Encrypted traffic classification: the QUIC case

Jan Luxemburk
FIT CTU & CESNET
Prague, Czech Republic

Karel Hynek
FIT CTU & CESNET
Prague, Czech Republic

Tomáš Čejka
CESNET
Prague, Czech Republic

***Abstract*—The QUIC protocol is a new reliable and secure transport protocol that is an alternative to TLS over TCP. However, compared to TLS, QUIC obfuscates the connection handshake and the server name indication domain, making a simple inspection more challenging. The classification of QUIC traffic has also received less attention than that of TLS. In this work, we present a comprehensive study aiming to explore the challenges of QUIC traffic classification. We selected three models: 1) multi-modal CNN, 2) LighGBM, and 3) IP-based classifier, and evaluated their properties using a large one-month CESNET-QUIC22 dataset with 102 web service labels. The developed classifiers reached up to 88% accuracy and set the new baseline in fine-grained QUIC service classification. Moreover, the real nature of the dataset and its long time span allowed us to collect a number of insights and measure the classifiers' performance in the presence of data drift.**

***Index Terms*—Traffic classification, QUIC, Deep learning, Encrypted traffic, Computer network**

I. Introduction

The QUIC protocol is a new reliable transport protocol standardized two years ago in 2021, which is already receiving enormous traction in adoption. It is implemented in Chrome-based browsers as well as in Firefox, and large companies, such as Google, Meta, or Microsoft, use it to deliver their most popular services. Moreover, we can observe the transition of TCP-based protocols to QUIC—for example, the QUIC-based SMB (Samba) [1] or the decision to use QUIC as the transport protocol in HTTP/3 [2], which suggests it will become the standard option for reliable and secure web communication.

Apart from various features that make QUIC superior to TCP, the most significant for network monitoring and measurement is the enforcement of secure communication—it is not possible to use QUIC without encryption. All QUIC communications are protected with TLS; thus, QUIC payload inspection is impossible without decryption middleboxes. Moreover, the QUIC connection handshake, which reuses the structures of the TLS handshake, such as ClientHello, ServerHello, and Certificate messages, is obfuscated. This obfuscation is performed as AES-GCM encryption using a specific key that the receiver can derive from unencrypted connection parameters [3]. As a result, the monitoring of QUIC traffic is difficult because there are no available plaintext fields related to the communication running inside the encrypted tunnel. TLS is also moving in this direction and is removing available plaintext information; TLS 1.3 moved server certificates into the encrypted part of communication, and there is the TLS Encrypted Client Hello extension draft, which enables clients to encrypt the entire ClientHello messages.

To handle QUIC traffic, detection systems thus need to de-obfuscate (i.e., decrypt) the QUIC handshake messages to obtain the internal TLS messages, which can then be analyzed as the standard TLS. This decryption process must be implemented in the monitoring software and puts an additional load on the infrastructure. The decryption process also must support all the variants (IETF, gQUIC, Facebook's) of the QUIC protocol and must be updated when a new QUIC version is released. Moreover, with the expected increase in QUIC traffic, it might not be feasible for monitoring infrastructure on the service-provider level to handle decryption on such a scale. These negative factors of the handshake decryption create an interesting and practical research challenge *to find novel approaches for effective QUIC traffic monitoring without handshake decryption.*

Recent works [4]–[6] on encrypted traffic classification show promising results in fine-grained TLS service classification. The motivation behind TLS service classification arises from a number of reasons, such as SNI faking [7], Encrypted Client Hello [8], QoS prioritization, or the enforcement of network-usage policies. All of these reasons are also applicable to QUIC; nevertheless, the QUIC handshake obfuscation makes the service classification task much more important because the SNI domains are not available in plaintext.

Despite its importance, the QUIC traffic classification is still nascent, and no previous fine-grained studies of QUIC traffic classification have been published. Existing works are evaluated on small datasets with not enough traffic classes.[1] The properties of QUIC traffic classifiers and their performance on real-world datasets were unknown—but not anymore! In this work, we research fine-grained QUIC service classification. We selected and compared three different classification approaches: 1) a multi-modal convolutional neural network (mm-CNN), 2) a tree ensemble model Light Gradient Boosting Machine (LightGBM), 3) and an IP-based classifier. We evaluate them using the CESNET-QUIC22 dataset, which is a large one-month QUIC network traffic dataset from ISP backbone lines [10]. The CESNET-QUIC22 dataset, which comprises over 153 million flows and has 102 service labels, allows us to create long-term performance evaluations, to reveal the properties of the three studied classification approaches, and

[1]The biggest public dataset in this regard is the UC Davis QUIC dataset [9] with five classes and ∼6500 flows.

978-3-903176-58-4

to compare them. Our results fill the gap in current knowledge about real-world QUIC traffic classification.

The rest of the paper is organized as follows: Section II summarizes related works. Section III provides a brief overview of the used dataset. Section IV describes the designs of the three QUIC classifiers and includes eXplainable AI (XAI) interpretations of their functioning. The main contributions are in Section V, in which the classifiers are evaluated and compared on three consecutive weeks of traffic. Section VI discusses the measured results, and Section VII concludes this article.

II. Related work

Encrypted traffic analysis has a long history of different approaches that utilize non-payload information to infer the type of transmitted content or even the payload of the messages. One of the approaches is IP-based classification, which is widely used in the form of blocklists in network intrusion tasks (e.g., to detect botnet communications). The IP lists are simple and efficient but often suffer from false positives due to list obsolescence or IP address sharing [11]. Nevertheless, according to Hoang et al. [12], IP-based web service classification might work well for popular services since these tend to be hosted on dedicated IP addresses that remain unchanged for a long time.

Other web service classification approaches use side-channel information, such as connection metadata (the number of transmitted bytes and packets, etc.) or sequences of packet lengths and inter-packet times. We can divide these approaches into two categories: 1) coarse-grained methods that aim to classify traffic into high-level categories (such as video streaming, download, or voice), or 2) fine-grained methods that aim to distinguish between particular services (e.g., YouTube, Facebook Messenger, or Netflix).

