Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling

Peng Wang[a], *Zhijie Tang*[b,*], *Junfeng Wang*[c]

[a] College of Cybersecurity, Sichuan University, Chengdu 610065, China
[b] College of Software Engineering, Sichuan University, Chengdu 610065, China
[c] College of Computer Science, Sichuan University, Chengdu 610065, China

ARTICLE INFO

Article history:
Received 12 December 2020
Revised 30 March 2021
Accepted 5 April 2021
Available online 18 April 2021

Keywords:
Malware classification
Few-shot learning
Multimodal distribution
Multi-prototype
Infinite mixture prototypes

ABSTRACT

New malware variants appear rapidly and continuously increase the difficulty to classify malware into correct families. This brings two challenges for malware classification: The first is the scarce samples problem, where collecting a large volume of a newly detected malware family to train a classifier can be extremely hard and it is unavoidable to suffer from overfitting using a small number of samples. The second is the dynamic recognition problem. Most widely adopted classifiers are trained on predefined known malware families, lacking ability to incrementally identifying novel families, which require to retrain from scratch. To tackle these challenges, in this study, we employ meta-learning based few-shot learning (FSL) technique and propose a new few-shot malware classification model called SIMPLE (Supervised Infinite Mixture Prototypes LEarning). With the help of meta-learning, SIMPLE is trained with predefined malware families and can maintain its ability to classify novel malware families that has never met. Furthermore, the prior knowledge learned via meta-learning can prevent from overfitting caused by scarce samples. Our proposed SIMPLE introduces multi-prototype modeling to generate multiple prototypes of each family to enhance the generalization ability, based on API invocation sequences from dynamic analysis. This is inspired by the observation that behaviors within the same family often match multiple subpatterns and satisfy multimodal data distribution. In the broad experiments, SIMPLE achieves state-of-the-art few-shot malware classification performance and outperforms all the baselines. With only 5 samples per family, SIMPLE reaches very high accuracy of 90% in 5-way classification task on novel malware families, which substantially solves the problem of scarce samples and dynamic recognition. We also make analysis on the reason of effectiveness with multi-prototype and fast adaption feature to provide more interpretability for the results.

1. Introduction

Emerging malware families are becoming major threats to modern cybersecurity and damages caused by variants of malware are surprisingly enormous every year. It is reported by axia[1] that notorious ransomware WannaCry knocked out 100 thousand systems worldwide and costed $5 billion in 2017. Traditional signature-based analysis relies on expert efforts to generate hand-designed signatures to identify malware,

* Corresponding author.
E-mail address: asichurter@gmail.com (Z. Tang).
https://doi.org/10.1016/j.cose.2021.102273

[1] https://www.ixiacom.com/resources/2018-security-report.

which seems to be impossible as new variants emerge so quickly using polymorphic or metamorphic techniques. Machine learning methods are becoming the mainstream trend to automatically complete signature generation and malware identification. This is because they self-learn malware features by end-to-end training without human interference and expert knowledge, which is robust and convenient in application scenarios (Ye et al., 2017). Particularly, deep learning based models are appealing due to their high accuracy in malware classification and some powerful structures such as Convolutional Neural Network(CNN) and Recurrent Neural Network(RNN) have been proven very effective (Bhodia et al., 2019; Huang et al., 2019; Kang et al., 2019; Vasan et al., 2020; Vu et al., 2019). Most of these methods can be viewed as a three-stage model, which comprises a data preprocessing module, a feature extractor and a classifier.

However, these models have some common drawbacks when applied to newly-arising malware families. They are not applicable to recognize unseen families not existed in the training dataset, which is constrained by the fixed parameters in their classifier modules after training. In such a situation where new variants are continuously developed by malware authors, it is vital to keep active on these new malware and dynamically build corresponding classifiers from collected samples, which is depicted as "Open Set Recognition Problem" (OSR). Another serious problem potentially encountered in practice is scarce malware samples, where each family lacks sufficient samples for training a data-hungry classifier. This problem is actually in two folds: First, scarce samples may lead to severe overfitting for these machine learning models especially for high-dimensional parameters; Second, deep learning models can often oscillate drastically and fail to converge when scarce samples are fed. However, collecting large volume of samples requires great efforts and is unpractical to quickly handle new malware families. After the attack of some zero-day malware, quick response is crucial to stop the wide spread of the novel malware to minimize the loss under the circumstance that only a few samples of the family are in hand. Existing conventional methods are usually incapable of meeting these strict requirements of recognizing novel malware family using a handful of samples dynamically.

In order to solve these two problems, we introduce meta-learning based FSL techniques and propose a new few-shot malware classification model called SIMPLE. Meta-learning framework enables SIMPLE to learn some high-level prior knowledge about malware (actually the embedding module) and build task-specific classifiers for different malware families from a few samples. It uses API invocation sequences as input features, which is an effective dynamic analysis approach adopted in other researches (Huang et al., 2019; Kang et al., 2019; Ki et al., 2015; Kim et al., 2019; Ndibanje et al., 2019; Shijo and Salim, 2015). All the API sequences are first embedded by a pre-trained word embedding and then modeled by Long-Short Term Memory Network (LSTM) (Kang et al., 2019) to reserve sequential information. Then, malware families are represented by multiple prototypes from clustering, rather than a sole prototype adopted in other baseline few-shot models (Snell et al., 2017). This is inspired by the viewpoint in Huang et al. (2019) that variants from the same family can exhibit distinct behaviors and a malware family may involve several similar traces. Also, the number of clusters need not to be determined in advance and the model capacity can be inferred adaptively from task data to balance both simple and complex data distribution, which is called infinite mixture prototypes (IMP) and first used for FSL in Allen et al. (2019). Furthermore, we fuse supervised label information into clustering processing to make more intelligent split decision resulting more proper family prototypes. Comparing to vanilla IMP, SIMPLE is simpler to implement and requires for no extra parameters. SIMPLE also achieves better accuracy than all the baseline models in all experimental settings and shows state-of-the-art few-shot malware classification performance.

Our contributions in this paper are:

- Addressing scarce malware samples and dynamic malware family recognition problem novelly with the help of meta-learning. More than 90% classification accuracy is achieved on unseen families when providing a few samples, and only one-time forward is needed during inference, free from time-consuming fine-tuning.
- Proposing a brand-new model called SIMPLE to generate multi-prototypes by supervised clustering, which is the first to use flexibly generated prototypes to handle complex multimodal distribution in malware datasets. Prototypes are intelligently inferred from given data to mitigate overfitting and reduce resource consumption.
- Conducting broad experiments to show SIMPLE achieves state-of-the-art few-shot malware classification performance. SIMPLE outperforms all the baselines including both traditional dynamic malware classification approaches and existing few-shot classification models, benefiting from both meta-learning and multi-prototypes.

2. Related work

2.1. *Malware classification*

Classification problems of malware can be divided into malware detection that decides a sample is malware or not and malware classification that outputs a sample's family label. Although these two have different goals, their methods are interchangeable to some extent and we mainly focus on the latter one. Static and dynamic analysis are two categories of malware analysis and have their advantages and disadvantages respectively. Static analysis often uses byte sequence of binary files, assembly sequence and opcodes without executing suspicious files and build machine learning classifier upon these features. Santos et al. (2009) used n-gram to generate malware signatures and k-nearest-neighbors (KNN) as classifier. Ma et al. (2019) used assembly opcode to align with raw byte sequence to compose sequence attention. Some work focus on assembly opcode to construct features like n-gram and then apply several classifiers. Differently, Bhodia et al. extracted meta-data such as header size and import information from PE files as malware features (Bhodia et al., 2019). Using entropy of different sections of PE files as input feature has also been explored by Lyda and Hamrock (2007).

Instead of using expert knowledge to extract features, malware image poses another direction of static methods. It was first proposed by Nataraj et al. (2011) that converts malware bytes to image and applies some image processing techniques like GIST descriptor. Some researchers employed deep learning structures such as CNN to process malware images and trained classifier end-to-end (Bhodia et al., 2019; Vasan et al., 2020; Vu et al., 2019). Recently, some novel image conversion methods emerged like Simhash combined with assembly opcodes to make malware images (Lim et al., 2020). However, static analysis can be misled by obfuscation techniques (packing, transformation, compressed) and it can not identify variants of previous malware when encryption, packing, polymorphism, obfuscation, meta-morphism techniques are used (Ye et al., 2017).

On the contrary, dynamic analysis often executes binary files in a virtual environment or sandbox to collect run-time features, such as API invocation sequence or API graph. It can reflect real intent of executed ones and naturally self-unpack packed malware when executing. API invocation histogram is a very simple statistical feature, which is adopted by Ndibanje et al. (2019). Also, there are some work trying to use bag-of-word (BOW) to model API information and represent each malware as a binary boolean vector of API dictionary (Shijo and Salim, 2015). For the sake of keeping sequential information, sequence alignment methods are also proposed to extract common motifs among API sequences and compute similarity based on them (Ki et al., 2015; Kim et al., 2019). Using Multiple Sequence Alignment (MSA) combined with Longest Common Sequence (LCS) algorithm is a frequently mentioned approach (Cho et al., 2014). Constructing API graph from sequence is a promising direction (Ki et al., 2015; Park et al., 2010), but it is NP-hard to match API graph that makes it inefficient to process. Some other methods include using association rule (Ravi and Manoharan, 2012), focusing on special type behavior such as registry and file accessing related invocation patterns (Du et al., 2019) and etc. Without involving manually designed features, deep learning architectures like LSTM are able to extract useful features from API sequences by themselves (Kang et al., 2019) and trained end-to-end. Some advanced models like BERT (Devlin et al., 2019) and Sent2Vec (Pagliardini et al., 2018) are also exploited to enhance contextual understanding of malware (Huang et al., 2019). Dynamic analysis can be accurate normally, but it can be evaded by some anti-virtual, anti-sandbox or triggered malware, which results in incomplete behavior coverage and invalid analysis.

