A Critical Study of Few-shot Learning for Encrypted Traffic Classification

Elham Akbari*, Sheikh A. Tahmid*, Navid Malekghaini*, Mohammad A. Salahuddin*, Noura Limam*, Raouf Boutaba*, Bertrand Mathieu†, Stephanie Moteau†, and Stephane Tuffin†
*David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
{eakbaria, sa2tahmi, nmalekgh, m2salahu, n2limam, rboutaba}@uwaterloo.ca
†Orange Labs, Lannion, France
{bertrand2.mathieu, stephanie.moteau, stephane.tuffin}@orange.com

***Abstract*—Over the past twenty years, a plethora of methods have been proposed for encrypted traffic classification (ETC), while the Server name indication (SNI) is deemed to solve the problem of classification for TLS traffic. However, SNI-based classification has its pitfalls and the SNI will likely be pushed into the encrypted tunnel in the future. In this work, we envision a futuristic scenario in which encrypted SNI is the norm and labeled traffic flows are scarce. In such settings, we tackle the problem of traffic classification at ISP level using few-shot learning. By means of six real-world ISP-level datasets collected between 2019 and 2021 and two publicly available client-side datasets, we study the performance of a few-shot learner on TLS data, including its cross-dataset generalizability. We further investigate the effect of the number of required labeled samples on the learner's performance. Our experiments show that the dataset-specificity of deep learners carries over to few-shot meta-learning, and calls for addressing the problem of generalizability for deep learning architectures.**

***Index Terms*—Encrypted traffic classification, Meta-learning, Few-shot learning, Matching networks**

I. Introduction

Outside controlled environments, with no visibility into the traffic-generating clients, labeling real-life encrypted traffic is akin to the classification task for a non-labeled dataset. While SNI-based labeling paves the way for supervised classification of encrypted traffic, it has its limitations. The SNI extension in TLS is primarily employed by servers that host multiple domain names to identify the appropriate SSL certificate for a client that initiates a connection. However, there is no guarantee for an SNI field to exist in a TLS flow or to have a value readable by third-party packet-sniffers. For example, the authors in [1] report that only 14% of the flows in their dataset were labelable using the SNI field. Moreover, some Content Delivery Networks (CDNs) have already started supporting the Encrypted SNI feature of TLS, which pushes the SNI field into the encrypted tunnel and prevents third parties from spying on users. In the absence of SNI, traffic classification models will rely on traffic collected in controlled environments for training, and will need to be generalizable to non-independent and identically distributed (non-i.i.d) data. Prior works [2]–[6] have proved that porting a model from dataset to dataset is challenging, suggesting a need for new approaches to traffic classification.

Few-shot learning allows a deep model to transfer the knowledge learned in previous learning experiences to a new task using only a few labeled data samples. Among few-shot learners, a certain class of metric-based models have been proposed that allow the insertion of any deep architecture as an embedding function [7]–[9] to effectively capture the features of the input data. This property allows the integration of tried and tested encrypted traffic classifiers with those few-shot models. The encrypted traffic classification (ETC) literature has explored different deep architectures and shown their merit [1], [10]–[12]. The effectiveness of deep models is often attributed to their ability to automatically discover distinguishing features in data, a trait that seems promising for their use as an embedding function (cf., Section III), as well as an advantage over classical machine learning (ML) models.

Another advantage that few-shot learning offers is a task-based approach to learning, which can potentially provide an advantage over other models in terms of generalizability to new datasets. Based on the definition of a task (cf., Section II), distinct network traffic datasets impose different classification tasks. Numerous prior works [2]–[4], [13] have found an acute drop in performance, when a model is trained and evaluated on different datasets. This poses a real challenge for production-level traffic classification, where the expectation is that a trained model should perform well on future collected data, unavailable at training time. Another advantage offered by the class of few-shot models explored in this work is their ability to extend their knowledge to previously unseen classes of data.

In this work, we study the performance of a few-shot approach, carefully designed based on state-of-the-art deep traffic classifiers, by means of a number of purely TLS traffic traces collected at an ISP. We keep the experiments' conditions realistic by refraining from filtering out viable traffic flows for which a ground truth is available, and by masking the SNI values where header byte features are used. Thus, we report the performance of the few-shot model on all labelable traffic flows in the traffic traces and ensure that the classification task is more complex than a text lookup. Our contribution lies in providing the answer to the following questions:

- Do unique features of a few-shot learner, such as a metric-based loss function and the use of evaluation support

978-3-903176-59-1

samples, offer an advantage in terms of generalizability and/or performance over a traditional classifier?
- How does the number of labeled samples affect the performance of the few-shot classifier and how many such samples are needed for a few-shot classifier to yield acceptable performance?
- How does the few-shot classifier fair against a comparable classical k-nearest neighbors (kNN) algorithm in terms of performance, given the same number of labeled samples?

The rest of the paper is organized as follows. We review the background and discuss the related works in Section II. Section III explains our approach, including the choice of few-shot learner, the datasets and the extracted features. Section IV answer the three questions mentioned above and showcases the performance and generalizability of the few-shot approach under various ETC scenarios for header bytes and flow time-series input features. We discuss the results in Section V and conclude in Section VI.