Fine-grained classification is more challenging because of the need to deal with more classes. Shbair et al. [4] focused on TLS service classification among one thousand services. The authors designed a random forest classifier, evaluated it using a lab-created dataset, and achieved a service identification accuracy of 93.1%. In our previous work [6], we had the same goal but used a real-world dataset from a large ISP network to evaluate a convolutional neural network (CNN) model that achieved 97.04% accuracy in the identification of 192 web services. Other similar works are Yang et al. [5] and Akbari et al. [13], which also explore encrypted traffic classification and evaluate their methods using large real-world datasets; both showing excellent performance even in the fine-grained service classification. Recent work of Malekghaini et al. [14] studied TLS service classification in the context of long-term data drift, with test periods one month or one year apart. Based on the results, the authors proposed several adaptations to their neural network architecture. This improved architecture was then used for the classification of eight QUIC services and achieved 95.6% accuracy.

The QUIC fine-grained classification research falls behind TLS. There are, however, several previous works focusing on coarse-grained classification. Tong et al. [15] proposed one of the first QUIC traffic classifiers. The authors used CNN, which was designed on a lab-created dataset with five traffic categories. This traffic type classifier achieved an accuracy of $\sim$99% and gave us the first proof that QUIC traffic analysis based on side-channel information is possible and can be accurate. Rezaei et al. [9] also studied QUIC traffic classification with CNN and achieved $\sim$98% accuracy. The authors used semi-supervised learning to train the classifier, which allowed training without an extensive dataset. The used lab-created dataset is public and is called the UC Davis QUIC dataset; it contains $\sim$6500 flows divided between five Google services. Akbari et al. [13] then used the same dataset to validate a novel CNN architecture and outperformed Rezaei et al. [9] with $\sim$99% accuracy.

Despite the promising results in coarse-grained classification, the performance and properties of fine-grained QUIC classifiers are under studied. We are not aware of any large-scale QUIC studies with a higher number of traffic classes. The lack of fine-grained approaches can be attributed to the lack of datasets. In 2023, however, we published the CESNET-QUIC22 dataset [10], which we will use to perform the first fine-grained long-term evaluation of multiple classification methods to fill the gap in the current knowledge about QUIC traffic classification.

III. Dataset

The CESNET-QUIC22 dataset contains four consecutive weeks (44th, 45th, 46th, and 47th week in 2022) of QUIC traffic transmitted in the CESNET2 network. CESNET2 is a research and educational network in the Czech Republic that has more than half a million users. The dataset consists of 153 million network flows extended with various features describing the encrypted QUIC communications. The most important features for our work are flow-based statistics (e.g., the number of transferred packets and bytes), packet histograms, and packet metadata sequences (packet sizes, inter-arrival times, and directions). Each flow is annotated with a service tag—a name of the web service that was obtained from the SNI domain in the decrypted QUIC Initial packet (the first packet of the handshake). The dataset has 102 different service labels, which are also organized into groups with the same provider, such as Google services or Facebook services. These two providers represent a significant part of the dataset's services; Google has 27 services in the dataset, Facebook has 9. The used mapping between services and service providers is available at the dataset download page.

More details about the dataset and its collection process (used software, sampling, flow export timeouts, etc.) are described in the original data article [10].

IV. QUIC traffic classifiers

In this section, we introduce the three classification models that we will evaluate on the CESNET-QUIC22 dataset. For each model, we describe the input features, data preprocessing, and provide performance metrics and model interpretations. The experiments presented in this section use the traffic of the first week for model training, and the evaluation is done on a

20 million flow subset of the dataset's remaining three weeks. To report the classification performance, we measure accuracy, macro averaged F1-score, and macro averaged recall.[2] We also calculate provider-level accuracy—a modified version of the accuracy metric, for which the predictions among the same provider are considered to be correct (for services without a defined provider, this works like the standard accuracy metric). The provider labels are used only for measuring this metric, not for training the models.

As the first model, we choose multi-modal CNN, which is a popular architecture often used in previous works [6], [13], [16], [17]. The advantage of this architecture is that it is designed to process multiple types (also called modalities) of network data in parallel. Three modalities discussed in the previous works are packet metadata sequences, flow statistics, and packet payload data.

As the second model, we choose LightGBM, which is a popular gradient-boosted decision tree ensemble model. Yang et al. showed in [5] that standard tree-based models and deep learning can offer comparable performance in encrypted traffic classification tasks. We also want to find out how these approaches compare on the QUIC classification task.

As the third model, we designed an IP-based classifier, which is based on a database of web service observations per destination IP addresses. The success of this approach depends on how stable are the IP addresses used for hosting QUIC web services, i.e., how often those IP addresses change. Related work of [12] focused on this question to evaluate the benefits of Encrypted SNI (now renamed to Encrypted Client Hello) and found out that a surprising number of services are hosted on stable IP addresses.

A. Deep learning classifier

As the multi-modal CNN (mm-CNN) classifier, we adapt the network architecture used in our previous work [6], with modifications described in Section IV-A3.

1) Input: As the multi-modal CNN input, we use packet metadata sequences (PSTATS) and flow statistics (FLOWSTATS), and do not touch packet payload at all. In fact, payload information is not even included in the CESNET-QUIC22 dataset.

a) Packet sequences: The most important input is sequences of packet sizes, directions, and inter-packet times (IPT) for the first 30 packets in each flow. The packet sizes in the dataset are the sizes after transport headers (UDP for QUIC). Note that acknowledgment packets are included in the sequences. Some TLS classification approaches filter them out as zero-payload TCP packets; for QUIC, this is not possible because their payload size is not zero. Packet directions are encoded as ±1. Inter-packet times depend on the location of communicating hosts, their distance, and the network conditions on the path. However, it is still possible to extract relevant information that correlates with user interactions and, for example, with the time required for a server to process the received data.

[2]We would like to note that because each service was sampled in a different ratio during the dataset collection (more prevalent services were subsampled more), the class imbalances in the dataset differ from the true distribution of the original network. As a result, the accuracy and F1-score metrics would be different if the models were deployed there. However, the recall metric is not affected by subsampling and thus translates well into the environment of the original network.