2.2. *Few-shot learning*

The concept of FSL was first proposed by Fei-Fei et al. (2006) that targets at building classifier with considerably high accuracy using a few samples. According to human experience, quickly learning about new concepts requires for a lot of prior knowledge acquired from past learning and there are many kinds of learned prior in different models, such as common embedding function (metric-based) (Snell et al., 2017), optimization steps (Ravi and Larochelle, 2017), fast adaption ability (Finn et al., 2017) and etc. Most few-shot models are trained via meta-learning framework in a learn-to-learn manner. Concretely, meta-learning samples different but similar tasks(mini-batches) from a latent task distribution formed by the training set and performs update on a distinct task at each iteration. This forces model to learn some general and high-level knowledge appropriate for all tasks and complete classification from mini-batch to mini-batch. Among FSL models, metric-based models have been a hot topic these years because they are easy to implement and often achieves very good results. Matching Networks (Vinyals et al., 2016) first formalize the metric-based methods, which use separated contextual embedding function (FCE) to embed and attention kernel to classify samples. Prototypical Networks (Snell et al., 2017) regard mean of samples within each class as the class prototype to represent each class. Induction Networks (Geng et al., 2020) use dynamic routing algorithm to induce class prototypes to solve text classification problems. Hybrid Attention-Based Prototypical Networks(HABPN) (Gao et al., 2019) leverages attention mechanism to weight samples within class and feature feature dimension to make comparison "query-dependent". ConvProtoNet (Tang et al., 2020) designs an independent convolutional module to extract class prototypes in order to handle multimodal distribution. FEAT (Ye et al., 2020) takes task adaption into consideration and use a set-to-set function to yield more discriminative and task-specific embeddings, which has a similar idea to Li et al. (2019); Oreshkin et al. (2018). Apart from these above, IMP (Allen et al., 2019) is the first to introduce multi-prototype modeling to solve multimodal distribution in FSL which adopts an unsupervised algorithm called DP-means (Kulis and Jordan, 2012) to generate clusters.

Fast adaption poses another promising direction of FSL and was first proposed by MAML (Finn et al., 2017). It aims at adapting a new task by a few optimization steps using a small number of samples, referred as inner loop. Outer loop is running by optimizing the meta loss after adaption during meta-learning stage, which requires for computing high-order derivations. Following this idea, ATAML (Abbasi et al., 2019) tries to separate task-specific from task-agnostic parameters and Layerwise MAML (Antoniou et al., 2019) emphasizes the importance of using layer-wise learning rate and normalization statistics. Reptile (Nichol et al., 2018) attempts to use first-order approximation to accelerate MAML without accuracy loss.

2.3. *Few-shot malware classification*

FSL has been explored for malware classification after detecting new variants as soon as revealing a tiny amount of samples due to strong scalability and versatility. To our best knowledge, Tran et al. (2020) first proposed to apply FSL techniques to malware classification, where they used a existing FSL model called MANN (Santoro et al., 2016) without modification. Another line of work is from Tang et al. (2020) where they considered the multimodal distribution of malware and used a deep CNN to extract class prototypes. Both of them achieved great results and demonstrate the feasibility of meta-learning in malware classification. However, there is still space to improve the performance for such a security-relevant topic as malware classification where accuracy is still below 90% in 5-way setting. In this paper, we will continue the work of

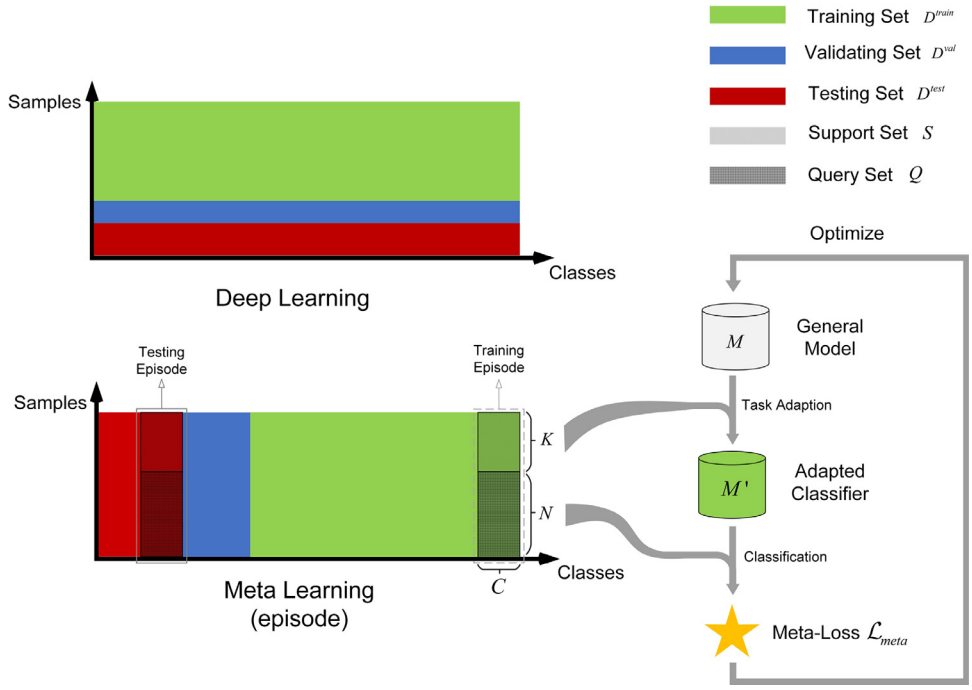


Fig. 1 – The episode-based meta learning framework for few-shot malware classification.

(Tang et al., 2020) to address the inadequacy of unimodal modeling but directly use multiple prototypes to describe the multimodal distribution, where prototypes are built from supervised information brought by small amount of samples. With multi-prototype modeling, we increase the accuracy of few-shot malware classification using SIMPLE model to make it more applicable.

3. Preprocessing and feature extraction

3.1. *Dataset split*

To make more comparable results with other existing models and baselines, two publicly accessible malware datasets were used for evaluating SIMPLE: VirusShare_00177 dataset and APIMDS, more details about these two datasets are depicted in Appendix A. For meta learning, dataset split is kind of different from traditional deep learning approaches, where we split classes(families) within the dataset into a training set D^{train}, a validating set D^{val} and a testing set D^{test} rather than splitting samples within each class, shown in Fig. 1. This is because meta-learning aims to perform well on unknown classes(families) instead of unknown samples of known classes(families). After splitting, these three subsets are ensured to have disjoint families and only families in D^{train} are "known" families (also called base families, families in D^{test} are called novel families respectively). D^{train} is used to optimize model parameters, D^{val} is used for monitoring training process and saving models with best generalizing performance and D^{test} is used for generating final results.

Table 1 – Configuration of Cuckoo Sandbox running dynamic analysis.

Name	Configuration
Host OS	Ubuntu 18.04 LTS
Host Hardware	Intel Core i7-9700F 3.0 GHz, 16 GB Memory
Host Python Version	2.7.17
Guest	Virtual Box 6.1.4, Windows 7, 4 GB Memory
Guest Python Version	2.7.13
Cuckoo Sandbox Version	2.0.7

3.2. *API sequence extraction*

To monitor and record API invocation sequence, we use Cuckoo sandbox[2] to execute malware in a virtual environment. Detailed setup for Cuckoo sandbox is listed in Table 1. Cuckoo sandbox is executed on host machine and manages the guest machines to analyze malware. A customized reporting module written in Python is used after analysis to generate JSON reports that only contain API invocation sequences to reduce disk space consumption and default common Cuckoo running setting is adopted during analysis. For the analysis report generated by Cuckoo, we only leave the API name while drop the parameters and returned value of each API invocation. For the malware launches several processes, API invoked by different processes are concatenated in the order of their

[2] https://cuckoosandbox.org/.

creation to make the API sequence linear. Note that we discard malware samples whose sequence length is less than 10, since too short sequence usually indicates the failure of execution (Kim et al., 2019). Also note that only VirusShare_00177 dataset is running in the Cuckoo sandbox but not APIMDS dataset, because the samples are originally API sequences in APIMDS but these in VirusShare_00177 are binary executables.

3.3. API preprocessing and word embedding

It is of great importance to do some preprocessing to remove redundant features and only keep these most discriminative features after obtaining API sequences. When the analysis is on, it is very common that a program will repeatedly call some identical APIs when doing some file-related work or executing loop and these identical invocation subsequences can be thought to be redundant. Thus, we substitute these long API subsequences where an identical API appears more than twice successively with a short subsequence of two calls. Similar preprocessing is also adopted in Ndibanje et al. (2019).

We also adopt N-gram from raw API sequences to enhance local feature representation, which is reported to be effective in Tran and Sato (2017). Besides, using N-gram can increase the scale of sequence's word dictionary, from m to m^n if there exists m unique APIs in the original sequences. It can significantly enhance the diversity of sequence expression using n-gram sequences. Furthermore, we calculate the Term Frequency - Inverse Document Frequency (TF-IDF) of each N-gram combination on the whole dataset as (Tran and Sato, 2017) and only keep these N-gram items of top-l TF-IDF value in the sequence to reduce overall sequence length:

$$TF-IDF(x_i) = TF(x_i)\cdot IDF(x_i) = \frac{n_i}{\sum_j n_j}\cdot \log\left(\frac{n}{f(x_i)+1}\right) \tag{1}$$

where n_i denotes the times that ith N-gram appears and $f(x_i)$ is the count of malware samples containing ith N-gram item. After filtering, all the N-gram items are replaced with their corresponding index in the dictionary. We truncate some long sequences and only use first t items of N-gram invocation after applying n-gram and TF-IDF for matrix alignment and calculation consideration. The impact of truncation is investigated in Section 5.3.5.

