971
Twitter	0.962	0.985	0.973
YouTube	0.937	0.966	0.951

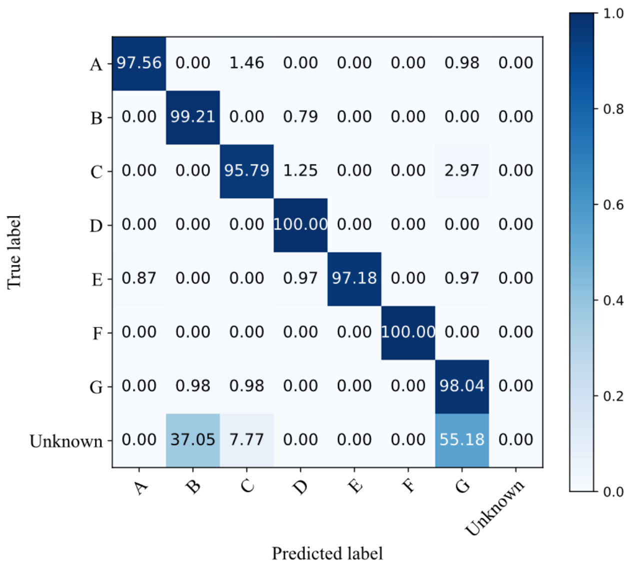
Figure 12 represents the confusion matrix of the classification accuracy for mobile application traffic under encryption. The closer the elements on the diagonal of the matrix are to 100% and the closer the other elements are to 0, the better the classification performance of the algorithm model for mobile application traffic. Table 6 presents the precision, recall and F1-score of the model for the identification of the 8 mobile applications. By considering the experimental results from Figure 12 and Table 6, it can be observed that the model achieves the highest classification accuracy for the 'Toutiao' application under encryption, and the recognition accuracy for all 8 applications is above 95%. Therefore, the designed classification model in this study demonstrates good recognition performance without the introduction of background traffic interference. The next section will evaluate the performance of the proposed method in resisting background traffic interference by introducing such interference.

5.5.2. Evaluation of resistance to background traffic interference performance

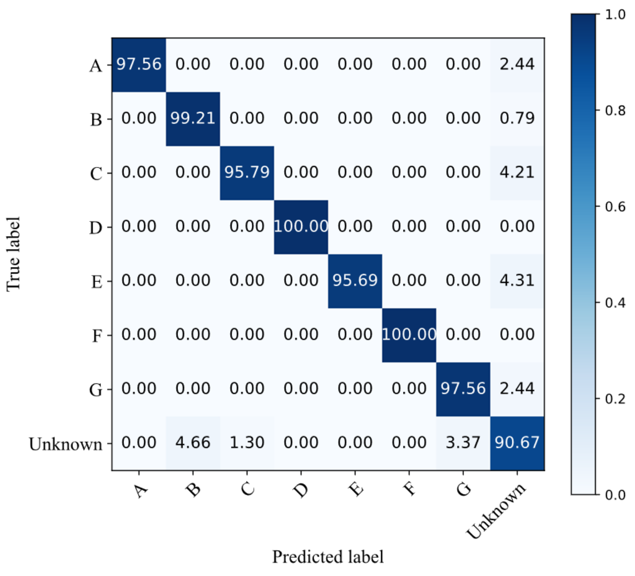
To evaluate the performance of the background traffic filtering module, multiple experiments were conducted in this section. The experiments considered all eight classes of applications in the collected dataset and created two separate subsets: A training set and a test set. In each experiment, one application was selected and separated as an unknown class sample solely for the test dataset, labeled as 'Unknown'. The remaining seven applications were used as known class samples in the training set, labeled as 'A', 'B', 'C', 'D', 'E', 'F' and 'G'. The model was trained using the training set, and after training, the test set samples from the eight classes of applications were classified for testing the model. To test the performance of the model in filtering background traffic, Figure 13 shows the confusion matrices before and after introducing the background traffic filtering module.

From Figure 13(a), it can be observed that before the introduction of the background traffic filtering module, the model achieved high accuracy in classifying the known classes 'A', 'B', 'C', 'D', 'E', 'F' and 'G'. However, the samples of the unknown class 'Unknown' were misclassified by the classifier into classes 'B', 'C' and 'G', indicating that the model had weak classification ability for unknown classes and some generalization performance issues.