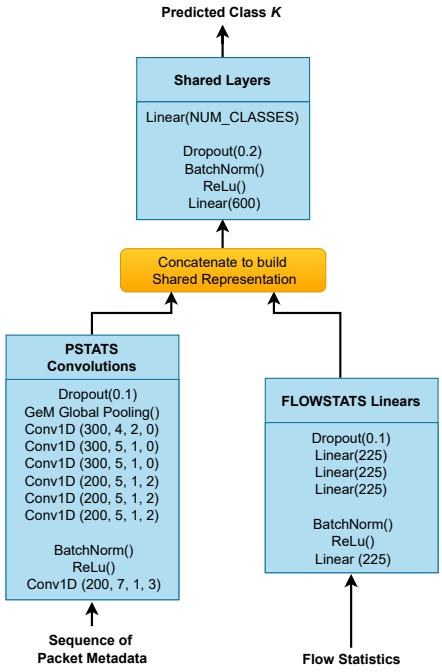


Fig. 1. The multi-modal CNN architecture. The layer parameters represent: `Conv1D(#filters, kernel_size, stride, padding)`, `Linear (#out_features)`, `Dropout(rate)`. Some `BatchNorm` and `ReLu` layers are omitted to save space.

TABLE I
CLASSIFICATION RESULTS OF THE THREE MODELS. THE RESULTS ARE IN THE FORMAT AVERAGE (± STDEV) ACROSS FIVE TRAINING RUNS. THE IP BASELINE MODEL IS DETERMINISTIC, AND THUS NO AVERAGING WAS PERFORMED. PROV. ACC. STANDS FOR PROVIDER-LEVEL ACCURACY.

Classifier	Acc.	Prov. acc.	F1 score	Recall
Multi-modal CNN	77.41% (±3.20%)	97.94% (±0.47%)	82.74% (±1.05%)	80.60% (±1.06%)
LightGBM	80.87% (±0.53%)	98.26% (±0.02%)	80.65% (±0.21%)	78.05% (±0.24%)
IP baseline	68.68%	99.01%	70.02%	70.29%

b) Flow statistics: The second input type is flow statistics, which contain aggregated information about the bidirectional flow. We use all FLOWSTATS fields available in the dataset: the number of transmitted bytes and packets in both directions, the duration of flow, and the flow export reason. FLOWSTATS also include the length of the packet sequence, its duration, and the number of roundtrips—the number of changes in the communication direction (computed from the packet directions sequence).

The last FLOWSTATS features are packet histograms that contain binned counts of packet sizes and inter-packet times for each direction across the entire flow. Each histogram has eight bins, which have a logarithmic scale; please refer to the

CESNET-QUIC22 dataset article [10] for more details. In total, packet histograms are represented as 32 features.

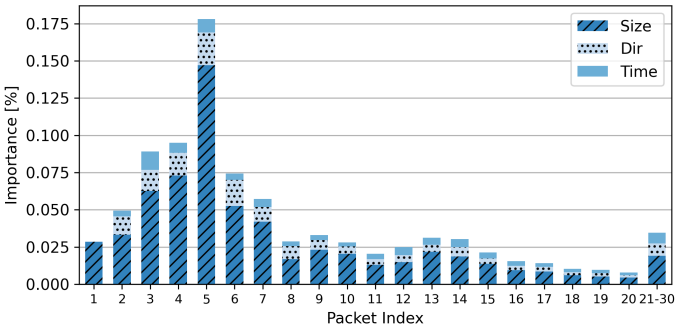
2) Data preprocessing: We standardize packet sizes and times using mean and variance (z-score normalization). Directions of packets are encoded as ±1, and there is no need to standardize them. We normalize packet histograms so that the sum of bins of each histogram is equal to one.

For FLOWSTATS features that do not have a fixed range, such as the number of transmitted packets, we opted for robust scaling that uses median and inter-quartile range instead of mean and variance to limit the negative influence of outliers. We also clip FLOWSTATS features to their 0.99 quantiles to further reduce extreme values. Packet times are clipped to a maximum of 15 seconds. Packet sizes are clipped to the maximum of 1460 bytes, which is the maximum size of an unfragmented UDP packet when the typical MTU 1500 is used.

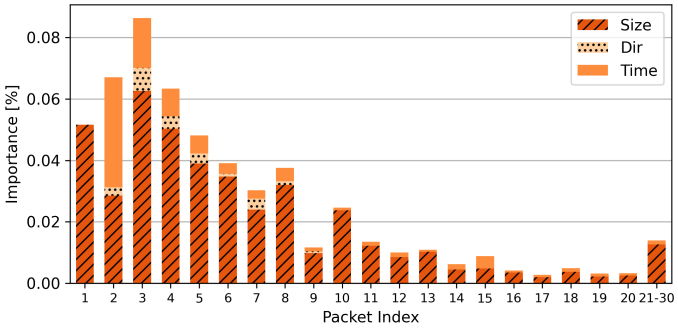
3) Multi-modal CNN architecture: The neural network has two separate chains for processing PSTATS and FLOWSTATS; the outputs of those chains are concatenated to create a shared representation, which is processed with more layers until the final classification layer and the softmax function. PSTATS are processed with 1D convolutions, which require a fixed input length. The packet sequences are padded with zeros to the maximum length of 30. The FLOWSTATS features are processed with linear layers. Dropouts are used for regularization, and batch normalization is used to make the network training faster and more stable. The network uses ReLU as the activation function.

This architecture is reused from [6]; to adapt it for the QUIC classification, we did the following modifications: 1) increased the number of convolution layers and their size (i.e., the number of output channels), 2) increased the size of linear layers, and 3) put a pooling layer after the convolutions instead of just flattening the feature maps (the output of convolutions). We opted for global pooling, which transforms each feature map to a single scalar, and as the pooling operation, we choose Generalized Mean Pooling (GeM) with a learnable parameter p. To select the exact number of layers, their sizes, dropout rates, etc., we used the Optuna framework [18], which implements the state-of-the-art algorithms for hyperparameters search. The best hyperparameters were selected based on the classification performance on the validation set. The final model is visualized in Figure 1 and has 2.2 million trainable parameters. A detailed layer printout of the model's PyTorch implementation is provided in Appendix A.

4) Deep learning evaluation: The model was trained on the first week of the CESNET-QUIC22 dataset, which consists of 30 million flows. We performed five training runs, where each run had its own train-validation split (80% train, 20% validation) of the first week. Each model was tested using a 20 million subset from the remaining three weeks (this subset was selected once and is the same for all training runs of all three classification approaches). The results are presented in Table I, which contains averages across the five runs. We opted for testing on the 20 million subset to save computational resources and because results on unsampled test weeks are presented in Section V.