Integer API N-gram sequences are usually embedded by an embedding matrix and output vector sequences, known as embedding phase. Inspired by word embedding employed in NLP, we initialize the embedding matrix by pre-trained word embeddings rather than random initialization to give semantic meaning to word vectors. Some previous work also adopted word embedding to provide a more reasonable initial embedding (Tran et al., 2020), but differently we use GloVe (Pennington et al., 2014) instead of Word2Vec to emphasize co-occurrences. It turns out that using word embedding can promote model training and accelerate training process, compared to random initialization.

4. Few-shot multi-prototype modeling

4.1. Definition of few-shot malware classification

The goal of few-shot malware classification is to use a few labelled malware samples (defined as *support set* S) to predict families of the unlabelled malware samples (defined as *query set* Q). When support set S has C families in total and each family possesses K samples, this few-shot classification task is called a C-way K-shot task. Usually, K is too small (e.g. $K = 1, 5, 10\ldots$) to train a classifier with thousands of parameters, known as "few-shot". In principle there is no limitation on the number of queried samples, but for convenience we define that each family has N unlabelled malware samples. To conclude, the support set S totally has $K\times C$ samples $S = \{(x_i^S, y_i^S)\}_{i=0}^{K\times C}$ and the query set Q totally has $N\times C$ samples $Q = \{(x_i^Q, y_i^Q)\}_{i=0}^{N\times C}$. Note that only in the training stage query set has family labels but not testing stage. Whenever given a new arbitrary classification task that classifies novel malware families, a new classifier has to be built using a new support set to classify a new query set, which is described as "dynamic recognition problem" in Section 1.

4.2. Meta-learning framework

To overcome the scarce sample problem during testing, our model must learn some transferrable high-level knowledge from training set, which is enabled by meta-learning (learning to learn). Here a frequently-used "episode training" scheme (Snell et al., 2017; Sung et al., 2018; Vinyals et al., 2016) is adopted as the implementation of meta-learning to mimic testing behavior. Conceptually, episode training performs few-shot classification where only a few labelled samples are shown during training, making the training procedure matches the testing procedure where only a few labelled samples are provided.

To be specific, C families (label space) are first randomly selected from training set D^{train} and then $K\times C$ and $N\times C$ samples are sampled from selected families served as support set S and query set Q. S and Q collectively form an episode and they are fed to model as a single batch. Labelled support set S is used for model to adapt the target task (actually generating prototypes in our SIMPLE). Then the adapted model tries to predict the labels of query set Q (actually comparing with generated prototypes in our SIMPLE) condition on support set S and loss value can be calculated from predictions (namely "meta loss"). At last, parameters of model are updated according to the meta loss from Q by optimizer. A more straightforward illustration is shown in Fig. 1. It also indicates that our trained model is not a plug-and-play classifier but a general classifier generator that is "family-agnostic", where a classifier can be immediately built by adapting the support set S.

During testing, C selected families, support set S and query set Q are successively sampled and the same work flow as training stage will be executed, where model uses information in support set S to predict labels of query set Q. Because C selected families can be diverse in each episode, model is forced to learn transferrable knowledge during training, which

is beneficial for recognizing novel malware families. The detailed algorithm is expressed in Algorithm 1 .

Algorithm 1: Episode training strategy

Input: Training set D^{train}, loss function $\mathcal{L}$, maximum training epoch T, task parameter C, K and N
Output: Trained model M

Initialize the model M with parameter ϕ;
for $i \leftarrow 1$ **to** T **do**
  Randomly sample C families from D^{train} to form task label space L;
  Randomly sample K and N samples from each family in L to formthe support set $S = \{(x_i^S, y_i^S)\}_{i=0}^{K\times C}$ and the query set$Q = \{(x_i^Q, y_i^Q)\}_{i=0}^{N\times C}$ respectively ;
  Feed the support set S to the model M to adapt the new task and outputtailored task classifier: $M' \leftarrow M(\phi, S)$;
  Use the task classifier M' to classify query set Q and produce theclassification loss (meta loss): $\mathcal{L}_{meta} \leftarrow \sum_{i=1}^{N\times C} \mathcal{L}(M'(x_i^Q|\phi, S), y_i^Q)$;
  Derive gradients of ϕ from meta loss $\mathcal{L}_{meta}$ and optimizeϕ using SGD optimizer;
end
return M

4.3. *Sequence representation*

On account of variable lengths of different sequences, we use a sequence projecting module denoted as h to embed each sequence to a fixed-length vector to make it convenient to compute distances between sequences. These vectors are expected to represent the family-wise information and contextual features in the malware sequences. The first submodule of projecting module is an embedding layer consists of a vocabulary matrix and a dropout layer. This submodule projects the integer of API N-gram index to a numeric vector, which is trained by GloVe introduced in Section 3.2. Followed dropout layer randomly cuts off some connections to alleviate the overfitting on the testing set.

We suppose the embedded sequence with length T to be $\hat{x} = (w_1, w_2, \ldots, w_T)$ in which w_i stands for i-th word vector in the sequence. Then the vector sequences will be input to the second submodule, a bidirectional LSTM (BiLSTM). LSTM is a widely used gated RNN model to address gradient problem, which has been proved to be effective in other malware classification work using API sequence (Kang et al., 2019). It can capture sequential dependency by storing long term memory in hidden states and short term memory in cell states to provide more semantic representation. It will output a hidden state vector at each sequential step on receiving the next word vector as following:

$$\overrightarrow{h_t} = \overrightarrow{LSTM}(h_{t-1}, w_t). \tag{2}$$

The inverse traversal is similar:

$$\overleftarrow{h_t} = \overleftarrow{LSTM}(h_{t+1}, w_t). \tag{3}$$

The BiLSTM can have multiple layers if needed. We concatenate the outputs from two directions to form the result of BiLSTM that is still a vector sequence $(h_1, h_2, \ldots, h_t, \ldots, h_T)$:

$$h_t = concatenate(\overrightarrow{h_t}, \overleftarrow{h_t}). \tag{4}$$

The last submodule is a 1D-CNN structure followed by a temporal max pooling layer. The 1D-CNN can extract the local temporal features by convoluting over the sequential dimension of the BiLSTM outputs and 1D temporal max pooling collapse the sequence dimension to keep only the maximum value among sequential items of each feature dimension:

$$h'_t = Conv_{1D}(h_t), \tag{5}$$

$$\tilde{x}(i) = max(h'_{[1:T]}(i)) = max(h'_1(i), h'_2(i), \ldots, h'_T(i)). \tag{6}$$

This submodule is used to reduce the vector sequences to a fixed-length feature vector while maintaining the precious shift-invariance of malware sequence, which is key for detecting subpatterns of API invocation and also adopted in Athiwaratkun and Stokes (2017); Pascanu et al. (2015). For support set S and query set Q, they will be equally embedded by the sequence projecting module h:

$$h_\phi(S) = \tilde{S} = \{\tilde{x}_i^S, y_i^S\}_{i=1}^{K\times C} \tag{7}$$

$$h_\phi(Q) = \tilde{Q} = \{\tilde{x}_i^Q, y_i^Q\}_{i=1}^{N\times C} \tag{8}$$

Although BERT (Devlin et al., 2019), a powerful sequential modeling structure, has become a popular choice and has been explored in malware representation (Huang et al., 2019), it requires for pre-training on large-scale datasets which is not applicable for time-consuming dynamic analysis. The overall embedding architecture is shown in Fig. 2.

4.4. *Unimodal modeling and nearest neighbor*

After projecting sequence to a fixed-length vector, how to classify the queried samples according to the projected samples in the support set S is the most fundamental question. To recognize the novel families dynamically, it is a better choice using local prototype-based method like KNN than using a parametric classifier.

Among the nonparametric methods, unimodal modeling (UM) and nearest neighbor (NN) are on two extremes. UM assumes there exists a sole representative prototype for each family and using family prototypes to classify queried samples rather than raw data points can be more efficient and robust. The most typical unimodal model can be the Prototypical Networks (ProtoNet) (Snell et al., 2017) which simply forms family prototypes as the mean of projected support set within each family. It is equal to the Gaussian Mixture Model (GMM) in the embedding space with identity covariance matrix (Allen et al., 2019). The prototype of cth family can be formulated as:

$$\mu_c = \frac{1}{K}\sum_{i=1}^{K} \tilde{x}_{c,i}. \tag{9}$$

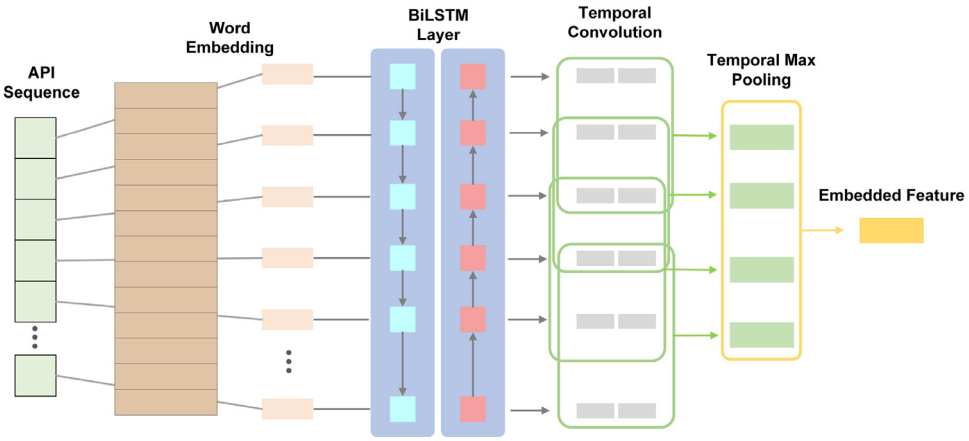


Fig. 2 – The overall architecture of malware sequence embedding module h.