From Figure 13(b), it can be seen that after introducing the background traffic filtering module, the proposed model was able to recognize and filter out 90.67% of the unknown class samples. This improvement significantly enhanced the model's classification ability and generalization performance for unknown classes. After filtering, the model only focused on more reliable samples, allowing for more accurate classification of unknown classes and avoiding misclassification of unknown class samples into known classes.



(a) Before introducing modules module



(b) After introducing modules module

Figure 13. Confusion matrix before and after the background traffic filtering module is introduced.

However, from the result graph, it can be observed that the model has a misclassification rate of 14.19%. Among them, 2.44% of class ‘A’ samples, 0.79% of class ‘B’ samples, 4.21% of class ‘C’ samples, 4.31% of class ‘E’ samples and 2.44% of class ‘G’ samples were misclassified by the classifier as unknown class samples. According to the analysis, the main reason for the misclassifications by the classifier is the insufficient accuracy of the confidence threshold setting. To achieve the optimal balance between the background traffic detection rate and the misclassification rate, this section further analyzed the confidence distribution of each sample class, as shown in Figure 14.

Figure 14(a)–(g) represent the confidence distributions of known class samples, while (h) represents the confidence distribution of unknown class samples, i.e., background traffic samples. Analyzing the confidence distribution graph, it can be observed that the majority of known class samples have confidence values distributed in the higher probability range. Specifically, most of the known class samples have confidence values of 0.97 and above. This indicates that the model has high confidence and accuracy in classifying these known class samples. On the other hand, the confidence distribution of unknown background traffic samples exhibits a scattered trend, with lower confidence values, mostly located in the lower probability range. To achieve the optimal balance between the background traffic detection rate and the misclassification rate, this section conducted comparative experiments using multiple confidence values (0.95, 0.955, 0.96, 0.965, 0.97, 0.975 and 0.98) as filtering thresholds. The experimental results are shown in Figure 15.