(a) mm-CNN



(b) LightGBM

Fig. 2. SHAP-based feature importances of the packet sequence input.

The model training and evaluation follow the standard deep-learning pipeline. We opted for the AdamW optimizer, the one-cycle learning rate scheduler, and the cross-entropy loss. We trained the neural network for 30 epochs, which showed the best performance. For each epoch, we sample a random subset of one million training flows.[3] During the training, the classification performance is measured on the validation set, and the best-performing epoch is saved.

5) Deep learning XAI interpretations: It is crucial to understand the inner workings of a model and identify features that are the most important for making correct predictions. In this section, we provide feature importances of the trained neural network. For that, we used the SHAP method [19] to compute per-sample attributions, which are then aggregated (mean of absolute values) to create a global view into the model's functioning. We normalize the global attributions so their sum is one and thus can be interpreted as "percentages" of total importance. This normalization also allows a better comparison of mm-CNN and LightGBM feature importances.

We focus on packet metadata sequences to determine which packet positions are the most important. The results are shown

[3]We use sampled training epochs instead of the standard definition of an epoch as a pass of the entire training set. With smaller epochs, we can better track the performance during training (compute the validation performance each epoch, save the best model, etc.) and can tune the amount of training in smaller steps (as opposed to having to use multiples of the entire large training set). The training set has 24 million samples after the train-validation split; thus, our 30 sampled epochs correspond to 1.25 passes of the training set (though the selection of samples is randomized).

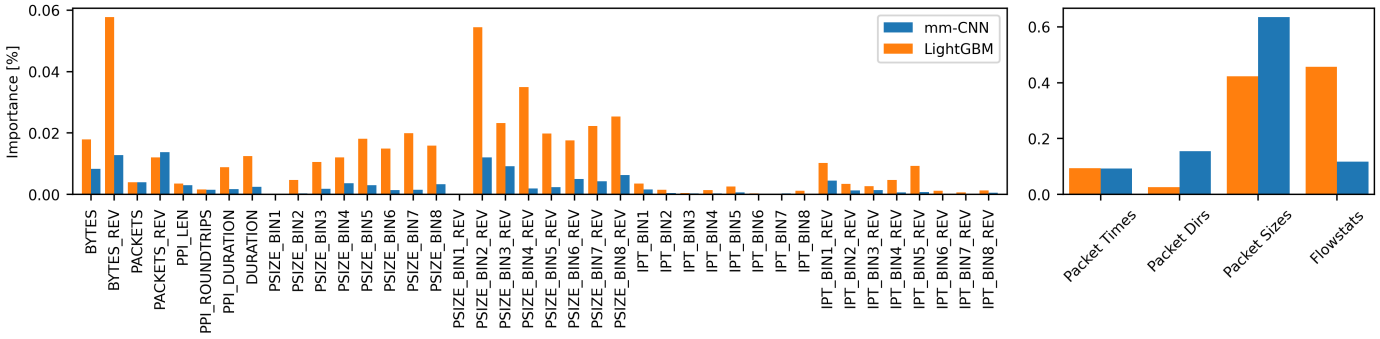


Fig. 3. SHAP-based feature importances of FLOWSTATS features of both LightGBM and mm-CNN classifiers (left); and aggregated feature importances of packet times, directions, sizes, and FLOWSTATS features (right).

in Figure 2a. Packets sent at the start of the communication are more informative than later packets, and packet sizes are more important than inter-packet times and directions. We interpret the peak at the 5th packet as related to the transmission of the server certificate. Even though the certificate is encrypted, its size can still leak information useful for service identification.

B. LightGBM classifier

For the LightGBM classifier, we use the same FLOWSTATS and PSTATS input features with the same standardization as for the mm-CNN classifier (see Section IV-A1). As opposed to convolutions in mm-CNN, which use a sliding window over the packet sequence, LightGBM considers each packet position in PSTATS as a separate feature without relation to the adjacent packets. This difference, however, does not stop LightGBM from achieving good results.

1) LightGBM evaluation: We use the same training and evaluation setup as for the mm-CNN classifier (five train-validation splits of the first week and the same 20 million subset for testing), with the difference that the LighGBM training requires the whole training set to be loaded and processed at once. The model was trained up to one hundred iterations with the early stopping technique—until the validation performance had not improved for ten iterations.

We used Optuna to find the best configuration of the model based on the validation performance. We tuned the following parameters in a step-wise manner: `feature_fraction`, `num_leaves`, `bagging_fraction`, `bagging_freq`, `lambda_l1`, `lambda_l2`, and `min_child_samples`.[4]

2) LightGBM feature importances: We used the same SHAP-based approach to compute feature importances of the LightGBM classifier. The exact algorithm for obtaining SHAP values differs; for mm-CNN, we used the Captum DeepLiftShap implementation;[5] for LightGBM, we used its internal implementation provided in the `predict` method (with the `pred_contrib` argument set).

[4]These parameters and their order are used in `LightGBMTuner`, which is a special Optuna hyperparameter tuner for LightGBM. The description of individual parameters can be found here https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html.

[5]https://captum.ai/api/deep_lift_shap.html.

The resulting feature importances for PSTATS are shown in Figure 2b. LightGBM focuses less on packet directions, and the importance of individual packet sizes is more spread out between the first ten packets (as opposed to the peak at the 5th packet for mm-CNN). Figure 3 shows a comparison of FLOWSTATS importances and aggregated feature importances of the two models. The main difference is that LightGBM relies much more on FLOWSTATS features—the relative importance of FLOWSTATS and packet sizes is almost the same; on the other hand, for mm-CNN, packet sizes are 6x more important than FLOWSTATS.

C. IP baseline

The IP-based classifier is described in Algorithm 1 and has two hyperparameters: `/P` network prefix for inexact subnetwork matching, and `exact_T` threshold for cases when multiple services are hosted on the same IP address.

Algorithm 1 IP-based classification algorithm.