After getting prototypes, queried samples are compared to these family prototypes and their similarities are measured by some distance functions like Square Euclidean distance. These similarities are then input to SoftMax function to generate the final categorical distribution on predicted malware family. During training, all the samples in the same class will be pulled to the class center even for those have completely different features. It can perform very well on some unimodal datasets due to simplicity and concision but fail to work well in some multimodal datasets, such as malware classification datasets.

On the contrary, NN treats all the samples in the support set as prototypes and locally assign the label of closest data point to the test sample. It can model more complex data distribution than being unimodal, but is more sensitive to noise in the data and memory consumption. However, it can often surprisingly achieve remarkable performance and serves as a high-capacity model for few-shot learning (Allen et al., 2019). Formally, the classification strategy of NN can be depicted as following:

$$\hat{y}_i = \arg\min_c d(x_i, x_{c,j}) = \arg\max_c -d(x_i, x_{c,j}), \quad j = 1, 2, \ldots, K \tag{10}$$

where d is an arbitrary distance function. Note that both UM and NN have a determined model capacity which can not be inferred from data dynamically: For a C-way K-shot task, UM has capacity of only C but NN can have capacity of even CK.

4.5. *Multi-prototype modeling*

4.5.1. *Background of multimodal distribution in malware family*

Despite the intuition of solely clustering of malware within the family, malware samples always tend to distribute around multiple separated clusters, shown in Fig. 3. These API N-gram sequences of malware samples within the same cluster have a large percentage of common subsequences and they will be projected to nearby regions in the embedding space whereas different clusters are projected to regions far apart. We intuitively attribute this phenomenon to code reuse with which malware authors write variants of an existed family. Due to usage of existing malicious code, new variants tend to have similar even identical behavior traces with existed ones and thousands of variants can be derived every time when a newly-written malware emerges, which can potentially create a brand-new malware cluster and makes distribution multimodal. This kind of characteristics is also mentioned in Huang et al. (2019) that reports malware samples within the same family always share multiple common behaviors and Tang et al. (2020) also presents the existence of multimodal distribution in malware family in their experiments.

4.5.2. *SIMPLE*

There are mainly two phases for our proposed SIMPLE when applied to few-shot malware classification: The adapting phase and the inference phase, corresponding to support set adaption and query set prediction in Section 4.2. In adapting phase, SIMPLE generates multi-prototypes from embedded support set $\tilde{S}$ described in Section 4.3 as classification basis; In inference phase, embedded query set $\tilde{Q}$ is compared with generated prototypes and queried samples are assigned with the label of nearest prototype. We will describe these two phases in the following paragraphs.

In adapting phase, SIMPLE will do clustering on embedded support set $\tilde{S}$ only and resulted cluster centers are actually prototypes. First, the mean of each family will be added to prototype set S_p as initial prototypes, the same as the prototypes of ProtoNet in Eq. (9). Then, more potential prototypes will be added in a very straightforward way: Each data point of $\tilde{S}$ checks the label of nearest prototype among S_p and establishes a new prototype as itself if mismatched. This kind of strategy is similar to "try-catch" error handling coding style: try classify, catch misclassification and add new prototype to handle. Different from vanilla IMP model (Allen et al., 2019) which only measures the minimal distance, our SIMPLE meets the actual classification requirement and considers the supervised information brought by labels in support set. When a malware family is scattered in the embedding space, it can possibly add several new prototypes based on results of "try classify" and therefore multi-prototypes of this mal-

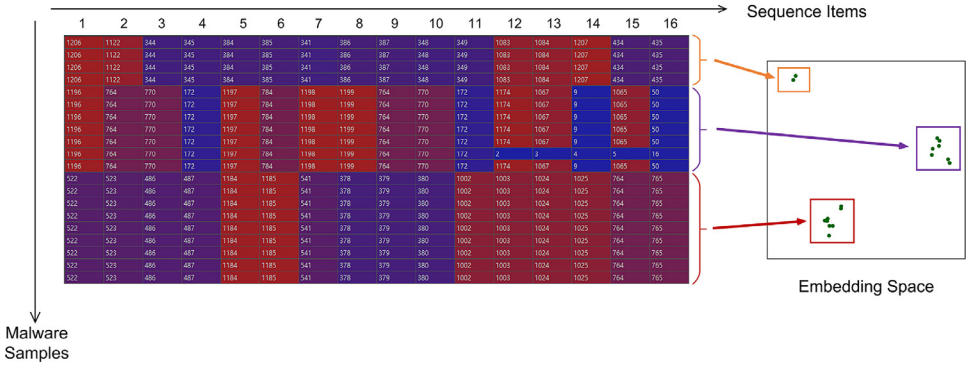


Fig. 3 – An illustration of multimodal distribution of malware family. All the malware samples are from "klez" family of APIMDS dataset but distribute around three separated clusters. Only first 16 elements in the sequences are displayed.

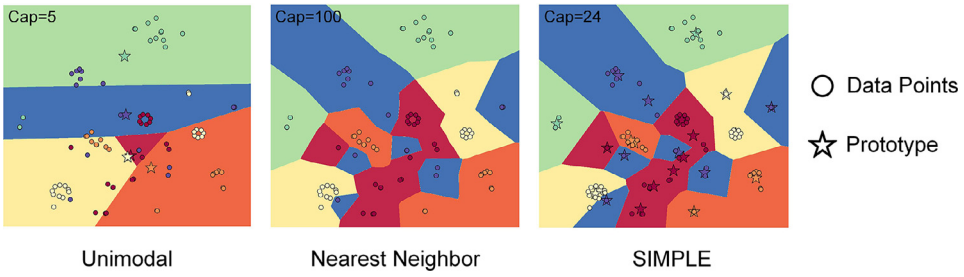


Fig. 4 – Comparison between UM, NN and SIMPLE. Samples are drawn from the D^{test} of APIMDS dataset. UM can not handle multimodal task properly in this illustration and SIMPLE accurately models the distribution using far less prototypes than NN. The decision boundaries in SIMPLE are more straight than NN's whereas the smoother boundaries in NN often incur poor generalization performance. "Cap" in the figure refers to capacity of classifier, namely number of prototypes.

ware family coexist in prototype set S_p for subsequent inference, namely "multi-prototype modeling". However, for malware family distributes around a sole cluster compactly, there is no need to generate more prototypes and possibly only one prototype exists in S_p. In this way, number of prototypes is intelligently inferred from given data to adaptively adjust the model capacity and prevent from both inadequate modeling of UM and overfitting of NN illustrated in Fig. 4. It is obvious that SIMPLE has a more proper number of prototypes compared to UM and NN to handle complex data distribution.

After checking and adding prototypes, reassignment is needed to adjust the prototypes (cluster centers) by assigned data points, like traditional clustering method K-Means. However, the hard assignment scheme is not applicable in our end-to-end differential training architecture. Instead, we adopt the soft assignment scheme in Allen et al. (2019) which calculates the assignment logits of each point to prototypes using Gaussian distribution. Every generated prototype is set as the mean of a Gaussian component and its covariance is set as a diagonal matrix that is the product of an identity matrix and a trainable parameter σ. Then, each data point is given to each Gaussian component to get an assignment logit, formulated as following:

$$\begin{aligned} logit(\tilde{x}_i^S, \mu_j) &= \log \mathcal{N}(\tilde{x}_i^S | \mu_j, \sigma) \\ &= \log\left[\left(\frac{1}{(\sqrt{2\pi}\sigma)^{D/2}}\right)\exp\left(-\frac{(\tilde{x}_i^S - \mu_j)^2}{2\sigma}\right)\right] + \lambda_{i,j}, \end{aligned} \tag{11}$$

where $\lambda_{i,j}$ is a very small negative constant λ' to penalize unmatched point-prototype pairs and avoid prototype shift:

$$\lambda_{i,j} = \begin{cases} \lambda', & l_j \neq y_i^S \\ 0, & l_j = y_i^S \end{cases} \tag{12}$$

Resulted logits will be normalized to an assignment distribution within each point by SoftMax function. Finally, prototypes in S_p will be adjusted to the weighted mean of their members' assignment probability. So far, a complete iteration of clustering has been done and this clustering process can have multiple iterations, which is determined by a hyperparameter T_c. The entire process of generating multiple prototypes is depicted in Algorithm 2 .

In the inference phase, a task classifier M' can be built using prototype set S_p resulted from adapting phase. The embedded query set $\tilde{Q}$ are compared with prototypes in S_p and their similarities are measured by a distance function d and input to SoftMax function to generate a categorical distribution. Because we add the initial prototype as family-wise mean to S_p, there must be at least one prototype in S_p for each family. To encourage multimodal distribution of families and avoid over-penalization on multi-prototype modeling, we adopt the multimodal loss proposed in Allen et al. (2019). For each queried point $\{x_i^Q, y_i^Q\} \in \tilde{Q}$, we first find out the closest prototype for each family to the queried point from $S_p = \{\mu_c, l_c\}$:

$$c_n^* = \arg\min_c d(x_i^Q, \mu_c) \quad s.t. \quad l_c = n, \quad n = 1, 2, \ldots, C \tag{13}$$

Algorithm 2: Generation of Multiple Prototypes of SIMPLE in adapting stage

Input: Projected support set $\tilde{S} = \{\tilde{x}_i^S, y_i^S\}_{i=1}^{K\times C}$,maximum clustering iteration T_c, trainable variance σ, task parameter K, C