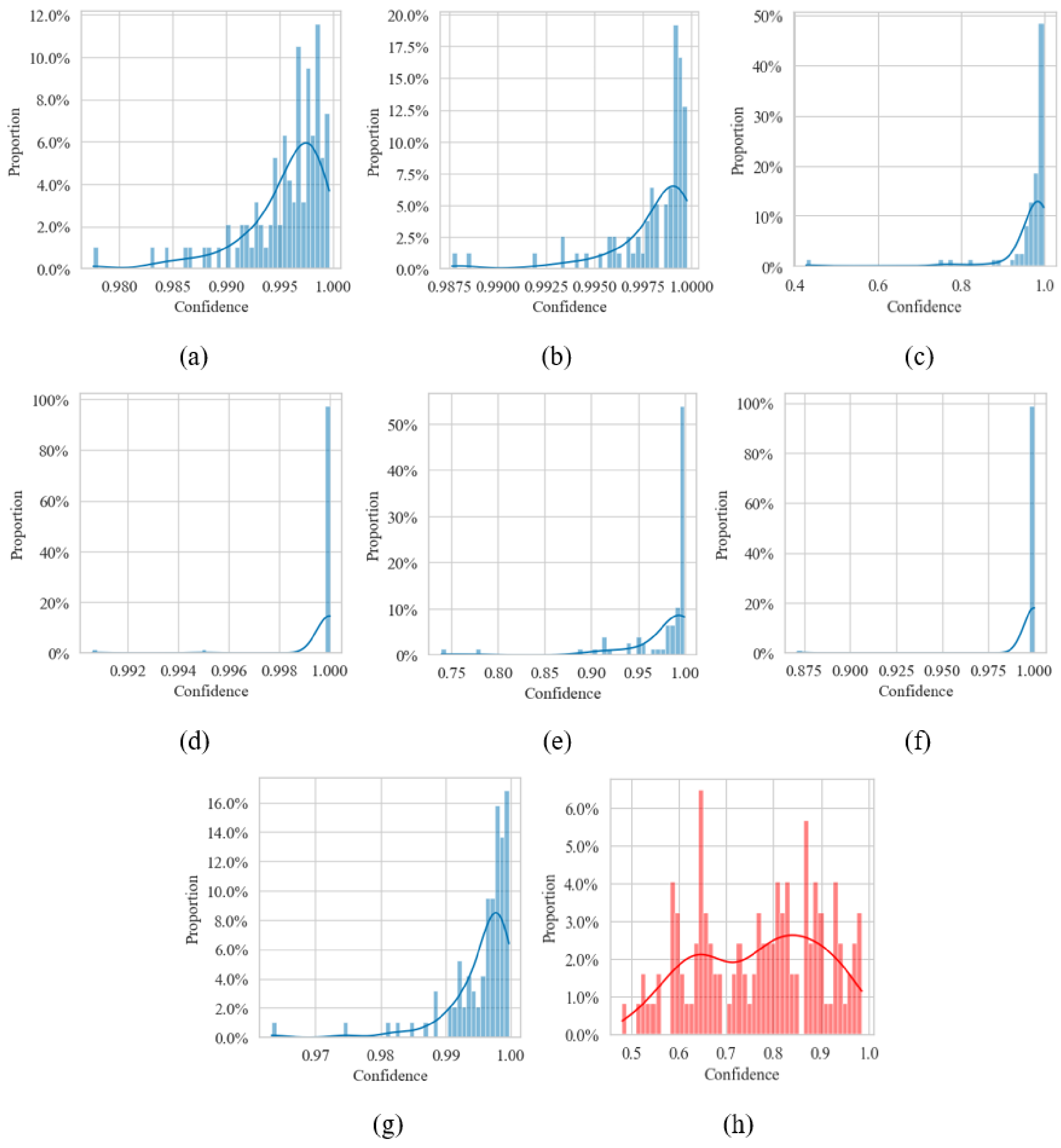


Figure 14. Confidence distribution map of various samples.

According to the experimental results in Figure 15, as the confidence threshold increases, the misclassification rate of known traffic gradually increases, while the background traffic detection rate gradually increases. Therefore, selecting an appropriate confidence threshold requires finding a balance between the misclassification rate of known traffic and the background traffic detection rate.

When the confidence threshold increases from 0.96 to 0.965, the misclassification rate of known traffic slightly increases, but the background traffic detection rate significantly improves. However, when the confidence threshold continues to increase to 0.97, the misclassification rate of known traffic sharply increases, while the growth of the background traffic detection rate slows down. This indicates that at this threshold, too many target mobile application traffic samples are misclassified as unknown background traffic, leading to a significant increase in the misclassification rate. Therefore, 0.965 is chosen as the final confidence threshold to achieve the best balance.

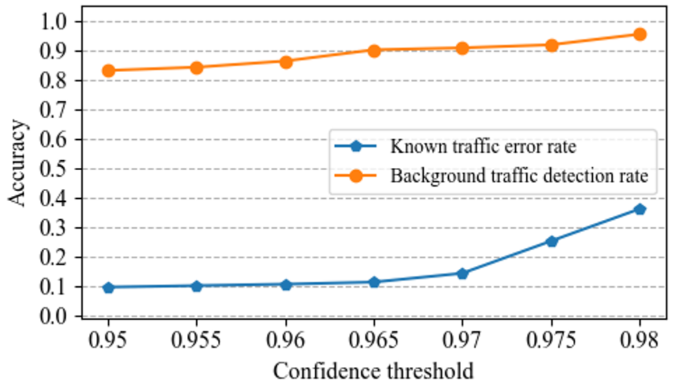


Figure 15. Confidence threshold comparison experiment.

It is important to note that when setting the confidence threshold, specific application scenarios and requirements should be taken into account. If a higher background traffic detection rate is required, the confidence threshold can be appropriately increased to increase the detection rate. If a lower misclassification rate of known class mobile application traffic samples is desired, the confidence threshold can be appropriately lowered to reduce the misclassification rate.

5.5.3. Ablation experiments

The proposed method includes multiple components that contribute to the improvement of the classification performance. In order to evaluate the contributions of each component to the final recognition performance of the model, ablation experiments were conducted on both the original dataset and the dataset with background traffic. The compared models included independent CNN model, GRU model, CNN and GRU combined model and the model with the additional background traffic filtering gain module. The results of the comparison showed improvements in various classification performance metrics when the models with performance gain modules were introduced.