Require: `P` ← Network prefix
Require: `exact_T` ← Threshold for exact match

procedure TRAIN($[IP_1, IP_2, IP_n]$,$[label_1, label_2, label_n]$)
  for $i \in 1...n$ **do**
    Dict[IP_i][$label_i$] += 1
    $prefix \leftarrow IP_i$.getNetworkPrefix(`P`)
    PrefixDict[$prefix$][$label_i$] += 1

procedure CLASSIFY(IP)
  if IP in Dict and Dict[IP].getMaxFract() $\geq$ `exact_T` **then**
    return Dict[IP].getMaxLabel()
  else
    $prefix \leftarrow IP$.getNetworkPrefix(`P`)
    if $prefix \in$ PrefixDict **then**
      return PrefixDict[$prefix$].getMaxLabel()
  return NoPrediction

During the training, for each IP address and its /`P` IP prefix, the algorithm stores hosted services and the number of occurrences into dictionaries. For classification, an exact match trial is performed. When the exact match is not successful, either due to an unknown (not present in the training set) IP address or due to multiple co-hosted services at the given IP (the fraction of the most occurring service is smaller than `exact_T`), a subnetwork match is performed, and the service

TABLE II

PER-WEEK ACCURACY (ACC.), PROVIDER ACCURACY (PROV. ACC.), F1, AND RECALL SCORES OF CLASSIFIERS TRAINED WITH DATA FROM WEEK 1. THE RESULTS ARE IN THE FORMAT AVERAGE (± STDEV) ACROSS FIVE TRAINING RUNS.

	LightGBM				Multi-modal CNN				IP baseline			
	Acc.	Prov. Acc.	F1	Recall	Acc.	Prov. Acc.	F1	Recall	Acc.	Prov. Acc.	F1	Recall
Week 2	87.34% (±0.26%)	98.32% (±0.02%)	83.17% (±0.13%)	80.65% (±0.14%)	86.27% (±1.53%)	98.32% (±0.19%)	86.12% (±0.49%)	84.02% (±0.50%)	69.17%	99.82%	70.99%	71.81%
Week 3	77.89% (±0.71%)	98.20% (±0.04%)	79.29% (±0.37%)	76.84% (±0.35%)	72.79% (±4.08%)	97.68% (±0.61%)	80.38% (±1.40%)	78.59% (±1.33%)	69.16%	98.99%	69.61%	70.05%
Week 4	76.88% (±0.67%)	98.24% (±0.02%)	78.86% (±0.21%)	76.44% (±0.27%)	72.35% (±4.15%)	97.76% (±0.63%)	80.69% (±1.42%)	78.75% (±1.38%)	67.82%	98.27%	68.52%	69.11%
	LightGBM no handshake				**Multi-modal CNN no handshake**							
	Acc.	Prov. Acc.	F1	Recall	Acc.	Prov. Acc.	F1	Recall				
Week 2	87.40% (±0.06%)	97.11% (±0.05%)	79.37% (±0.15%)	76.14% (±0.11%)	87.96% (±0.12%)	97.50% (±0.02%)	82.55% (±0.03%)	79.63% (±0.07%)				
Week 3	83.81% (±0.05%)	97.27% (±0.04%)	77.54% (±0.19%)	74.17% (±0.10%)	84.81% (±0.28%)	97.48% (±0.02%)	80.95% (±0.10%)	77.78% (±0.20%)				
Week 4	83.09% (±0.04%)	97.35% (±0.04%)	77.16% (±0.26%)	73.84% (±0.26%)	84.32% (±0.27%)	97.53% (±0.03%)	80.41% (±0.12%)	77.33% (±0.21%)				

with the maximum number of occurrences in the subnetwork is selected. When even the subnetwork was not present in the training set, the classifier makes no prediction (which counts as a wrong prediction during the evaluation).

1) IP baseline evaluation: As in the previous cases, the IP baseline was trained on the first week and evaluated on the 20 million subset of the remaining three weeks. However, we used all samples of the first week for single training without train-validation splitting.[6]

During the evaluation, we found out that the subnetwork matching has poor performance and that it is best to set `exact_T` to zero—in other words, if an IP was observed during the training period, predict the service with the highest number of occurrences at the given IP. The subnetwork matching, with `P=/30` for IPv4 and `P=/120` for IPv6, is therefore used only for IP addresses not present in the training set. The performance of the IP baseline is shown in Table I.

V. DETAILED EVALUATION OVER TIME

The previous sections described each classifier's design and measured performance on the 20 million subset of the three test weeks. Looking at the results in Table I, one could conclude that LightGBM is the best model; however, before making such a conclusion, we first should measure classifiers' performance in each test week and investigate it in more detail.

In this section, we compare the properties of the three classifiers and their performance over the one month of traffic provided in the CESNET-QUIC22 dataset. We use the same models trained on the first week as in Section IV; however, we report performance metrics for each test week so that it is possible to observe how the performance evolves over time and uncover various interesting phenomena influencing classifiers' performance. The detailed per-week results are shown in the upper part of Table II (the lower part with no handshake models is discussed in the following section).

[6]Using random train-validation splitting does not make sense for the IP baseline, for which the order of the time axis must be preserved. Without train-validation splitting, the IP baseline is in itself deterministic, so there is no reason to perform multiple training runs.

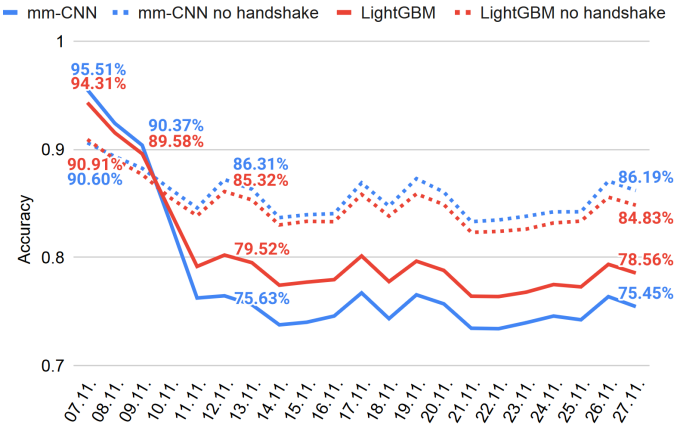


Fig. 4. Per-day accuracy scores of the original versus no handshake ML models during the entire test period (7th–27th Nov. 2022).

There is a clear pattern across all three weeks of LightGBM being better than mm-CNN in accuracy but worse in recall. This means that LightGBM works better on more prevalent classes, which are often Google and Facebook services, while mm-CNN is better on other classes in general. All three classifiers have comparable provider-level accuracy, with the IP baseline being the best with 99.82% in the second week. The gaps between provider accuracies and regular, fine-grained accuracies mean that the classifiers tend to make mistakes among the same provider; for example, mismatching Google Play traffic with Google Analytics.