Output: Prototype set S_p

Compute mean of each family as initial prototypes: $\mu_c \leftarrow \frac{\sum_{j,y_j=c} x_j^S}{K}, c = 1, 2, ..., C$;
Initialize the prototype set: $S_p = \emptyset, m = 0$;
for $c \leftarrow 1$ **to** C **do**
  Add initial family prototype to prototype set: $S_p \leftarrow S_p \cup \{\mu_c, l_c = c\}, m \leftarrow m + 1$;
end
for $t \leftarrow 1$ **to** T_c **do**
  foreach $\tilde{x}_i^S, y_i^S \in \tilde{S}$ **do**
    foreach $\mu_j \in S_p$ **do**
      Compute distance between prototype and point: $d_j = \left\|\mu_j - \tilde{x}_i^S\right\|_2$
    end
    Get the nearest prototype: $c^* = \arg\min_j d_j$;
    if $l_{c^*} \neq y_i$ **then**
      Establish a new prototype from the misclassified point: $m \leftarrow m + 1, S_p \leftarrow S_p \cup \{\mu_m = \tilde{x_i^S}, l_m = y_i^S\}$;
    end
  end
  foreach $\tilde{x}_i^S, y_i \in \tilde{S}$ **do**
    Compute normalized assignment logits: $z_{i,j} \leftarrow \frac{\exp(\log \mathcal{N}(\tilde{x}_i^S|\mu_j,\sigma)+\lambda_{i,j})}{\sum_{j'}\exp(\log \mathcal{N}(\tilde{x}_i^S|\mu_{j'},\sigma)+\lambda_{i,j'})}, j = 1, 2, ..., m$
  end
  foreach $\mu_j \in S_p$ **do**
    Adjust prototype according to assigned members: $\mu_j \leftarrow \frac{\sum_i z_{i,j}\cdot\tilde{x}_i}{\sum_i z_{i,j}}$
  end
end
return S_p

Then, we apply SoftMax only on these closest family-wise prototypes to generate the categorical distribution of predicted family:

$$P(y_i^Q = n \mid x_i^Q) = \frac{\exp(-d(\tilde{x}_i^Q, \mu_{c_n^*}))}{\sum_{n'}\exp(-d(\tilde{x}_i^Q, \mu_{c_{n'}^*}))} \tag{14}$$

In training stage, this distribution is input to cross-entropy loss function to generate meta-loss $\mathcal{L}_{meta}$ for model training; In testing stage, the predicted family label y_i' of a queried sample x_i^Q can be inferred from this distribution by checking the maximum term:

$$y_i' = \arg\max_n P(y_i^Q = n \mid x_i^Q) \tag{15}$$

Combining Eqs. (13)–(15), we can formulate the adapted classifier M' using prototype set $S_p = \{\mu_c, l_c\}_{c=1}^m$ to classify an unlabelled test sample x:

$$M'(x|S,\phi) = \arg\max_n \frac{\exp(-d(h_\phi(x), \mu_{c_n^*}))}{\sum_{n'}\exp(-d(h_\phi(x), \mu_{c_{n'}^*}))}, \quad n \in \{1, 2, \ldots, C\} \tag{16}$$

where c_n^* should be recomputed using $h_\phi(x)$ and S_p through Eq. (13).

5. Experimental evaluation

As described in Section 3, we evaluate proposed SIMPLE model on two datasets using dynamic analysis for few-shot malware classification: VirusShare_00177 and APIMDS. We mainly focus on the standard few-shot malware classification scenario in this section that emphasizes a model's ability to dynamically recognize novel families and adapt new tasks, and only unseen malware families are tested in the testing stage. Detailed dataset splitting strategies are depicted in Appendix A. Apart from the results and analysis, baseline models and experiment setup will also be involved in this section.

5.1. *Baselines and experiment setup*

We conduct comprehensive comparison experiments between our SIMPLE and existing classification models to demonstrate the performance of SIMPLE. First, some existing malware classification algorithms based on API sequences previously proposed are tested, combined with nonparametric kNN with $k = 1$, namely Nearest-Neighbor classifier:

- API Frequency Histogram. We count the invocation frequency of each API in each API sequence and use this frequency vector to represent each malware sample. Ndibanje et al. (2019)
- API Sequence Alignment. We follow the work of (Cho et al., 2014) which uses a local alignment algorithm called Smith-Waterman algorithm to calculate the sequence-wise similarity after refining the original sequences.
- API Sequence Markov Chain. We implement the algorithm in Amer and Zelinka (2020). At the beginning, the API are clustered to replace API invocation with cluster index to build the cluster transition matrix. Then malware family transition matrices are built within each class accordingly. Finally the family score of a tested sequence is obtained by traveling through the sequence via the family transition matrices.

Since there has been few work concentrating on few-shot malware classification so far (Tang et al., 2020; Tran et al., 2020), some existing few-shot models proposed for CV and NLP tasks serve as strong baselines against SIMPLE. All the listed few-shot baselines are trained under the same meta-learning framework proposed in Section 4.2 and share the same sequence embedding architecture proposed in Section 4.3, only their downstream classifiers make difference:

- MANNWARE (Tran et al., 2020) First proposed few-shot malware classification model by Tran et al. which makes use

of memory-augmented neural network (Santoro et al., 2016) as the classifier.

InductionNet (Geng et al., 2020) It uses dynamic routing algorithm to induce the class prototypes and measures the similarity between a queried sample and a prototype by a neural tensor layer.

HybridAttentionNet (Gao et al., 2019) This model respectively uses an instance attention module to capture the relationship among class samples and a feature attention module to weight the dimensions of feature vectors.

ATAML (Abbasi et al., 2019) It is a variant of fast adaption model MAML (Finn et al., 2017) that separates the task-agnostic feature extraction parameters and task-specific element attention parameters.

PerLayer ATAML (Antoniou et al., 2019) An improved version of ATAML that adds layer-wise learning rate in each adaption loop to precisely control the step length.

FEAT (Ye et al., 2020) Using a set-to-set function to capture the task-specific information in samples in an unordered manner to adapt the target task.

IMP (Allen et al., 2019) It models the multimodal distribution in dataset by DP-means clustering and multiple prototypes coexist to represent one complex class distribution.

All the traditional malware classification models use 64-dimension Word2Vec as API word embedding if necessary and test for at least 1000 iterations. The number of clusters are determined via grid search. For few-shot models, we use SGD optimizer with initial learning rate 0.001, momentum 0.9 and weight decay 0.0005, and we train all the models for 30,000 episodes. Our experiments include $K = 5, 10$ and $C = 5, 10$ resulting in 4 settings in total and we make $N = 5$ in all settings. Early stopping technique is used by running meta-validating on D^{val} for 100 epochs every 100 training episodes to keep track of the best-performed checkpoint and prevent from overfitting. We employ the training trick from (Snell et al., 2017) that uses a higher way N (e.g. $N = 20$) in the training-time than which will be used in the testing-time (e.g. $N = 5, 10$) to boost the performance. The final results are chosen from the better one of larger way training and original way training. The hyper-parameters in typical experiments are listed in Table 2.

Table 2 – Hyper-parameters in typical experiments.

Type	Name	Value
Common Parameters	TF-IDF Items l	2000
	Maximum Sequence Length t	200
	N-gram Window Size	3
Few-Shot Model Parameters	GloVe Dimension	300
	GloVe Learning Rate	0.05
	GloVe Training Epoch	20
	BiLSTM Hidden Dimension	128
	BiLSTM Layer	1
	Dropout Ratio	0.5
	Temporal Convolution Kernel Size	3
	Temporal Convolution Padding Size	1
SIMPLE Parameters	Initial Variance σ	1
	Maximum Clustering Iteration T_c	1

5.2. Evaluation metric

Following the convention of malware classification, we use four kinds of metrics to evaluate the performance of model: accuracy, precision, recall and F1-score. Note that we compute each metric in a multi-class way to fit our multi-class classification experiments and we average each metric in "macro" convention as following:

$$Accuracy = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i} \tag{17}$$

$$Precision = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FP_i} \tag{18}$$

$$Recall = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FN_i} \tag{19}$$

$$F_1 - Score = \frac{1}{C}\sum_{i=1}^{C}\frac{2 * Precision(i) * Recall(i)}{Precision(i) + Recall(i)} \tag{20}$$

In addition, the cross-entropy loss value and running time are taken into consideration in some experiments to reveal some detailed difference between models, such as degree of confidence for results and running efficiency.

5.3. Experimental results

5.3.1. Comparison with baselines

To demonstrate that SIMPLE achieves state-of-the-art malware few-shot classification performance, we compare it with the baselines in Section 5.1. For simplicity, we only measure accuracy when running comparison experiments to decide the performance of each model, shown in Table 3. The detailed classification results are presented as confusion matrixes in Fig. 5(a) and (b).

[3] [4]

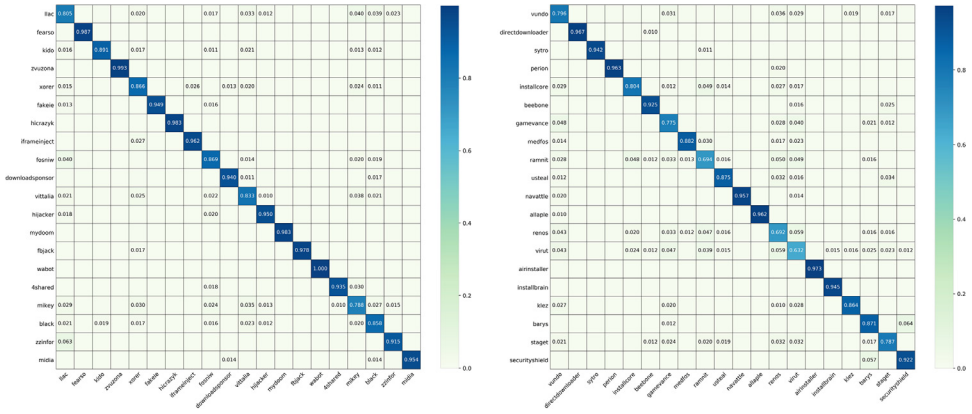
Although some strong baselines like IMP has very competitive accuracy, SIMPLE still outperforms all the baselines on both two datasets and achieves 90% accuracy, which is quite satisfactory when there are only 5 shots in 5 way classification task. Some fast adaption models in baselines, like ATAML and PerLayer ATAML, fall behind proposed SIMPLE by more than 7% on VirusShare_00177 dataset, and the gap is even larger on APIMDS dataset. This shows that metric-based models, such as SIMPLE, IMP and FEAT, are more suitable than these adaption based models for few-shot malware classification tasks. Moreover, fast adaption models need to compute high-order derivations during optimization, where some such complex operations of sequential models are even unsupported on existing platforms. For metric-based models like our SIMPLE, this

[3] Statistics are taken from the no-finetuning version in Tran et al. (2020).