According to Tables 7 and 8:

1) Under the original dataset, the CNN and GRU models have relatively lower classification performance but exhibit some level of classification ability. The combination of CNN and GRU effectively utilizes their respective advantages. CNN can extract local features through operations such as convolution and pooling, while GRU, with its recurrent neural network structure, can model and classify long-term dependencies. This feature extraction and combination approach leads to improved model performance.

2) Under the dataset with background traffic, the CNN, GRU and CNN+GRU models exhibit relatively poorer classification performance. This indicates that these models have weaker resistance to interference when dealing with datasets containing background traffic. The addition of the background traffic filtering module significantly improves the performance of the CNN+GRU+background traffic filtering model compared to the other models. This suggests that the model effectively reduces the impact of background traffic on classification performance, enhances its resistance to interference and improves robustness when dealing with datasets containing background traffic.

Table 7. Results of evaluation indexes of ablation experimental model in the original dataset.

Model	Accuracy	Precision	Recall	F1-Score
CNN	0.924	0.933	0.914	0.925
GRU	0.888	0.896	0.873	0.886
CNN+GRU	**0.975**	**0.976**	**0.976**	**0.975**

Table 8. Results of evaluation indexes of ablation experimental model with background flow dataset.

Model	Accuracy	Precision	Recall	F1-Score
CNN	0.761	0.746	0.782	0.764
GRU	0.724	0.714	0.745	0.727
CNN+GRU	0.812	0.808	0.816	0.812
CNN+GRU+Background flow filtering	**0.954**	**0.959**	**0.961**	**0.961**

5.5.4. Comparative experiments

In order to demonstrate the superiority of the proposed framework, comparative experiments were conducted with existing state-of-the-art frameworks. The baseline methods compared are described as follows:

Reference [19] proposed the CLD-Net model, which utilizes the capability of Convolutional Neural Networks (CNNs) to classify image categories. It learns and classifies preprocessed grayscale images of original flows and further enhances the model's classification ability using Long Short-Term Memory (LSTM) networks for temporal sequence data. Experimental results show that the model can differentiate between VPN and non-VPN network traffic on the publicly available ISCX dataset and accurately identify specific traffic types (chat, audio, or file) of Facebook and Skype applications.

Reference [28] proposed a network traffic classification model called ABL-TC, which introduces attention mechanism to improve the LSTM model. Based on the experimental results on the publicly available ISCX VPN-nonVPN and Tor-nonTor datasets, the model performs well in 18 classification tasks involving regular encryption, VPN, and Tor traffic, with average precision, recall and F1-score all exceeding 99.6%.

Reference [29] introduces the Transformer model, where we employ 32-dimensional embedding vectors to represent each input token. The model incorporates 6 attention heads to concurrently capture correlations within different feature subspaces. Additionally, the internal feedforward neural network comprises hidden layers of 32 dimensions. These parameter selections aim to strike a balance between model capacity and computational efficiency.

Although existing works have shown good recognition performance, none of the three mentioned papers considered the presence of background traffic generated by unknown applications in real-world scenarios. Existing works focus on classifying network traffic in closed testing environments where the training and testing sets contain the same traffic classes. This results in significantly reduced classification accuracy when faced with background traffic interference.

In this section, the above three methods were first applied to the laboratory-collected pure original dataset for classification experiments of mobile application traffic under encrypted channels. A comparison was made with the proposed method, and the classification performance of different methods for each application traffic is shown in Table 9.

Table 9. The effect of different methods on traffic classification for different applications.