We measured a huge drop in the fine-grained performance of ML-based classifiers (LightGBM, mm-CNN) between the second and the third week. For LightGBM, accuracy dropped from 87.34% to 77.89% (–9.45%); for mm-CNN, the drop was even bigger, from 86.27% to 72.79% (–13.48%). Recall drops were lower but still considerable. On the other hand, the IP baseline remained stable for the entire test period.

The two main observations—the huge drop in the performance of ML classifiers between the second and the third week

and the stable performance of the IP baseline over the entire test period—are further discussed in the following sections.

A. Performance drop of ML-based classifiers

We start by investigating the origin of the performance drop of the ML-based classifiers. We focus on the case of mm-CNN since it experienced a larger performance drop than LightGBM did; however, the following analysis applies to both models.

Looking at per-day recalls of individual services during the second week, we observed a steep drop in Google services. Out of 27 Google services, 12 experienced a recall decrease larger than 20%, and five experienced a decrease larger than 50%. Figure 5 shows the mm-CNN average recall of Google services compared to the rest, which remained stable.

Given the mm-CNN's strong reliance on the fifth packet of the connection (see Figure 2a), **we formed a hypothesis that a change of the Google TLS server certificate caused the decrease in the performance of ML-based classifiers**.

The dataset does not contain information about certificates; however, since the dataset was created from real-world traffic, we used Certificate Transparency (CT) logs for the certificate change investigation. Google renews its `*.google.com` certificate quite often—a new certificate is logged in CT logs at least once a week. On 1st Nov. 2022, Google added two domains into the X509v3 alternative name field, resulting in an increase in the certificate size. The time needed for the deployment of this certificate could then explain the performance drop observed during the second week (7th–13th Nov. 2022).

To further validate the hypothesis, we trained the LightGBM and mm-CNN classifiers with removed handshake packets—we zeroed out the first eight packets from the packet sequences.[7] The performance of no handshake classifiers is shown in the lower part of Table II, and a per-day comparison with the original models is depicted in Figure 4. Without handshake packets, both classifiers showed more stable performance across all test weeks, supporting our hypothesis about the certificate influence. The accuracy drop of the no handshake mm-CNN between the second and third week is −3.15%, while the drop of the original mm-CNN is −13.48%. For LightGBM, the drop changed from −9.45% to −3.59%. The training process also stabilized, and the standard deviation of performance metrics is much smaller when the handshake information is not used. For both ML-based classifiers, however, the removal of the handshake information resulted in a significant decrease in recall scores.

Of all tested classifiers, mm-CNN trained without the handshake packets has the best accuracy scores in all three test weeks; this is due to better performance on data-drifted Google services, which represent around 56% of the dataset flows. Original mm-CNN with handshake packets offers the best recall scores in all three test weeks, and the IP baseline offers the best provider-level accuracy scores in all three test weeks. Thus, there is an apparent trade-off when choosing the classifier depending on the preferred metric.

[7]We used the same training pipeline and hyperparameters as described in Sections IV-A and IV-B; flow statistics input features remained unchanged.

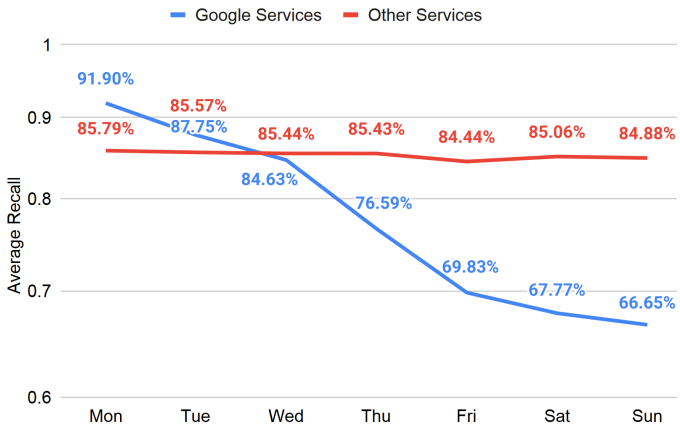


Fig. 5. Multi-modal CNN performance in the second week (7th–13th Nov. 2022). The average recall of Google services dropped from ∼92% to ∼67% while the recall of other services remained stable.

B. Stability of IP addresses

The IP-based classifier proved to have more stable (but worse than ML-based) performance in the studied period. Even though the IP baseline fails at recognizing 24 services, which have recall lower than 10% in all weeks, 58 services have recall higher than 99% and a recall drop lower than −1%. Out of the remaining services, 12 services have a recall drop lower than −5%. During the test period, we noticed three services that changed all their IP addresses, resulting in a recall drop to 0%. Overall, the IP baseline either works well and is stable; or does not work at all for co-hosted services (of course, there are few services with behavior somewhat between, but these are the two main patterns).

TABLE III
PER-WEEK RECALLS OF SELECTED SERVICES FOR IP BASELINE (IP), LIGHTGBM (LGBM), AND MULTI-MODAL CNN (MM-CNN).

		Week 2	Week 3	Week 4	Diff
alza-www	IP	99.99%	23.87%	0.00%	-99.99%
	LGBM	80.33%	79.16%	74.49%	-5.83%
	mm-CNN	85.75%	85.57%	86.54%	0.79%
mentimeter	IP	100.00%	31.70%	0.00%	-100.00%
	LGBM	62.30%	51.94%	61.72%	-0.58%
	mm-CNN	69.58%	60.40%	71.34%	1.76%
kiwi-com	IP	32.14%	0.91%	1.66%	-30.49%
	LGBM	81.26%	70.88%	53.66%	-27.60%
	mm-CNN	87.38%	83.65%	76.46%	-10.92%
rohlik	IP	100.00%	100.00%	100.00%	0.00%
	LGBM	52.73%	65.93%	57.65%	4.92%
	mm-CNN	50.57%	65.46%	56.01%	5.44%
doi-org	IP	99.98%	100.00%	99.99%	0.01%
	LGBM	88.45%	86.48%	69.32%	-19.13%
	mm-CNN	91.28%	89.71%	68.02%	-23.26%

We selected a couple of services to demonstrate the differences between the studied classifiers; their per-week recalls are shown in Table III. We can notice that `alza-www` (an e-commerce service) and `mentimeter` (a web-presentation

software) show an IP baseline recall drop from 100% to 0%, but remain stable for ML-based classifiers. Another service with a big recall drop is `kiwi-com` (flights broker); nevertheless, the IP baseline struggled with this service from the beginning, and ML-based classifiers also are not stable in this case. All these three services are hosted in the Cloudflare CDN infrastructure, in which IP addresses might change relatively often. This is not a surprising observation or explanation; however, it is still interesting to capture this phenomenon in a real-world dataset in 3 out of 102 services (or out of 78 that were recognized at the beginning) during a month-long test period.