[4] 5-shot was not reported in the original literature and we turn to show the 6-shot accuracy in the column of 5-shot instead.

Table 3 – Testing accuracy of all models on VirusShare_00177 and APIMDS datasets in few-shot malware classification. 95% confidence interval are attached after the mean accuracy and the best accuracy of each column is in bold.

Model	VirusShare_00177				APIMDS			
	5-way		10-way		5-way		10-way	
	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot
Traditional Malware Classification Methods								
API Sequence Histogram (Ndibanje et al., 2019)	86.59 ± 0.31	89.90 ± 0.26	81.51 ± 0.26	85.96 ± 0.23	75.04 ± 0.37	80.78 ± 0.36	68.91 ± 0.29	75.46 ± 0.28
API Sequence Alignment (Cho et al., 2014)	79.43 ± 0.77	81.10 ± 0.77	72.37 ± 0.55	74.11 ± 0.62	59.40 ± 0.86	62.00 ± 0.83	48.15 ± 0.63	50.14 ± 0.63
API Markov Chain (Amer and Zelinka, 2020)	64.12 ± 0.45	66.61 ± 0.44	61.87 ± 0.32	63.82 ± 0.30	55.12 ± 0.41	58.33 ± 0.42	46.32 ± 0.30	49.37 ± 0.30
Few-Shot Models								
MANNWARE (Tran et al., 2020)	–	-	–	–	75.66	78.85	–	–
InductionNet (Geng et al., 2020)	79.43 ± 0.28	80.33 ± 0.28	75.11 ± 0.18	73.23 ± 0.19	73.04 ± 0.34	74.18 ± 0.32	66.09 ± 0.21	67.08 ± 0.21
HybridAttentionNet (Gao et al., 2019)	89.73 ± 0.21	87.88 ± 0.23	81.45 ± 0.18	80.35 ± 0.18	77.53 ± 0.32	79.90 ± 0.33	64.11 ± 0.25	68.37 ± 0.23
ATAML (Abbasi et al., 2019)	85.98 ± 0.24	88.13 ± 0.20	79.89 ± 0.19	83.28 ± 0.17	77.84 ± 0.30	80.74 ± 0.29	63.09 ± 0.23	68.90 ± 0.25
PerLayer ATAML (Antoniou et al., 2019)	86.01 ± 0.19	87.92 ± 0.21	80.21 ± 0.17	84.67 ± 0.14	76.35 ± 0.30	78.63 ± 0.31	63.22 ± 0.22	69.91 ± 0.21
FEAT (Ye et al., 2020)	91.12 ± 0.16	91.56 ± 0.17	87.65 ± 0.13	90.07 ± 0.12	84.89 ± 0.26	86.31 ± 0.24	78.27 ± 0.20	80.44 ± 0.20
IMP (Allen et al., 2019)	91.33 ± 0.17	93.50 ± 0.14	88.21 ± 0.13	90.08 ± 0.13	84.77 ± 0.26	87.68 ± 0.23	78.48 ± 0.19	82.30 ± 0.18
SIMPLE(ours)	**92.35 ± 0.20**	**94.15 ± 0.17**	**89.15 ± 0.12**	**91.20 ± 0.13**	**86.13 ± 0.24**	**89.22 ± 0.21**	**80.04 ± 0.18**	**84.21 ± 0.16**



(a) VirusShare_00177 (b) APIMDS

Fig. 5 – Confusion matrixes of few-shot malware classification results on D^{test} of VirusShare_00177 dataset and APIMDS dataset. Matrix items are normalized and weights less than 0.01 are omitted.

kind of problem never troubles because common optimizers are sufficient for training.

Almost all the models have lower accuracy on APIMDS dataset than VirusShare_00177 dataset, and we explain it as APIMDS is more difficult and closer to production environment than VirusShare_00177. For traditional malware classification methods, the performance difference between SIMPLE and them becomes more obvious on the APIMDS dataset than VirusShare_00177, which exceeds 10% at best. It proves that our proposed SIMPLE is more robust and more effective than these ones when handling API sequences. It should be noted that SIMPLE owns 10% better accuracy than the previous few-shot malware classification model MANNWARE on the same dataset APIMDS, which reflects our contribution.

For IMP and FEAT two recent advanced few-shot classification models, SIMPLE still outperforms them especially IMP, the original version of SIMPLE. Compared to vanilla IMP, SIMPLE fuses supervised information from labels of support set into the decision of prototype establishment, which really makes sense during training. FEAT also takes account of interactions between data points of support set, but it lacks explicit multimodal modeling like proposed SIMPLE that limits its performance.

5.3.2. *Effect of meta learning*

We train SIMPLE under meta learning framework, which is detailed in Section 4.2, to learn some general high-level knowledge across tasks and enhance SIMPLE's ability to recognize novel malware families. To investigate the effect of meta learning, we compare it with two more models trained without meta-learning in the few-shot malware classification experiments.

Fine-tuning uses a linear classifier to classify the queried embedded sequences into predicted families after parameters are updated for several iterations, according to the forward loss of the support set owning a few samples. It is a common parametric baseline in few-shot learning literatures (Gao et al., 2019), compared to nonparametric SIMPLE. SIMPLE$^{\#}$ represents the SIMPLE model without meta-learning, which means it will be applied to D^{test} as soon as being initialized. It inherits the nonparametric nature of proposed SIMPLE, which builds the adapted classifier by comparing queried input with multiple prototypes, but lacks the task-level knowledge learned across tasks in training stage to produce high-quality embeddings. Neither of these two models use meta learning, with comparison to proposed SIMPLE, and all the listed models use the same sequence embedding architecture detailed in Section 4.3. In experiments, we set the maximum fine-tuning iteration $T_{ft} = 20$ to make a tradeoff between efficiency and performance.

Table 4 shows the results of these three models. The testing time is measured on our machine of 20 cores Intel Core i9-10900X 3.7 GHz CPU and RTX 2080Ti GPU. Note that the testing time refers to the total time to complete a few-shot malware classification task in testing stage, including the time to adapt the given labelled few shots (support set) and make inference on unlabelled samples (query set). For fine-tuning and SIMPLE$^{\#}$, they do not employ meta-learning to learn some transferrable knowledge, thus they do not have a explicit training process; By the way for proposed SIMPLE, it takes 163.2 ms on average to run an episode during training.

From the table, we can see fine-tuning works quite well on VirusShare_00177 dataset, but breaks down on the APIMDS dataset. This is because sequences within each family are highly similar on VirusShare_00177 dataset and in most of the time direct sequence comparison is enough to ensure the performance. But on APIMDS dataset, it is of greater importance to enhance contextual understanding, and fine-tuning fails to obtain sufficient useful information from support set, which leads to terrible performance. Untrained SIMPLE has worse performance than proposed SIMPLE as expected, but still remains acceptable compared to fine-tuning. This proves proposed SIMPLE indeed learns some high-level general knowledge across tasks and meta-learning is feasible in malware classification. Besides, fine-tuning requires for long time adaption during testing, which is more than 10 times longer than SIMPLE. SIMPLE

Table 4 – Experimental results of investigation of meta learning effect in few-shot malware classification. SIMPLE# stands for the SIMPLE model without applying meta learning. Best metric value among all the models under the same settings is in bold.

Metric	Model	VirusShare_00177				APIMDS			
		5-way		10-way		5-way		10-way	
		5-shot	10-shot	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot
Accuracy	Fine-tuning	83.35	85.01	79.28	80.97	43.32	42.63	36.93	35.62
	SIMPLE#	87.30	90.44	83.03	86.16	74.99	77.66	67.73	68.23
	SIMPLE	**92.35**	**94.15**	**89.15**	**91.20**	**86.13**	**89.22**	**80.04**	**84.21**
Precision	Fine-tuning	84.30	85.52	79.92	81.29	33.22	31.8	30.54	28.58
	SIMPLE#	89.19	91.90	85.35	88.28	77.13	80.09	70.03	70.34
	SIMPLE	**93.80**	**95.08**	**90.81**	**92.25**	**87.47**	**90.37**	**81.55**	**85.76**
Recall	Fine-tuning	83.35	85.01	79.28	80.97	43.32	42.63	36.93	35.62
	SIMPLE#	87.3	90.44	83.03	86.16	74.99	77.66	67.73	68.23
	SIMPLE	**92.35**	**94.15**	**89.15**	**91.21**	**86.12**	**89.22**	**80.04**	**84.21**
F1-score	Fine-tuning	81.49	83.21	76.92	78.65	33.68	32.69	29.65	28.07
	SIMPLE#	86.56	90.10	82.18	85.59	73.71	76.78	66.34	66.88
	SIMPLE	**92.02**	**93.79**	**88.68**	**90.86**	**85.84**	**89.07**	**79.56**	**83.98**
Testing Loss	Fine-tuning	0.972	0.790	0.850	0.801	4.047	4.136	2.549	2.606
	SIMPLE#	1.579	1.576	2.267	2.266	1.6	1.601	2.293	2.293
	SIMPLE	**0.299**	**0.257**	**0.521**	**0.420**	**0.449**	**0.386**	**0.712**	**0.618**
Testing Time (ms)	Fine-tuning	923.22	1026.57	1021.88	1156.70	899.76	1150.63	1164.08	1438.59
	SIMPLE#	40.90	60.30	88.33	129.77	**41.39**	66.67	81.87	113.55
	SIMPLE	**37.83**	**59.02**	**74.49**	**105.86**	42.40	**58.68**	**75.62**	**106.26**

Table 5 – Experimental results of three prototype-based models owning different model capacity in few-shot malware classification scenario. Bold items indicate the highest metric value among all the models under the same settings.