II. Background and Related Works

A. Knowledge Transfer

Knowledge transfer across tasks has been a focus across a number of topics in ML, such as transfer learning, multi-task learning, and meta-learning. Central to the idea of knowledge transfer is the notion of a task. A task $\mathcal{T}_i$ is defined as a distribution of data points $p_i(x)$, a distribution of labels over data points $p_i(y|x)$, and a loss function $\mathcal{L}_i$, as shown in Equation 1 [14]:

$$\mathcal{T}_i \triangleq \{p_i(x), p_i(y|x), \mathcal{L}_i\} \tag{1}$$

Different tasks vary in one or more of the three elements in Equation 1. For example, the classification of two datasets collected over two different networks constitutes two separate tasks as the underlying $p_i(x)$ differs between the two datasets, even if the labeling mechanism is the same. Similarly, classifying a hierarchically labeled dataset at two different levels of the hierarchy denotes two different tasks, because $p_i(y|x)$ differs between the two classification tasks.

B. Few-shot and Meta-learning

Few-shot learning, the focus of this paper, is typically categorized under meta-learning [15]. It aims at developing models that can learn a previously unseen class using only a few labeled data samples. Few-short learning is a response to the data intensive nature of deep models, which limits their applicability in scenarios where labeled data is intrinsically scarce, e.g., with new emerging classes, rare categories, or when labeling is cumbersome or expensive.

As with all meta-learning approaches, in few-shot learning, a *learner* is created by training a deep model on a set of tasks with the objective of minimizing the loss on a new task from the same task distribution. However, few-shot learning differs from other meta-learning approaches such that the number of training data points in the target task is small, therefore, the learner should be trained for *fast* learning.

Matching Networks (MN) [7], along with Prototypical Networks [8] and Relation Networks [9], are examples of episode-based few-shot learners. These models are similar in using a deep embedding function and training it via episodes. Prototypical Networks have been employed for few-shot traffic classification in [16]. However, as most works on traffic classification, [16] restricts the training and evaluation data to the same largely non-encrypted dataset [17], which leaves a realistic estimate of the model's performance to future work.

Contrastive learners such as Siamese Networks [18], Triplet Networks [19], and SimCLR [20] are similar to the above few-shot learners, in that, they also feature a deep encoder architecture and are metric-based. They train the deep encoder to differentiate between augmentations of the same data sample as opposed to different data samples. Contrastive learners have been applied for few-shot ETC in [19], [21], [22] and differ from the approach in this paper in that they use unlabeled data in pretraining.

C. Learning from Few Samples in the Networking Domain

Traffic classification. Rezaei and Liu [23] were the first to address traffic classification in the presence of few labels. They suggested a transfer learning approach where the statistical features of flows were used as labels to pre-train a convolutional neural network (CNN) model using an unlabeled dataset. The pre-trained model was then incorporated into another model and fine-tuned using a few labeled flows. The authors showed a comparable performance to a random forest (RF) classifier using 30 flows augmented by different flow sampling techniques. They extended the same approach to multi-task learning in [24] and showed that for 10 to 100 labeled samples per class, multi-task learning outperforms a baseline CNN+recurrent neural network (RNN)-based deep model as well as their transfer learning approach.

Towhid and Shahriar [22] proposed a self-supervised approach for traffic classification using time-series features. Their deep model includes a 1D ResNet architecture as an encoder. Each flow was sampled multiple times using a sampling technique proposed in [23] to augment data. The authors showed that their model is effective when pre-trained on the QUIC dataset from [11] and fine-tuned on the app-category dataset from [1] or vice-versa, with the difference in accuracy being less than 1%. The 1% drop is a substantial improvement over an accuracy drop of ˜18% reported for a similar evaluation in [23] between a different pair of datasets.

Both works [22], [23] ignored short flows in their evaluations. The cutoff point for flow length is 100 packets in [23], while it is 300 packets in [22]. Both cutoff points substantially reduce the number of flows in the dataset. For example, the number of labeled flows from the Orange'20 dataset reported in [1] is 120K, whereas the number of flows considered in [22] from the same dataset is 3.7K, preserving only 2 out of 8 classes in the dataset.

Transformer-based architecture was employed for both flow-level and packet-level ETC in [25]. The model is pre-trained on a large volume of header and payload bytes in the same

fashion as a language model, i.e., by masking tokens and using them as labels. The approach was evaluated on 5 public and proprietary datasets with stellar results at both flow- and packet-level classification. The cost of the pre-training, e.g., number and capacity of GPUs, was not exposed.

Yang et al [26] explored the detection of zero-day applications in a large ISP-level commercial-grade dataset with very fine application classes. They conclude that both existing ML and DL methods perform well at detecting known applications, and propose a gradient-based technique that grants DL methods an advantage at performing unknown classes.

Our approach is distinguished from previous works in that 1) it focuses on ISP-level datasets, 2) keeps the SNI value masked in training and evaluation data, 3) evaluates the trained model across datasets, and 4) compares the few-shot model to a comparable classical model to show the advantages that each approach presents in practice. We treat an entire real-world dataset as the evaluation dataset, without filtering out flows (e.g., larger flows) to fit the requirements of the model. The closest works in the literature to ours are [2] and [26] as they both highlight the challenges of classifying traffic under realistic conditions at the ISP level. [2] does not consider few-shot models. In contrast with [26], we consider pure TLS datasets, whereas [26] reports that 55% of flows in their TCP traffic are encrypted. Furthermore, [26] focuses on fine-grained application-level classification in the presence of a large number of classes, while we primarily focus on app-category level classification across traffic traces.