APP	Reference [19]				Reference [28]			
	A	P	R	F1	A	P	R	F1
IQiyi	0.911	0.938	0.911	0.924	0.936	0.948	0.936	0.942
Douyin	0.948	0.993	0.948	0.970	0.896	0.995	0.896	0.943
JD	0.958	1.000	0.958	0.979	0.926	0.957	0.926	0.941
Toutiao	0.99	0.915	0.990	0.951	0.975	0.940	0.975	0.957
NetEase Cloud Music	0.963	0.976	0.963	0.970	0.928	0.944	0.928	0.936
Instagram	0.993	0.944	0.993	0.968	0.990	0.911	0.990	0.949
Twitter	0.965	0.976	0.965	0.971	0.910	0.938	0.910	0.924
YouTube	0.923	0.912	0.923	0.912	0.959	0.886	0.959	0.921
Macro average	0.955	0.957	0.955	0.956	0.939	0.940	0.939	0.939
APP	Rerference [29]				Our proposed			
	A	P	R	F1	A	P	R	F1
IQiyi	0.969	0.968	0.993	0.971	0.982	0.990	0.982	0.986
Douyin	0.972	0.987	0.977	0.976	0.966	0.991	0.962	0.978
JD	0.989	0.969	0.957	0.982	0.957	0.994	0.957	0.975
Toutiao	0.966	0.991	0.965	0.976	1.000	0.977	1.000	0.987
NetEase Cloud Music	0.986	0.971	0.989	0.984	0.963	1.000	0.963	0.981
Instagram	0.991	0.969	0.971	0.986	0.985	0.957	0.985	0.971
Twitter	0.986	0.968	0.961	0.949	0.985	0.962	0.985	0.973
YouTube	0.925	0.963	0.976	0.963	0.966	0.937	0.966	0.951
Macro average	0.973	0.973	0.974	0.973	**0.975**	**0.976**	**0.976**	**0.975**

The experimental results show that, in the absence of background traffic interference, the proposed method performs well in classifying mobile application traffic under encrypted channels. In terms of classification accuracy, Reference [19] achieves an accuracy of 0.955, and Reference [28] achieves an accuracy of 0.939, while the proposed method achieves an accuracy of 0.975, surpassing the other two methods by over 2%. The proposed method also outperforms the other two methods in terms of precision, recall and F1 score, with an improvement of over 2% in each metric. In conclusion, the proposed method demonstrates superior performance overall and exhibits good classification ability for mobile application traffic under encrypted channels.

Considering that in real identification scenarios, the target and background traffic often coexist, the background traffic can interfere with the classifier. To highlight the advantages of the proposed method in resisting background traffic interference, in addition to the comparative experiments conducted on the laboratory-collected pure original dataset for encrypted channel mobile application traffic classification, this section constructs a dataset that includes background traffic according to the method described earlier for evaluating the performance against background traffic interference. The comparison experiments on the dataset were performed, comparing with three methods , respectively, from references [19, 28, 29]. The performance of different methods on the experimental dataset is shown in Table 10.

The experimental results indicate that when the generated dataset containing background traffic is used for classifying mobile application traffic under encrypted channels, the three compared models show a significant decrease in accuracy, with a reduction of 14.2%, 15.7% and 13.1%, respectively. This is because the compared methods only consider the same dataset for training and evaluating model performance, and they perform well when tested in an ideal environment. However, when unknown data is present, their classification accuracy is affected by the presence of background traffic. Therefore, when these models are deployed in real network environments, their ability to resist background traffic interference is significantly weaker than the proposed method in this study. In summary, the proposed method in this research exhibits higher robustness and practicality in classifying mobile application traffic under encrypted channels.

Table 10. Performance of different methods on experimental datasets.

Reference	Model	Original dataset accuracy	Including background traffic dataset accuracy
Reference [19]	CNN+LSTM	0.955	0.813
Reference [28]	Attention-based LSTM	0.939	0.782
Reference [29]	Transformer	0.973	0.842
This paper	CNN+GRU+Background flow filtering	**0.975**	**0.954**

6. Conclusions

In this work, we propose a novel method for mobile application recognition in encrypted channels. We process the traffic through the encrypted channel using a slicing method and have devised a neural network model that combines CNN and GRU. This model leverages CNN for extracting spatial features from network traffic and employs GRU for modeling the temporal sequences. This approach effectively characterizes the spatiotemporal features of mobile application traffic over encrypted channels. It enables the extraction of features from mobile application traffic under encrypted channels and employs comprehensive analysis based on probabilistic outputs to filter out low-confidence background traffic interference. Relevant experiments demonstrate that the proposed method exhibits a high recognition accuracy and robust interference resistance. This approach presents a novel perspective and method for addressing the challenge of identifying mobile application traffic under encrypted channels, offering significant practical application potential.