Metric	Model	VirusShare_00177				APIMDS			
		5-way		10-way		5-way		10-way	
		5-shot	10-shot	5-shot	10-shot	5-shot	10-shot	5-shot	10-shot
Accuracy	UM	89.14	91.01	87.33	89.14	83.10	85.28	76.37	80.31
	NN	91.38	93.67	88.70	90.92	84.19	87.82	78.59	83.14
	SIMPLE	**92.35**	**94.15**	**89.15**	**91.20**	**86.13**	**89.22**	**80.04**	**84.21**
Precision	UM	91.27	92.90	89.80	91.31	85.36	87.16	79.11	82.19
	NN	92.87	94.72	90.22	92.33	85.82	89.18	80.20	83.75
	SIMPLE	**93.80**	**95.08**	**90.81**	**92.52**	**87.47**	**90.37**	**81.55**	**85.76**
Recall	UM	89.14	91.01	87.33	89.14	83.10	85.28	76.37	80.31
	NN	91.38	93.67	88.70	90.92	84.19	87.82	78.59	83.14
	SIMPLE	**92.35**	**94.15**	**89.15**	**91.21**	**86.12**	**89.22**	**80.04**	**84.21**
F1-score	UM	88.61	90.61	86.86	88.75	82.45	84.81	75.68	79.95
	NN	90.95	93.43	88.13	90.54	83.31	87.24	77.56	82.26
	SIMPLE	**92.02**	**93.79**	**88.68**	**90.86**	**85.84**	**89.07**	**79.56**	**83.98**

adopts nonparametric classifier to complete the classification in one forward time, free from the time-consuming fine-tuning.

5.3.3. *Effect of multi-prototype modeling*

To further prove that SIMPLE can indeed automatically adjust the model capacity to best fit the data distribution, we compare it to two baselines of extreme capacity: NN and UM(ProtoNet (Snell et al., 2017)), which has been detailed in Section 4.4. The results are shown in Table 5.

Because SIMPLE outperforms these two baselines again, it can be inferred that too few prototypes (UM) can not identify the multimodal distribution and too much prototypes (NN) will sometimes mislead the queried samples, which are both roots of relatively low accuracy. These two models have fixed model capacity because their number of prototypes is fixed for all given data.

By contrast, SIMPLE can intelligently inference the appropriate number of prototypes from samples in a "try-catch" manner. By pre-checking classification result using existing prototypes, unnecessary prototypes will be skipped before added to prototype set and this prevents from generating excessive prototypes. This brings about 1% accuracy improvement on VirusShare_00177 and 3% accuracy improvement on APIMDS over NN. Besides, low capacity of UM can only work well on highly linearly separable data distribu-

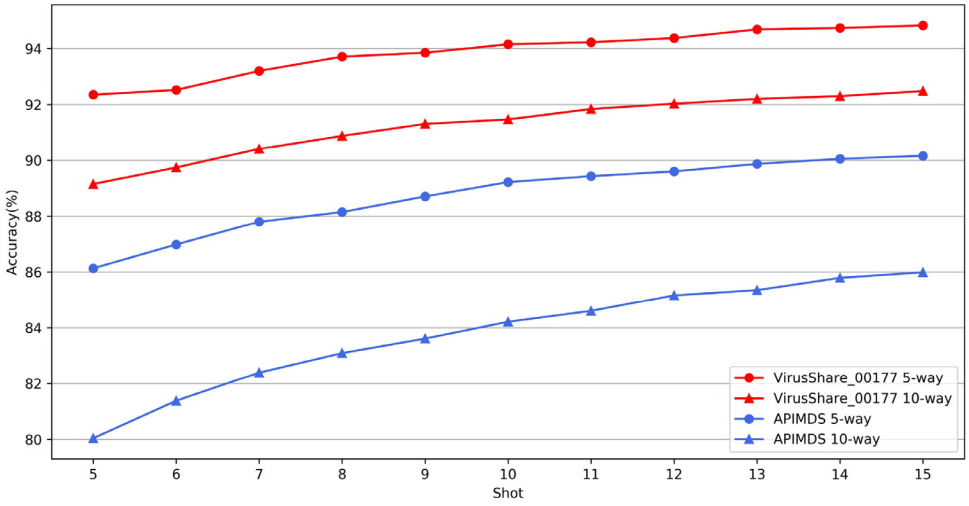


Fig. 6 – Influence on few-shot malware classification accuracy with shot number.

tions, but fail on complex multimodal distribution of malware API sequence data. The accuracy improvement over UM is more obvious: about 3% on VirusShare_00177 and 4% on APIMDS. We can infer that relative low capacity is the core problem of API sequence modeling, compared to high capacity.

Surprisingly, NN achieves good results compared to SIMPLE. However, it relies on all the data points of the support set S with very high capacity and requires for more memory resources than SIMPLE, which only needs to hold S_p. When the support set S is getting larger, the cost of such high capacity is unaffordable, in particular precious GPU memory.

5.3.4. *Influence of K-shots*

The concept of "few-shot" can be differently defined in various scenarios according to the requirements. Thus, to evaluate SIMPLE comprehensively, we gradually increase the shot number k in our experiments and come to investigate the influence of shot number K on few-shot malware classification performance in detail. Also for simplicity, only accuracy is measured and presented in Fig. 6. With more shots within each family of the support set S, the accuracy increases obviously as expected and there is at most 6% increment of accuracy at the cost of only 10 more shots. This kind of phenomenon is reasonable and in line with expectation, because more shots within each family means model can extract more information from the support set to adapt the new task, which largely alleviates the overfitting caused by scarce samples. In addition, it indicates that we can get very excellent performance of malware classification by adding a small number of samples to the original few-shot trained models, free from training a very cumbersome classifier with thousands of samples from scratch. This feature can be very appealing in practice, because testing performance will not be constrained by the training shot number and a few-shot trained model can be applied in both few-shot and many-shot environments.

5.3.5. *Influence of maximum sequence length*

In the preceding experiments, we set a common maximum sequence length t and truncate the sequences with longer lengths. Using truncated sequences to replace the original misaligned sequences as input is a frequently-used way to complete sequence embedding task in the literatures (Tran et al., 2020). After truncation, short sequences are padded with zero value to align with truncated sequences to form a high-dimensional matrix, which is crucial for parallel processing to accelerate training and testing process. Above this, a natural question is how long should we truncate the sequences (how much of t) to obtain best performance and meanwhile reduce the processing delay and resource consumption. Larger t means more sequence items are considered during training, which may promise better performance, but this will bring about longer processing time and more memory usage at the same time. On the other hand, longer aligned sequences will increase the burden on model and possibly decrease the classification accuracy, caused by an excess of meaningless padded zero. Thus we come to investigate the effect of maximum sequence length and measure the performance and processing speed of proposed SIMPLE with different t, presented in Figs. 7 and 8. When sequences are short, there is notable performance gain with the increase of maximum sequence length t. However, once t exceeds a certain level, such as $t = 200$ in Fig. 7, the performance changes become irregular and accuracy sometimes drops drastically with longer sequence length. From this, it can be inferred that accuracy has no explicit positive linear relationship with sequence length, and there exists an optimal maximum sequence length for malware few-shot classification, such $t = 200$ in experiments, which is against intuition. The process delay of testing also increases linearly with t and $t = 1000$ has almost four times longer of delay as optimal $t = 200$ in Fig. 7. On VirusShare_00177 dataset, the delay has already exceeded 200 ms per episode for $t = 1000$ that is hard to be ignored in practice. Note that most samples in APIMDS dataset have lengths less than 500, which indicates using t larger than 500 does not help at all.

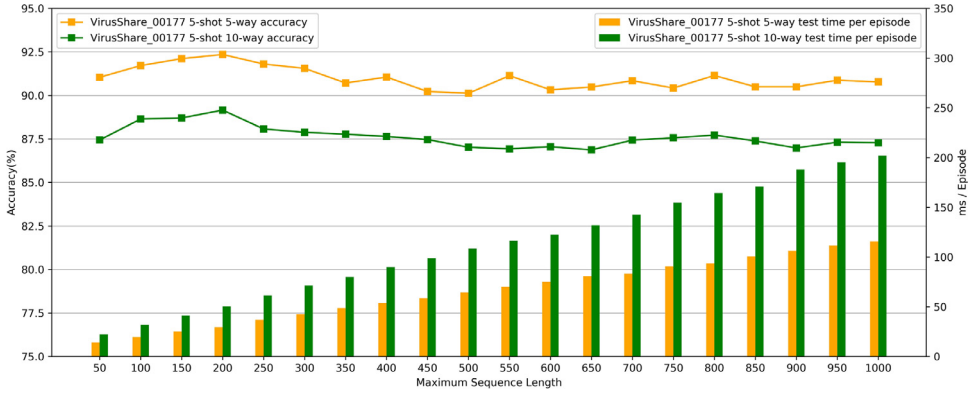


Fig. 7 – Influence of maximum sequence length on few-shot malware classification accuracy and testing time of VirusShare_00177 dataset.