Website and Mobile fingerprinting. Attacking the onion router (Tor) network using website fingerprinting (WF) was explored in a series of previous works [4], [5], [27], [28]. Herrmann et al [27] discovered that an array of packet sizes of the encrypted traffic can be leveraged to discover the websites visited by a Tor client using a simple Naïve Bayes classifier. Other classical ML models (e.g., kNN, support vector machine (SVM), and RF) were proposed for website fingerprinting in [29], [30] using the same side channel. Later on, [4] questioned the practicality of proposed attacks on Tor by investigating how four suggested attacks in the literature perform when trained and evaluated on separate datasets. They showed that the attack accuracy can drop from 62% to as low as 6% when the classifier is trained and evaluated across different Tor Browser Bundle (TBB) versions. They also demonstrated the effect of data drift by showing that the accuracy of a trained SVM-based classifier drops from around 80% to less than 50% in under 10 days. Countering this problem requires frequent retraining. With the goal of lowering the frequency of the required retraining, Wang and Goldberg [5] studied the effect of varying training data size on accuracy. They concluded that a kNN-based classifier can be an effective threat with a true-positive rate of 0.77.

Sirinam et al. [19] surveyed the assumptions made in previous works on Tor website fingerprinting attacks. They proposed a few-shot approach based on Triplet Networks to address the high bootstrap time of classifiers and to improve model transferability. They concluded that few-shot learning

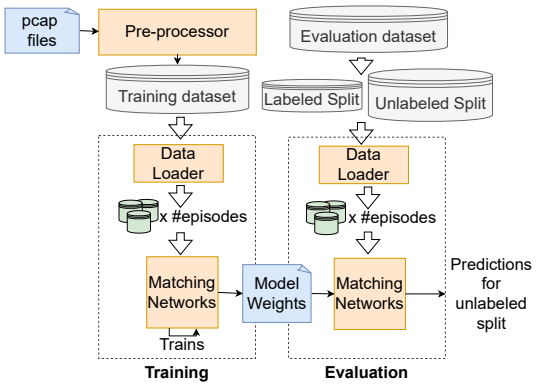


Fig. 1: The evaluated few-shot procedure

can reduce the frequency of retraining for website fingerprinting attacks against Tor. Flowprint [31], a mobile app fingerprinting approach, was introduced the next year and used temporal, device and destination-based features, including device IP addresses to fingerprint seen and unseen apps. The approach assumes traffic can be traced back to the device which is the case in client-side networks, but does not apply at ISP level. The works in this section highlight the difficulty of creating high-performing cross-dataset models, however, the use of client-side datasets in WF somewhat changes the problem context from our use case. For example, the proposed solution in [31] leverages information unavailable at ISP level. Therefore, we believe that traffic classification at the ISP level deserves a separate discussion.

III. Methodology

A. Matching Networks

We employ the few-shot procedure proposed in MN [7], the objective of which is to train a *learner* able to learn a new task fast, i.e., using a few samples. MN operates at two stages: (i) a training stage, in which the deep learning core is trained on data from a large number of tasks, and (ii) an evaluation stage, where the model uses the trained deep learner to project the evaluation data points into a latent space, and classifies them based on their similarity to a few labeled *support samples* by means of a nearest neighbor algorithm. The overall few-shot procedure is depicted in Figure 1.

Both training and evaluation are carried out in *episodes*, where an episode is a learning experience, and is equivalent to one step of training in traditional deep learning. The difference is that an episode subsamples the training classes as well as the data, whereas traditionally a batch of data subsamples the data and not the classes. As a result, in episode-based learning, the model does not need to be aware of the total number of classes of data in one episode. The number of classes that appear in each episode is determined by a constant *ways* parameter for the entire training or evaluation stage.

Another distinguishing property of an episode is that it features *query samples* as well as support samples. The model predicts the label of each query sample in each episode. The loss function value is computed based on a Softmax on

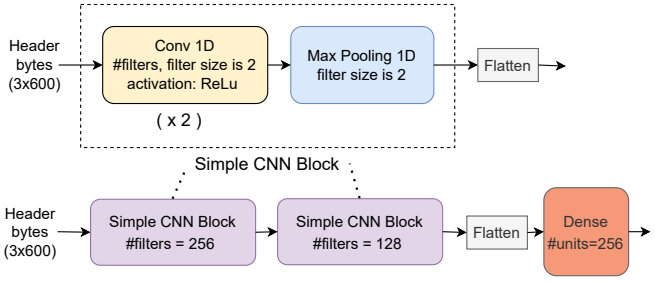
the similarities (or distances) of the predicted representation of each query sample to representations of support samples. Unlike a traditional classifier, in episode-based learning, the evaluation stage needs a number of labeled samples to serve as support samples for the nearest neighbor algorithm. These constitute the number of *shots* per class, and their number is typically small, hence the name *few-shot*. The number of training and query samples is equal for all classes and constant throughout an experiment, resulting in balanced training and test sets across classes. Therefore, we resort to accuracy, a reliable performance metric, in our experiments.

B. Embedding Function

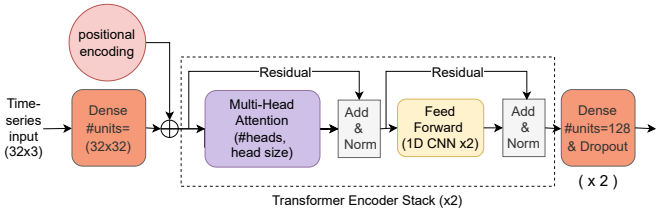
Matching Networks (MN) relies on a deep learning core, called the embedding function, to learn representations of data. The paper [7] suggests using two embedding functions, the first of which learns to represent individual data points. The second, called the Fully Contextual Embedding function (FCE) learns to represent groups of data, in order to rely both on individual data points and the *context* in which they reside. Throughout this paper, we use the term Embedding function to refer to the first embedding function, and the term FCE to refer to the second. While the FCE's structure remains the same as in MN's throughout our experiments, the embedding function's architecture is tailored to the input data type.