Although this paper does not compare with multimodal models, it does compare with other models (CNN, GRU, CNN+GRU). Additionally, we focus on evaluating the performance against background interference. Within the current field, it is noted that many advanced techniques often overlook interference issues, while our work aims to address this research gap. At this stage, we have chosen to concentrate the paper's emphasis on the model performance comparison before and after introducing the anti-interference module. We believe this decision helps highlight the specific contribution of our research. Nonetheless, in future work, we plan to conduct further comparisons with state-of-the-art techniques, such as multimodal networks, to ensure readers understand the significance of our current design. Moreover, the work conducted in this paper was based on a dataset constructed in a laboratory environment. With the continuous growth of the internet environment and the number of applications, these data have certain limitations. In the future, expanding the dataset by incorporating more devices and a wider range of application data could lead to better improvements in traffic recognition solutions on a larger scale. Finally, leveraging Explainable Artificial Intelligence (XAI) techniques is also a potential avenue for further research, as discussed by Nascita et al., to explicate and strengthen our proposed method [30]. Through XAI technology, we can delve into understanding the decision-making logic of the model in identifying applications within encrypted channels, thereby enhancing the method's transparency and interpretability. This not only enhances the performance and robustness of our method but also provides deeper insights and avenues for improvement within the realm of identifying mobile applications in encrypted channels. Future research could further explore identification techniques in encrypted scenarios, methods to enhance security and customized identification models tailored to different application types, thereby further propelling the development and innovation in this field.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grants Nos. 61931004 and 62072250), the National Key Research and Development Program of China (Grants No. 2021QY0700), Key Laboratory of Intelligent Support Technology In Complex Environment.

Conflicts of interest

The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. S. Kumar, S. Indu, G. S. Walia, Smartphone traffic analysis: a contemporary survey of the state-of-the-art, in *Proceedings of the Sixth International Conference on Mathematics and Computing*, Springer, **1262** (2021), 325–343. https://doi.org/10.1007/978-981-15-8061-1_26

2. J. Wang, H. Han, H. Li, S. He, P. K. Sharma, L. Chen, Multiple strategies differential privacy on sparse tensor factorization for network traffic analysis in 5G, *IEEE Trans. Ind. Inform.*, **18** (2022), 1939–1948. https://doi.org/10.1109/TII.2021.3082576

3. J. Wang, Y. Yang, T. Wang, R. Sherratt, J. Zhang, Big data service architecture: a survey, *J. Internet Technol.*, **21** (2020), 393–405.

4. P. Wang, X. Chen, F. Ye, Z. Sun, A survey of techniques for mobile service encrypted traffic classification using deep learning, *IEEE Access*, **7** (2019), 54024–54033. https://doi.org/10.1109/ACCESS.2019.2912896

5. H. Yan, H. Li, M. Xiao, R. Dai, X. Zheng, X. Zhao, et al., PGSM-DPI: precisely guided signature matching of deep packet inspection for traffic analysis, in *2019 IEEE Global Communications Conference (GLOBECOM)*, IEEE, (2019), 1–6. https://doi.org/10.1109/GLOBECOM38437.2019.9013941

6. M. S. Sheikh, Y. Peng, Procedures, criteria, and machine learning techniques for network traffic classification: a survey, *IEEE Access*, **10** (2022), 61135–61158. https://doi.org/10.1109/ACCESS.2022.3181135

7. A. Agrawal, A. Bhatia, A. Bahuguna, K. Tiwari, K. Haribabu, D. Vishwakarma, et al., A survey on analyzing encrypted network traffic of mobile devices, *Int. J. Inf. Secur.*, **21** (2022), 873–915. https://doi.org/10.1007/s10207-022-00581-y

8. G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapé, Mobile encrypted traffic classification using deep learning: experimental evaluation, lessons learned, and challenges, *IEEE Trans. Netw. Serv. Manage.*, **16** (2019), 445–458. https://doi.org/10.1109/TNSM.2019.2899085

9. M. Wang, K. Zheng, D. Luo, Y. Yang, X. Wang, An encrypted traffic classification framework based on Convolutional Neural Networks and stacked autoencoders, in *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, IEEE, (2022), 634–641. https://doi.org/10.1109/ICCC51575.2020.9344978

10. T. Shapira, Y. Shavitt, FlowPic: a generic representation for encrypted traffic classification and applications identification, *IEEE Trans. Netw. Serv. Manage.*, **18** (2021), 1218–1232. https://doi.org/10.1109/TNSM.2021.3071441

11. Z. Ahmad, A. S. Khan, C. W. Shiang, J. Abdullah, F. Ahmad, Network intrusion detection system: a systematic study of machine learning and deep learning approaches, *Trans. Emerging Telecommun. Technol.*, **32** (2021), e4150. https://doi.org/10.1002/ett.4150