There are also services for which the IP baseline works better than ML-based models. For example, the `doi-org` service, for which IP baseline has 100% recall all three weeks while ML-based models get worse from ∼100% to ∼70%. Another example is `rohlik` (online food shopping), for which IP baseline has stable 100% performance while ML models have ∼55%.

VI. Discussion

The fine-grained QUIC service classification proved to be a challenging but doable task. To summarize the results: multi-modal CNN achieved the best average recall of 84.02%, multi-modal CNN trained without the handshake packets achieved the best accuracy of 87.96%, and the IP baseline offered the best provider-level accuracy surpassing 98% in all test weeks. During our experiments, we collected a lot of insights, and the most important ones are summarized in Table IV.

Direct comparison with other works, which use different datasets, is problematic. For QUIC, no related works are similar enough considering the dataset size, service count, and service composition. For TLS, our previous work [6] evaluated a similar neural network architecture on a dataset from the same source network and achieved a classification accuracy of 97.41%, which is about 10% better than the best result for QUIC. However, the exact classes differ (e.g., the QUIC dataset has a larger fraction of Google service). Nevertheless, this is still the most similar related work, and we believe this comparison is valid.

We performed a one-month evaluation, which we consider to be the best practice because it can show the time-related properties of the studied models. In our case, the long test period uncovered a sharp reduction in the performance of ML-based models. We attribute this reduction to network traffic data drift. We provided several indications supporting that the data drift was related to the certificate change of a major service provider—Google. According to the CT logs, the size of the `*.google.com` certificate changed eight times between 1.11.2022 and 28.2.2023. Therefore, we assume that CESNET-QUIC22 does not capture a rare event but rather a general pattern that can happen more than twice a month.

To our knowledge, such fast data drift has not been observed in previous studies targeting network traffic classification. Compared to the related work of Malekghaini et al. [14], who measured data drift on datasets years or months apart, we show that substantial data drift can develop within a single week.

TABLE IV
Key observations and findings.

Observation	Reference
mm-CNN has the highest recall, while it is also the most susceptible to data drift in the form of certificate change.	Table II Figure 4
Models trained without the handshake packets are more stable and robust in the presence of data drift.	Table II Figure 4
IP baseline either works well (for over ∼50% of dataset services) or does not work at all (∼25%).	Section V-B
IP-based classification broke down (0% recall) for three services during the one-month test period (out of 78 that were recognized at the beginning).	Table III
LighGBM uses FLOWSTATS features much more than mm-CNN.	Figure 3
XAI-based interpretations show that mm-CNN focuses a lot on the fifth packet of the sequence, which in most cases should be related to the transmission of the server certificate.	Figure 2a
We observed a huge 30% drop in recall of Google services during the second week. This demonstrates how massive and sudden the data drift in network traffic can be.	Figure 5
Network traffic classifiers should be evaluated on multiple consecutive time periods; otherwise, interesting and important phenomena might not be discovered.	Table I Table II

To mitigate the observed performance drop, we can exclude the first eight packets from the packet sequence input and direct the model's attention to different characteristics. However, models trained without the handshake packets achieved lower recall scores. The fact that multi-modal CNN is more susceptible to data drift compared to LighGBM might be related to different processing of packet metadata sequences. Multi-modal CNN uses 1D convolution, while LightGBM processes each packet position as a separate feature. Convolutions can extract information from consecutive packets and, for example, sum consecutive packet sizes to obtain the size of the encrypted server certificate. Convolutions are also invariant to exact packet positions; thus, CNN can find server certificates starting at different positions in the sequence. For LightGBM, extracting the same information is much more difficult when packet positions and packet features are processed without relation to each other. This makes mm-CNN less robust to certificate changes, which leads to an interesting research question for future work—to investigate the differences in packet sequence processing in more detail, find a method to combine the two approaches, and get the best from both.

Limitations of the presented approach: We would like to state several limitations of this work. First, we focused on the standard closed-world classification and evaluated trained classifiers on known classes. As several related works pointed out, it is important for practical deployment to handle out-of-distribution detection to recognize unknown classes that were not present in the training set. Second, even though the dataset

spans one month, which is the longest time span for a public QUIC dataset, a longer dataset would still be needed for better evaluation of classifiers' properties over time; for example, a longer time span would allow us to estimate the frequencies of events such as the observed massive data drift of Google services; or the IP baseline breaking down for some services. Third, of course, more classes would make the problem harder and more realistic; still, the used dataset has the most class labels out of the public QUIC datasets.

VII. Conclusion

The goal of this work was to explore the classification of encrypted QUIC traffic in a comprehensive fashion, evaluating multiple classification models on multiple consecutive test periods. This allowed us to uncover properties of individual classifiers and changes in their behavior over time. We based our evaluation on CESNET-QUIC22, which is the largest, with the most class labels, and the longest-spanning public dataset of QUIC flow traffic.

Apart from per-week per-classifier results presented in Table II, we collected a number of observations, out of which the most important are discussed in Section VI and summarized in Table IV. In encrypted traffic analysis, it is standard to use the first N packets of communication; we are the first to report that omitting the handshake packets transmitted at the beginning of the communication might bring some benefits in the form of improved robustness against data drift.

As the first paper targeting QUIC traffic classification on a larger scale, we set a new baseline for this task. Moreover, our results create a foundation for future research, e.g., in novel multi-modal CNN architectures that would be more resistant to data drift of network traffic; or in designing an ensemble model that would combine the benefits of IP-based classification with the benefits of ML models processing packet metadata sequences.