Our experiments showed that the architectures that are more successful as traditional classifiers are more effective as the embedding function. Indeed, CNN-based architectures have successfully been employed for ETC [1], [6], [11], [32] in the past. Therefore, for the header bytes input, we employ a CNN-based embedding function shown in Figure 2a, which is inspired by the header bytes classifier proposed to classify a real-life ISP-level dataset in [1]. We name our embedding function *Model U*, and call the inspiration the *UW Header* classifier. We manually searched through different CNN-based embedding functions before choosing Model U.

Our search followed a greedy methodology. The search was not exhaustive as there are too many combinations of parameter values to search through. Rather, we evaluated the performance of a pair of architectures with a given set of parameter values in each step of the search, where only one parameter value differed between the pair. We chose the best-performing architecture among the two to proceed to the next step, where another single parameter of the architecture was tweaked and evaluated. The tweaked parameters for the header bytes embedding function are the number of filters in the CNN block, the number of dense layers, the number of units in the dense layers, and the number of Simple CNN Blocks (cf., Figure 2a). In evaluating an architecture, we assess the performance of an MN model that uses the architecture as its embedding function on a benchmark evaluation dataset after training it on a benchmark training dataset. The benchmark training and evaluation datasets were non-overlapping small and large selections of the Jul'19 dataset introduced in [1] (cf., Section III-C).



(a) Model U: CNN-based embedding function for TLS header bytes



(b) Model W: Transformer-based embedding function for flow time series

Fig. 2: Embedding functions

As the embedding function for time series input, we employ *Model W* shown in Figure 2b. The architecture of Model W is borrowed from the Transformer [33], a sequence-to-sequence model featuring encoder and decoder stacks. Model W only uses the encoder part, which relies on an attention mechanism [34], residual connections, normalization, and a Feed Forward block. The input embedding and positional encoding proposed in [33] were also used in Model W. We implemented the input embedding using a dense layer and the Feed Forward block using 1D Convolution layers. The same search strategy for assigning values to Model U's parameters was used to determine the size and number of heads, encoder stacks, units in the dense layers, and output dense layers in Model W.

Throughout the experiments (cf., Section IV), traditional (i.e., non-few-shot) classifier baselines are needed. For the header bytes input, the UW Header classifier serves as a baseline as it is an established SoTA classifier and similar in architecture to Model U, with the exception of the number of Dense layers at the end. For time-series input, we designed a traditional classifier by feeding Model W's output to a Softmax layer. We name this classifier the *UW Transformer*.

C. Datasets and Preprocessing

1) Orange datasets: The Orange datasets are a collection of six datasets extracted from six packet traces collected from the mobile network of the Orange ISP. Each packet trace was captured in a one to two-hour session. The IP addresses and ports were anonymized, and the packets were cut beyond a certain point by the provider for privacy concerns. This point was chosen such that it allowed the inclusion of the TLS headers for the TLS flows but left out the application-layer payload. Further, we masked the SNI values after using them for labeling to prevent the models from relying on *Canary*

features. A total of six traces were collected across three years. Based on their month and year of collection, the traces and their corresponding datasets are called Jul'19, Sep'20, Apr'21, May'21, Jun'21, and Oct'21, respectively. The datasets were pre-processed into 8 app-category classes, including email, streaming, file download, chat, social, games, search, and browsing using the preprocessing method described in [1].

2) MLDIT dataset: The Multi-Label Dataset of Internet Traffic (MLDIT) [35] is a dataset of user digital behaviors over the Internet. A public sample of the dataset is released on Kaggle [36] and contains a large number of classes ordered according to a hierarchy. The hierarchy includes four levels from top to bottom, namely; application category, the specific application or website, browser, and action. The action includes items such as ImageBrowse and ConnectionWithAccount. The leaf level of the hierarchy contains 653 classes. The public sample of the dataset includes 2 to 5 pcap traces for each class. Each pcap trace captures the traffic generated by a specific user action. We use this public sample for experiments that require a large number of classes.

We preprocessed the MLDIT dataset to extract flow-level TLS handshake bytes. We broke each packet capture trace into flows using YAF, and stored each flow within a separate pcap file. The pcap file for each flow was then read using TShark and the TCP payload (which includes the TLS header) of the TLS handshake packets was extracted from each flow. We filtered out the flows that did not contain a handshake packet or an SNI value. The latter choice was an attempt to bring the data distribution as close as possible to the Orange datasets. As not all classes contain TLS flows that contain an SNI, after preprocessing, 26,724 samples and 338 classes remained in the dataset. This is the number of classes that contain at least 11 such TLS flows (i.e., with a handshake packet and an SNI value), where 11 is the minimum number of labeled samples needed for our experiment settings, i.e., the support and query data points needed in training and evaluation.

3) USTC-TFC: USTC-TFC2016 [17] is a network security dataset comprised of benign and malware traffic traces, each of which contains 10 classes of traffic. The bulk of the traces are non-encrypted, but the dataset is used as a benchmark for evaluating a few-shot traffic classifier in [16]. As such, we preprocessed this dataset to compare our approach to the work in [16]. A total number of 540,863 flows were extracted from the dataset. Most classes in the dataset did not contain TLS or SSL flows, with Shifu being the only malware class in which more than 1% of the flows were encrypted. Therefore, we only extracted the flow time-series data from this dataset.

4) Extracted Features: Our experiments use two sets of features extracted from the datasets:

TLS header bytes: The raw header bytes in the first three handshake packets in a TLS flow. If the flow contained at least one handshake packet, the raw handshake header bytes were extracted for up to three handshake packets, 600 bytes per packet. The SNI value was then masked by a fixed-length string in the raw bytes. The extracted bytes were padded by zeros, if necessary. If the flow did not contain handshake packets, as was the case for the J6 datasets because of a preprocessing choice, the first 600 bytes of the first three packets in the flow were extracted.