12. H. F. Alan, J. Kaur, Can Android applications be identified using only TCP/IP headers of their launch time traffic? in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, ACM, (2016), 61–66. https://doi.org/10.1145/2939918.2939929

13. V. F. Taylor, R. Spolaor, M. Conti, I. Martinovic, Appscanner: automatic fingerprinting of smartphone apps from encrypted network traffic, in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, (2016), 439–454. https://doi.org/10.1109/EuroSP.2016.40

14. K. Park, H. Kim, Encryption is not enough: inferring user activities on KakaoTalk with traffic analysis, in *Information Security Applications*, Springer, **9503** (2016), 254–265. https://doi.org/10.1007/978-3-319-31875-2_21

15. B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, et al., Eavesdropping on Fine-Grained user activities within smartphone apps over encrypted network traffic, in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, USENIX Association, 2016.

16. S. E. Coull, K. P. Dyer, Traffic analysis of encrypted messaging services: apple iMessage and beyond, *ACM SIGCOMM Comput. Commun. Rev.*, **44** (2014), 5–11. https://doi.org/10.1145/2677046.2677048

17. M. Conti, L. V. Mancini, R. Spolaor, N. V. Verde, Analyzing android encrypted network traffic to identify user actions, *IEEE Trans. Inf. Forensics Secur.*, **11** (2015), 114–125. https://doi.org/10.1109/TIFS.2015.2478741

18. Z. Wang, The applications of deep learning on traffic identification, *BlackHat USA*, **24** (2015), 1–10.

19. X. Hu, C. Gu, F. Wei, Cld-net: a network combining CNN and LSTM for internet encrypted traffic classification, *Secur. Commun. Netw.*, **2021** (2021), 5518460. https://doi.org/10.1155/2021/5518460

20. G. Aceto, D. Ciuonzo, A. Montieri, A. Pescapè, MIMETIC: mobile encrypted traffic classification using multimodal deep learning, *Comput. Netw.*, **165** (2019), 106944. https://doi.org/10.1016/j.comnet.2019.106944

21. W. Wang, M. Zhu, J. Wang, X. Zeng, Z. Yang, End-to-end encrypted traffic classification with one-dimensional convolution neural networks, in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, IEEE, (2017), 43–48. https://doi.org/10.1109/ISI.2017.8004872

22. A. A. M. Al-Saffar, H. Tao, M. A. Talab, Review of deep convolution neural network in image classification, in *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, IEEE, (2017), 26–31. https://doi.org/10.1109/ICRAMET.2017.8253139

23. W. Wang, M. Zhu, X. Zeng, X. Ye, Y. Sheng, Malware traffic classification using Convolutional Neural Network for representation learning, in *2017 International Conference on Information Networking (ICOIN)*, IEEE, (2017), 712–717. https://doi.org/10.1109/ICOIN.2017.7899588

24. A. F. Agarap, Deep learning using rectified linear units (ReLU), preprint, arXiv:1803.08375.

25. A. Labach, H. Salehinejad, S. Valaee, Survey of dropout methods for deep neural networks, preprint, arXiv:1904.13310.

26. S. Yang, X. Yu, Y. Zhou, LSTM and GRU neural network performance comparison study: taking yelp review dataset as an example, in *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, IEEE, (2020), 98–101. https://doi.org/10.1109/IWECAI50956.2020.00027

27. H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, D. Estrin, A first look at traffic on smartphones, in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ACM, (2010), 281–287. https://doi.org/10.1145/1879141.1879176

28. W. Wei, H. Gu, W. Deng, Z. Xiao, X. Ren, ABL-TC: a lightweight design for network traffic classification empowered by deep learning, *Neurocomputing*, **489** (2022), 333–344. https://doi.org/10.1016/j.neucom.2022.03.007

29. Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, et al., Transformers in time series: a survey, preprint, arXiv:2202.07125.

30. A. Nascita, A. Montieri, G. Aceto, D. Ciuonzo, V. Persico, A. Pescapé, Improving performance, reliability, and feasibility in multimodal multitask traffic classification with XAI, *IEEE Trans. Netw. Serv. Manage.*, **20** (2023), 1267–1289. https://doi.org/10.1109/TNSM.2023.3246794