Acknowledgements

This work was supported by the Ministry of the Interior of the Czech Republic, grant No. VJ02010024: *"Flow-Based Encrypted Traffic Analysis,"* and also by the Grant Agency of the Czech Technical University in Prague, grant No. SGS23/207/OHK3/3T/18. Computational resources were supplied by the project *"e-Infrastruktura CZ"* (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] Microsoft, Inc., 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/storage/file-server/smb-over-quic

[2] M. Bishop, "HTTP/3," RFC 9114, Jun. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9114

[3] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9000

[4] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify HTTPS services," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Apr. 2016. [Online]. Available: https://doi.org/10.1109/noms.2016.7502818

[5] L. Yang, A. Finamore, F. Jun, and D. Rossi, "Deep learning and zero-day traffic classification: Lessons learned from a commercial-grade dataset," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4103–4118, Dec. 2021. [Online]. Available: https://doi.org/10.1109/tnsm.2021.3122940

[6] J. Luxemburk and T. Čejka, "Fine-grained TLS services classification with reject option," *Computer Networks*, vol. 220, p. 109467, Jan. 2023. [Online]. Available: https://doi.org/10.1016/j.comnet.2022.109467

[7] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "Improving SNI-based HTTPS security monitoring," in *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, Jun. 2016. [Online]. Available: https://doi.org/10.1109/icdcsw.2016.21

[8] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood, "TLS Encrypted Client Hello," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-esni-15, Oct. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-esni/15/

[9] S. Rezaei and X. Liu, "How to achieve high classification accuracy with just a few labels: A semisupervised approach using sampled packets," in *Advances in Data Mining - Applications and Theoretical Aspects, 19th Industrial Conference, ICDM 2019, New York, USA, July 17 - July 21, 2019*, P. Perner, Ed. ibai Publishing, 2019, pp. 28–42.

[10] J. Luxemburk, K. Hynek, T. Čejka, A. Lukačovič, and P. Šiška, "CESNET-QUIC22: A large one-month QUIC network traffic dataset from backbone lines," *Data in Brief*, vol. 46, p. 108888, Feb. 2023. [Online]. Available: https://doi.org/10.1016/j.dib.2023.108888

[11] S. Ramanathan, A. Hossain, J. Mirkovic, M. Yu, and S. Afroz, "Quantifying the impact of blocklisting in the age of address reuse," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20. New York, NY, USA: ACM, 2020.

[12] N. P. Hoang, A. Akhavan Niaki, N. Borisov, P. Gill, and M. Polychronakis, "Assessing the privacy benefits of domain name encryption," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 290–304.

[13] I. Akbari, M. A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, "A look behind the curtain: Traffic classification in an increasingly encrypted web," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 1, pp. 1–26, Feb. 2021. [Online]. Available: https://doi.org/10.1145/3447382

[14] N. Malekghaini, E. Akbari, M. A. Salahuddin, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, "Deep learning for encrypted traffic classification in the face of data drift: An empirical study," *Computer Networks*, vol. 225, p. 109648, 2023.

[15] V. Tong, H. A. Tran, S. Souihi, and A. Mellouk, "A novel quic traffic classifier based on convolutional neural networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[16] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "MIMETIC: Mobile encrypted traffic classification using multimodal deep learning," *Computer Networks*, vol. 165, p. 106944, Dec. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128619304669

[17] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "DISTILLER: Encrypted traffic classification via multimodal multitask deep learning," *Journal of Network and Computer Applications*, vol. 183-184, p. 102985, Jun. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804521000126

[18] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[19] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

APPENDIX A
PYTORCH IMPLEMENTATION DETAILS OF MULTI-MODAL CNN

```
MM_CNN_30(
  (cnn): Sequential(
    (0): Conv1d(3, 200, kernel_size=(7,), stride=(1,), padding=(3,))
    (1): ReLU()
    (2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Sequential(
      (0): Conv1d(200, 200, kernel_size=(5,), stride=(1,), padding=(2,))
      (1): ReLU()
      (2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (4): Sequential(
      (0): Conv1d(200, 200, kernel_size=(5,), stride=(1,), padding=(2,))
      (1): ReLU()
      (2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (5): Sequential(
      (0): Conv1d(200, 200, kernel_size=(5,), stride=(1,), padding=(2,))
      (1): ReLU()
      (2): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (6): Conv1d(200, 300, kernel_size=(5,), stride=(1,))
    (7): ReLU()
    (8): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): Conv1d(300, 300, kernel_size=(5,), stride=(1,))
    (10): ReLU()
    (11): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): Conv1d(300, 300, kernel_size=(4,), stride=(2,))
    (13): ReLU()
  )
  (cnn_global_pooling): Sequential(
    (0): GeM(kernel_size=10, p=3.0000, eps=1e-06)
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.1, inplace=False)
  )
  (fc_flowstats): Sequential(
    (0): Linear(in_features=44, out_features=225, bias=True)
    (1): ReLU()
    (2): BatchNorm1d(225, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Sequential(
      (0): Linear(in_features=225, out_features=225, bias=True)
      (1): ReLU()
      (2): BatchNorm1d(225, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (4): Sequential(
      (0): Linear(in_features=225, out_features=225, bias=True)
      (1): ReLU()
      (2): BatchNorm1d(225, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (5): Linear(in_features=225, out_features=225, bias=True)
    (6): ReLU()
    (7): BatchNorm1d(225, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): Dropout(p=0.1, inplace=False)
  )
  (fc_shared): Sequential(
    (0): Linear(in_features=525, out_features=600, bias=True)
    (1): ReLU()
    (2): BatchNorm1d(600, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): Dropout(p=0.2, inplace=False)
  )
  (out): Linear(in_features=600, out_features=102, bias=True)
)
```

```
def forward(self, t):
  pstats, flowstats = t
  out_cnn = self.cnn(pstats)
  out_cnn = self.cnn_global_pooling(out_cnn)
  out_flowstats = self.fc_flowstats(flowstats)
  out = torch.column_stack([out_cnn, out_flowstats])
  out = self.fc_shared(out)
  logits = self.out(out)
  return logits
```

Generalized Mean Pooling (GeM) implementation adapted from https://www.kaggle.com/code/scaomath/g2net-1d-cnn-gem-pool-pytorch-train-inference.