Flow time series: The sequence of packet sizes, packet inter-arrival times, and packet directions for each flow, i.e., a 32x3 array. We set the cut-off point to 32, as we found in our preliminary experiments with classical ML models that this cut-off point leads to a better performance than a larger one, e.g., 1024. The directions are coded by 1 and -1 for forward and backward packets, respectively.

TABLE I: Few-shot parameters (support, query and ways)

Section #	Training			Evaluation		
	#s	#q	#w	#s	#q	#w
IV-A	4,8	2	8	4,8	1	8
IV-B	4	2	5	4	1	2-10
IV-C	8	4	8	1-96	1	8

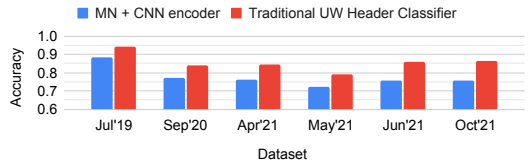
IV. Experiments

This section describes the experiments carried out to answer the questions enlisted in the introduction about a few-shot learner. The list of few-shot parameters for the experiments is provided in Table I.

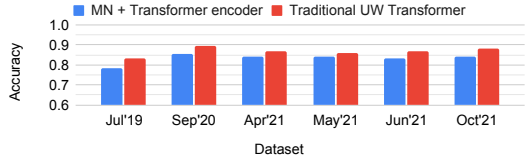
A. Few-shot Model as a Traditional Classifier

The experiments in this section answer the following question: How do the unique features of a few-shot learner, such as a metric-based loss function and the use of support samples in the evaluation phase, affect its performance compared to a traditional classifier, all else being equal? To this end, we evaluated the few-shot model in the same fashion as a traditional classifier; training it on 80% of each Orange dataset and evaluating it on the other 20% of the same dataset. MN was trained in 8-way episodes to keep the setup close to traditional learning. We performed two versions of the same experiment, one for each input feature. For raw header bytes, we use the CNN-based Model U as MN's embedding function. We compare the performance of the CNN-based embedding function to that of the UW Header classifier, a traditional classifier with the same architecture except for two additional dense layers. On the other hand, for time-series input, we use the transformer-based Model W as MN's embedding function and compare its performance to that of the UW Tranformer, a traditional classifier with the same architecture as Figure 2b, followed by an additional 8-unit dense layer and a Softmax function, as described in Section III. Figure 3 depicts the performances of the four models in terms of accuracy on all Orange datasets.

Figure 3 shows that a traditional classifier consistently outperforms the few-shot classifier when trained and evaluated on the same dataset. The inferiority of MN in these experiments could be the result of incorporating a similarity metric into the cost function rather than gradually reducing the dimensions by non-linear dense layers, or it could be the result of the support samples' bias in the evaluation phase. Furthermore,



(a)



(b)

Fig. 3: (a) Performance of CNN-based MN on the header bytes of the Orange datasets compared to that of the UW Header classifier; (b) Performance of Transformer-based MN on time-series data from the Orange datasets compared to the UW Transformer

Figure 3 shows that when trained and evaluated on the same dataset, MN shows a remarkably better performance using the Transformer-based embedding function compared to the CNN-based, on all datasets with the exception of Jul'19, which we believe to be a special case[1]. For the rest of the datasets, the transformer-based MN yields an accuracy between 83% and 85%, which is remarkably higher than the 72% to 77% accuracy onserved for Model U in Figure 3a. On a similar note, the traditional UW Transformer outperforms the traditional UW Header classifier on all but the Jul'19 dataset, with performances between 86% to 89% compared to the 79% to 86% accuracy of the UW Header classifier. We suspect that header raw bytes may be too noisy in the presence of encryption, hence the superiority of models relying on time series data.

B. New Tasks from Same Dataset

The experiments in this section test the following hypothesis: The few-shot model offers an advantage when trained on diverse tasks, i.e., tasks with different sets of classes, as opposed to training on similar tasks, i.e., tasks that feature different data points from the same 8 classes, as was the case in Section IV-A's experiments. We leverage the large number of classes in the MLDIT dataset to create diverse tasks in these experiments. We carried experiments on raw header bytes input only, as the flows in the MLDIT dataset were too short for flow time-series data to be informative, with a median of 4 packets per flow.

We trained MN with Model U as embedding function on 338 randomly chosen classes and reserved the remaining 50 classes as *unseen* for evaluation. This resulted in 21,892 training samples and 4,832 test samples. A small split of data from the seen classes was reserved for the evaluation of the model's performance on seen classes (depicted in Figure 4) and the model was trained on the rest. We report the results for both seen and unseen classes, increasing the difficulty of the evaluation tasks incrementally from 2-way to 10-way classification. Figure 4 shows the results of these experiments.

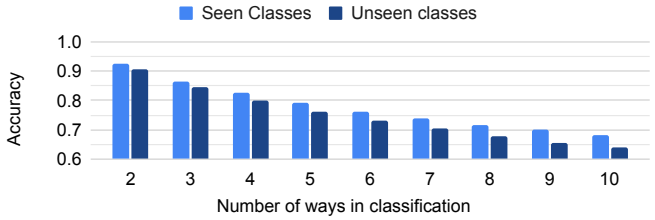


Fig. 4: Performance of MN on the MLDIT dataset's header data input

As evident in Figure 4, MN's performance when trained on diverse tasks seems to be similar to its performance on uniform tasks. The 8-way performance (which is comparable to the experiments in the previous section in terms of the number of classes in the task) performs at 72% accuracy, similar to what we saw for the May'21 dataset in Figure 3a. Furthermore, the performance of MN on unseen classes is surprisingly close to its performance on seen classes, with the gap being less than 2% in accuracy for binary classification, which widens gradually as the number of ways increases and reaches a maximum 4% in 10-way classification. One reason for the closeness of the seen vs. unseen performances could be that the nearest neighbor algorithm is carrying the load of the performance rather than the CNN-based feature extractor.

Despite being a good fit for few-shot models in terms of samples to class ratio, to the best of our knowledge the MLDIT dataset has not yet been used in the few-shot literature even though it is publicly available. Hence, to provide some perspective into how our model performs compared to the few-shot models suggested in the literature, we tested our model on USTC-TFC [17] the reference dataset in [16]. We switched to our Transformer-based MN to experiment on USTC-TFC time-series data as its TLS portion was slim. As a traditional classifier, our approach obtained 99% accuracy on a binary benign vs. malware classification task. The work in [16] reports binary classification results for seen vs. unseen class experiments where 12 out of the 20 classes in the dataset are used in training and 8 in evaluation. Their best achieved accuracy is 96%. The closest point of comparison we can offer is our 2-way classification results on unseen classes on the MLDIT which performed at 90% accuracy.

The MLDIT proved a tough dataset to classify when the SNI values are masked. This was corroborated when the UW Header classifier obtained an accuracy of 57.7% on MLDIT for app-category level classification (i.e., a 9-way classification task). In comparison, on USTC-TFC's 20-way classification task, UW Transformer obtains 87% accuracy and as a traditional classifier, MN obtains 73% accuracy using 8 support samples per class. This convinced us that MLDIT headers

[1]Since the Jul'19 dataset was collected in 2019 we believe the headers might have been prone to open-text attacks, as our work in [2], [6] explains.

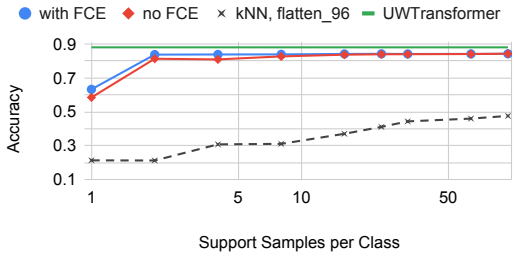
present a harder classification task than USTC-TFC time series, therefore our few-shot model performs well enough to be the subject of this investigation.

C. Generalizability

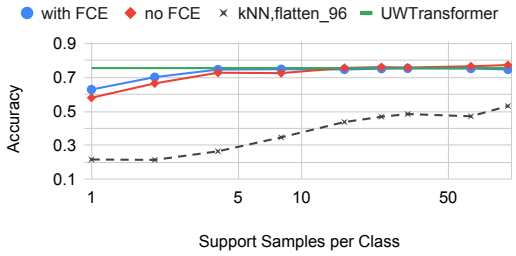
The experiments in this section put the generalizability of MN to test and answer the following questions. First, given its use of support samples from the evaluation dataset, is a few-shot model more apt at generalizing to new datasets than a traditional model? Second, how many support samples are required from the evaluation dataset for the few-shot model to perform optimally? To answer these questions, we train the few-shot model on the first of the Orange 2021 traces, namely Apr'21, and evaluate it on the rest of the Orange datasets as well as Apr'21. Thus, the classes that the model sees in training and evaluation are similar and the only difference between the pair of training and evaluation datasets is time of collection. We chose the Apr'21 dataset as the training dataset to see the effect of gradually increasing the disparity of training and evaluation tasks, as the disparity increases by an increase in the gap between the training and evaluation datasets' collection time.

We use the Transformer-based few-shot model in these experiments, both because it was the more successful few-shot model in Section IV-A and because time-series data is arguably preferable to raw handshake bytes for production use. We trained two versions of the model, one with the FCE and the other without the FCE, to see whether the FCE helps the performance. We compare the performance of the few-shot model on a nearest-neighbor algorithm configured closely to that of MN's, i.e., the kNN algorithm considers all the support set as neighbors and weighs the neighbors based on the inverse of their Cosine similarity to the query point. The three channels of the time-series data, i.e., packet sizes, packet inter-arrival times, and directions, had to be flattened and fed to the nearest neighbor algorithm as a 96-dimensional vector because kNN cannot deal with non-vector input.

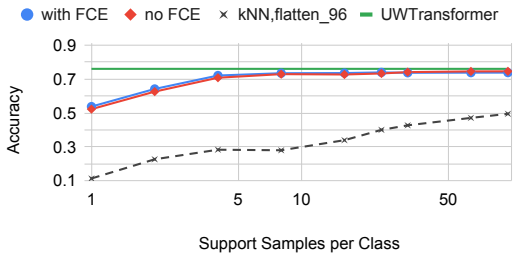
Figure 5 shows the performances of a few-shot model trained on the Apr'21 dataset and evaluated on all the 2021 Orange datasets, using 1 to 96 support samples per class. The dashed line shows the performance of a kNN algorithm, using the same number of support samples. The points in the figures show the average of three trials using three different support sets for each algorithm. Given the similarity of the kNN to the metric-based classifier in MN, the figure confirms that the embedding function does contribute to the performance. Interestingly, comparing the figures for different datasets shows that the closer the data distribution in the evaluation dataset is to the training dataset, the few-shot model relies on fewer data points to reach its optimal performance. It seems that the sweet spot for yielding the best performance is between 4 to 8 samples per class for the May'21 and Jun'21 datasets, but it is between 16-32 samples for the Oct'21 dataset. The graphs plateau beyond a certain point, both for the few-shot model and the kNN, which indicates that the support sample is representative of the entire dataset at that point.



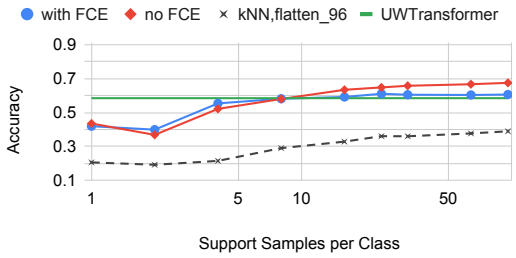
(a) Apr-trained and evaluated on Apr'21



(b) Apr-trained and evaluated on May'21



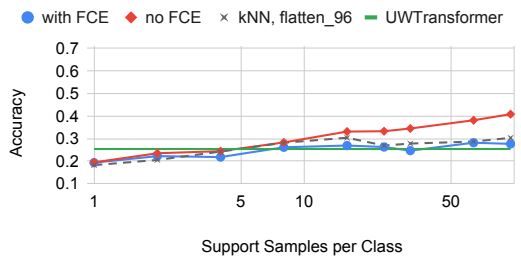
(c) Apr-trained and evaluated on Jun'21



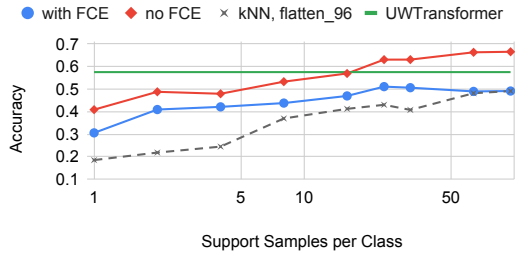
(d) Apr-trained and evaluated on Oct'21

Fig. 5: Cross-dataset evaluation of the Transformer-based few-shot model on Orange 2021 datasets

Another interesting takeaway has to do with the traditional deep learning baseline. We trained the UW Transformer on the Apr'21 dataset and evaluated it on the rest of the Orange datasets to find out whether there is a practical advantage to using a few-shot learner. The performance in terms of accuracy is shown as a horizontal green line in Figure 5. Note that the UW Transformer does not use any evaluation support samples by design. We show the line in the figures to showcase the number of support samples per class at which the few-shot model outperforms the traditional deep learner (if any). While the traditional deep learner outperforms the



(a) Apr-trained and evaluated on Jul'19



(b) Apr-trained and evaluated on Sep'20

Fig. 6: Evaluation of Transformer-based MN on datasets far from the training dataset

few-shot model when the training and evaluation datasets are close, its evaluation performance gradually drops lower as the two datasets diverge in collection time. When evaluated on the Oct'21 dataset, the performance drop of the UW Transformer is large enough such that the few-shot model outperforms it by 5% using 16 labeled samples per class, and by 10% at its best, as shown in Figure 5d.

To confirm whether the few-shot model is indeed more generalizable than its traditional deep learning counterpart, we show the performance of the April-trained few-shot model on the Jul'19 and the Sep'20 datasets, two datasets collected 22 and 7 months apart from the training dataset, respectively. Figure 6 shows the results for these datasets, demonstrating that using 16 support samples or more, the few-shot model outperforms both the traditional deep learner, and the kNN.

V. Discussion

Our expectation, based on the premise of learning *how to classify* rather than a *specific classification task* advertised by the few-shot meta-learning literature, was somewhat different from what we observed in our experiments. We did not see evidence of the model learning meta-relationships between classes and instances, rather, our takeaway was that the deep learning core acts as a feature extractor, i.e., it learns a representation of the data based on the features it finds important in the training data, and the nearest-neighbor classifier guesses the closest class to each test sample based on the labeled data sampled from the evaluation dataset and the learned feature extraction method. Hence, classification largely relies on the quality of the feature extraction, which is biased by the training data distribution.

Dataset-specific performance is a known pitfall of deep learners, which the meta-learning literature tries to mitigate. MN appears to rely on a simple embedding function and support samples from the evaluation dataset to mitigate the bias. The latter seems to be effective to some extent; as we saw in Section IV-C, MN generalizes better to datasets moderately far apart from the training dataset than a deep learner. However, as the training and evaluation distributions diverge, the learned feature extraction method quickly becomes irrelevant, and MN's performance falls below the acceptable threshold for an effective classifier (cf., Approx. 40% accuracy in Figure 6a). In fact, when performing experiments similar to Section IV-C from an MLDIT-trained MN to Orange datasets, we found that MN performs slightly worse than a 1-NN algorithm using the same 1800-dimensional header byte input. The learned representations from MLDIT were counter-productive when applied to Orange datasets. Although, from a production standpoint, a model may not be required to be portable between datasets as far apart as MLDIT and Orange.

Role of the FCE. We did not highlight the role of the FCE in this paper, because we found its role highly scenario-dependent. The experiments in Section IV-C show that the FCE either does not make a difference or hampers the performance when training and evaluation distributions diverge. However, in our CNN-based experiments we found that the FCE helps the performance, although it is better discarded in the evaluation phase when data distributions diverge. We attribute the gain in performance to the LSTM-based FCE capturing sequential data patterns, rather than learning *class contexts* which was the inspiration behind its design choice in [7]. We did not see any evidence of it affecting the performance in Section IV-B's seen vs. unseen experiments, which is where it should have made a meaningful difference if it learned class-specific patterns. Note that we did not report the no-FCE results in Section IV-B for the sake of brevity. We conclude that if the embedding function is capable of capturing sequential patterns, as is the case for transformer encoders, the FCE can be safely discarded.

Selection of evaluation support samples. The outcome of the evaluation stage varies significantly based on the selection of support samples, especially when support sets are small. For example, the standard deviation among the three evaluation trials in Section IV-C is in the range of 1 to 6% for 4-shot classification but can be as high as 15% for 1-shot classification. Resampling the support set for each evaluation episode could potentially yield a more representative performance of the model. However, in practice when labeled samples are few, the same set of labeled samples will have to act as the support set for the entire classification task. We report the average performance over three randomly chosen support sets to simulate the practical constraint, while keeping the reported performances representative.

VI. Conclusion

Based on the state of the art in few-shot learning and traffic classification, in this work we designed a few-shot classifier and showed its merit as a traditional classifier in experiments where training and test data were sampled from

the same distribution, as is typical in the traffic classification literature. Further, in order to show its performance as a few-shot classifier, we trained and tested it on the MLDIT dataset, a dataset that we view as the network traffic equivalent of Omniglot [37], as it features a few traffic traces per class for a large variety of traffic classes. Few-shot meta-learning was originally proposed as a solution to the challenge of classifying such datasets. We further put the effectiveness of the deep feature extractor to test, by performing cross-dataset experiments and comparing the generalizability of the approach to both a deep learner and a classical kNN.

Our findings put the effectiveness of the premise of automatic feature extraction in deep learning to question. We conclude that as far as ETC is concerned, cross-dataset generalizability of the extracted features remains an open research problem, which has been overlooked so far due to same-dataset evaluation of proposed models. Given the constraints of traditional deep learning, we believe reinforcement learning (RL) or explainable AI (XAI) may offer a solution to the challenges encountered by few-shot models. RL is believed to be capable of learning patterns beyond the statistics of the input datasets, while XAI offers a means to find out the features a model relies on, thus making it possible to mask dataset-specific features at the preprocessing stage.

REFERENCES

[1] I. Akbari *et al.*, "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, 2021.

[2] N. Malekghaini *et al.*, "Data drift in dl: Lessons learned from encrypted traffic classification," in *2022 IFIP Networking Conference (IFIP Networking)*. IEEE, 2022, pp. 1–9.

[3] F. Bronzino *et al.*, "Inferring streaming video quality from encrypted traffic: Practical models and deployment experience," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–25, 2019.

[4] M. Juarez *et al.*, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 263–274.

[5] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting." *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 4, pp. 21–36, 2016.

[6] N. Malekghaini, E. Akbari *et al.*, "Deep learning for encrypted traffic classification in the face of data drift: An empirical study," *Computer Networks*, vol. 225, p. 109648, 2023.

[7] O. Vinyals *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

[8] J. Snell *et al.*, "Prototypical networks for few-shot learning," *Advances in neural information processing systems*, vol. 30, 2017.

[9] F. Sung *et al.*, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 1199–1208.

[10] G. Aceto *et al.*, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[11] S. Rezaei *et al.*, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2019.

[12] N. Malekghaini *et al.*, "AutoML4ETC: Automated neural architecture search for real-world encrypted traffic classification," *arXiv preprint arXiv:2308.02182*, 2023.

[13] V. F. Taylor *et al.*, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2017.

[14] C. Finn, "Stanford cs330: Deep multi-task meta learning," 2022, accessed on 2023-12-01. [Online]. Available: https://cs330.stanford.edu/

[15] N. Bendre *et al.*, "Learning from few samples: A survey," *arXiv preprint arXiv:2007.15484*, 2020.

[16] J. Guo *et al.*, "Global-aware prototypical network for few-shot encrypted traffic classification," in *2022 IFIP Networking Conference (IFIP Networking)*. IEEE, 2022, pp. 1–9.

[17] W. Wang and D. Lu, "USTC-TFC2016," 2019. [Online]. Available: https://github.com/yungshenglu/USTC-TFC2016

[18] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2. Lille, 2015.

[19] P. Sirinam *et al.*, "Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1131–1148.

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

[21] E. Horowicz *et al.*, "A few shots traffic classification with mini-flowpic augmentations," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 647–654.

[22] M. S. Towhid and N. Shahriar, "Encrypted network traffic classification in sdn using self-supervised learning," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*. IEEE, 2022, pp. 243–245.

[23] S. Rezaei and X. Liu, "How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets," *arXiv preprint arXiv:1812.09761*, 2018.

[24] ——, "Multitask learning for network traffic classification," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2020, pp. 1–9.

[25] X. Lin *et al.*, "Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 633–642.

[26] L. Yang *et al.*, "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, 2021.

[27] D. Herrmann *et al.*, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, 2009.

[28] A. Khajehpour *et al.*, "Deep inside tor: Exploring website fingerprinting attacks on tor traffic in realistic settings," in *2022 12th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2022, pp. 148–156.

[29] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *23rd USENIX Security Symposium*, 2014, pp. 143–157.

[30] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1187–1203.

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

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

[33] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[34] D. Bahdanau *et al.*, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[35] W. Li and G. Quenard, "Towards a multi-label dataset of internet traffic for digital behavior classification," in *2021 3rd International Conference on Computer Communication and the Internet (ICCCI)*. IEEE, 2021, pp. 38–46.

[36] W. Li, "Multi-label dataset of internet traffics (mldit)," Dec 2020. [Online]. Available: https://www.kaggle.com/datasets/artemis1216/multi-label-dataset-of-internet-traffics

[37] B. M. Lake *et al.*, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.