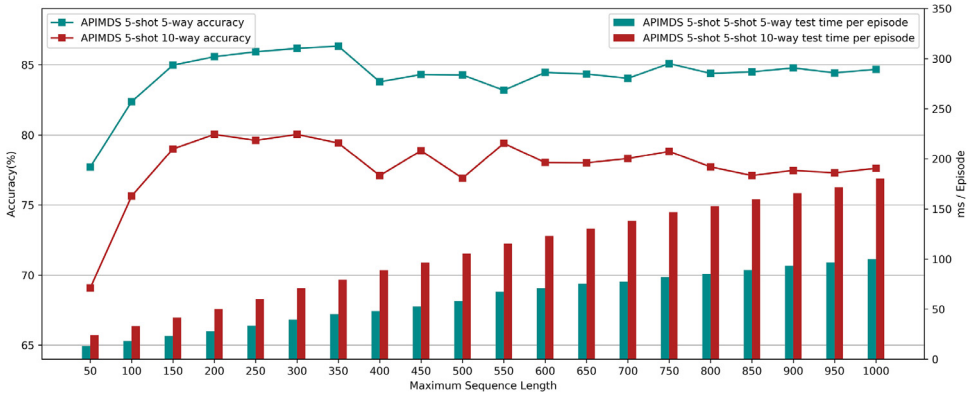


Fig. 8 – Influence of maximum sequence length on few-shot malware classification accuracy and testing time of APIMDS dataset.

6. Discussion and analysis

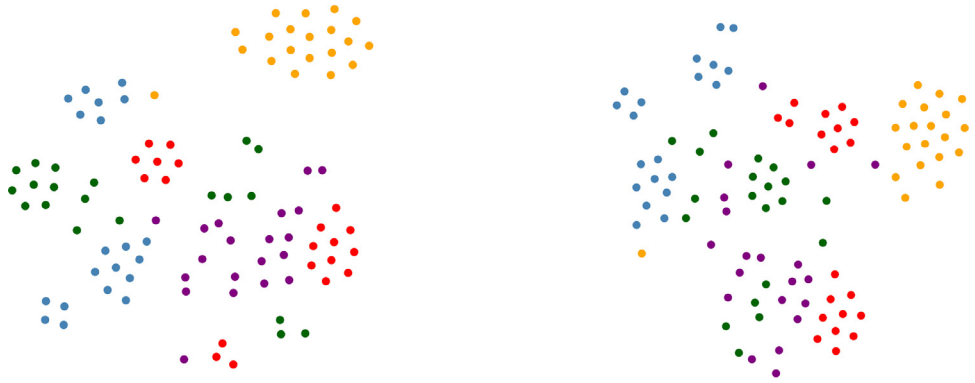
We will discuss more aspects to analyze the reason of effectiveness of SIMPLE in a more detailed way in this section to make our results more convincing and provide more interpretability. We set out from two topics in this section: the optimization superiority with multi-prototype modeling and fast adaption feature of SIMPLE.

6.1. *Optimization superiority with multi-prototype modeling*

Substantially, SIMPLE is a metric-based model, which embeds each malware sample into the embedding space and discriminate samples with some distance functions, thus the embedding quality can have great impact on the final classification performance. Normally, a good embedding has two features: Samples from the same class are projected into a compact cluster; Samples from different classes are separated with large margin. During training, the embedding module is continuously optimized on different tasks to better fit the model's characteristics, and the classification loss, namely optimization objective, can be quantized and decomposed with these two metrics, as the following form:

$$\mathcal{L}_{classification} \triangleq \mathcal{L}_{intra} + \mathcal{L}_{inter}, \tag{21}$$

where $\mathcal{L}_{intra}$ measures the compactness of cluster and $\mathcal{L}_{inter}$ measures the degree of separation of different clusters. For traditional unimodal models, these two metrics jointly decide the classification performance of the model and both should be minimized to improve performance. However, for some multimodal distribution tasks, the $\mathcal{L}_{intra}$ can be significantly hard to optimize, because samples from the same family can distribute around several different clusters and different clusters can be hardly merged through the embedding directly. Without modification to the optimization objective, unimodal models are hard to balance the $\mathcal{L}_{intra}$ and $\mathcal{L}_{inter}$ during optimization. As the result, they converge slowly and their final performance can not be guaranteed on multimodal distribution datasets. Now we look back at SIMPLE. It allows multiple prototypes coexist to describe a family's distribution, so there is no need to make all the samples gather at a sole cluster when it encounters the multimodal distribution. That



(a) SIMPLE (b) ProtoNet

Fig. 9 – t-SNE visualization of embeddings from (a) SIMPLE (b) ProtoNet. Samples are from APIMDS dataset of families: gamevance, ramnit, navattle, klez and staget.

is to say the $\mathcal{L}_{intra}$ term in the original objective can be ignored with multi-prototype modeling, and the model can focus on minimizing the $\mathcal{L}_{inter}$ only, which greatly reduces the difficulty of optimization. A straightforward demonstration is shown in Fig. 9. The embeddings of Fig. 9(a) are from SIMPLE, where samples from different families are well separated and there exists several different clusters for a certain family. However, the embeddings of Fig. 9(b) from ProtoNet(unimodal) are very messy, where samples from different families get mixed, as the result of excessive optimization bias on $\mathcal{L}_{intra}$. Besides, SIMPLE exploits the supervised information in the support set to facilitate the training process, which has been proved effective in some previous work (Lai et al., 2020).

6.2. *Fast adaption feature of SIMPLE*

It has been mentioned that SIMPLE belongs to metric-based models, different from fast adaption models like MAML (Finn et al., 2017). According to some literatures, lacking adaption is the reason some models fall behind the baselines (Chen et al., 2019) and a natural question is whether SIMPLE really degrades without explict adaption phase. In fact, it has been proposed in Snell et al. (2017) that the metric-based model is equivalent to a linear classifier with a particular parameterization. In Eq. (14), Euclidean distance function d is used and we can expand the term $-d(\tilde{x}_i^Q, \mu_{c_n^*})$:

$$-d(\tilde{x}_i^Q, \mu_{c_n^*}) = -(\tilde{x}_i^Q - \mu_{c_n^*})^2 = -(\tilde{x}_i^Q)^T(\tilde{x}_i^Q) + 2(\mu_{c_n^*})^T\tilde{x}_i^Q - (\mu_{c_n^*})^T(\mu_{c_n^*}), \tag{22}$$

where $-(\tilde{x}_i^Q)^T(\tilde{x}_i^Q)$ is a constant with respect to the prototypes and we will drop it as it does not affect the final result. The remaining terms can be organized as a linear classifier:

$$2(\mu_{c_n^*})^T\tilde{x}_i^Q - (\mu_{c_n^*})^T(\mu_{c_n^*}) = w_n^T\tilde{x}_i^Q + b_n, \tag{23}$$

where $w_n = 2\mu_{c_n^*}$ and $b_n = -(\mu_{c_n^*})^T(\mu_{c_n^*})$. This indicates that metric-based models (including SIMPLE) can generate a novel linear classifier when a new support set is input. This is highly similar to fast adaption model, where a new classifier is adapted by gradients or other terms from the support set. Through the nonparametric distance function, SIMPLE is able to quickly adapt to a new task with the prototypes generated from the support set. Note that our SIMPLE can only adapt the parameters in the classifier, but not the parameters in the embedding module. This is a major difference between SIMPLE and other fast adaption models and also may be potential limitation.

7. Future work

Although we achieve very attractive results with SIMPLE, there are still some challenges to be solved. For instance, we observed that some malware samples from different families have identical API invocation sequences and they can not be identified by embedding module using only sequences as input. Additionally, the generalized few-shot malware classification scenario, where testing set contain some base families, has not been tested in experiments, but it is a more practical and challenging scenario in the industry applications. Thus we suggest some future work to facilitate the few-shot malware classification. First is the dataset scale augmentation. There are only 20 samples for each family in datasets and it severely limits the model's generalization ability. We conclude that family scale is a key factor that affects the performance of few-shot malware classification from some extra experiments. For FSL especially with meta-learning, more samples in a family means more prior knowledge brought by the base families. Also, number of families should be increased to make more task combinations possible. Both two targets can be done by including more malware samples, like analyzing more VirusShare datasets. Second, replace the embedding architecture with a more powerful structure if dataset scale is relatively large, such as BERT (Devlin et al., 2019) and its variants. These models are better at contextual information extraction, and they can provide better sequence embeddings. Last, combining static and dynamic analysis (e.g. using multimodal fusion) is a very promising direction to solve

the identical sequence problem (Gibert et al., 2020). By fusing multimodal input, more features are introduced to combat overfitting in the FSL. A simpler way to bring more features is to take parameters in the API invocations into consideration (Agrawal et al., 2018). In short, it is beneficial to increase the amount of information of input data.

8. Conclusion

In this paper, we propose a novel model called SIMPLE to recognize novel malware families with limited samples and solve the few-shot malware classification task with multi-prototype modeling. We carry out dynamic analysis on malware samples to obtain the API invocation sequences as input to our embedding module that consists of word embedding, BiLSTM and temporal convolution. We design a new algorithm to generate multiple prototypes to jointly represent a family distribution, inspired by our observations of multimodal distribution in malware sequences. We train SIMPLE under the meta-learning framework and make it task-oriented to adapt new tasks. Experimental results show that SIMPLE achieve state-of-the-art few-shot malware classification performance and multi-prototype plays a key role in enhancing expression. We also demonstrate SIMPLE can automatically inference the appropriate number of prototypes to represent a family, and the performance can be notably improved with more shots in the support set. Few-shot malware classification shed light on improving the scalability of existing classification models and is worth to be explored.

Funding information

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Datasets

A1. *VirusShare_00177*

This dataset is a large subpackage of Windows PE malware downloaded from VirusShare website with No. 177 (Roberts, 2011), which contains 65,535 samples and is open for download on Oct 4th, 2015. Since this dataset is completely unlabelled, we borrow the labelling routines from (Tang et al., 2020), where we use VirusTotal (Total, 2012) to scan and make reports for each sample in the dataset. VirusTotal will report file hashes like MD5 and scan each sample with many anti-virus (AV) engines to generate scanning results in a JSON file. Due to variety of naming convention of AV engines, resulted family labels from different AV engines are highly inconsistent and we use AVClass (Sebastián et al., 2016) to extract an unique canonical family label from provided AV labels as the ground-truth family label of each sample. Note that we discard all the SINGLETON samples after running AVClass because they do not belong to any of the known families. After labelling, there are 127 resulted families containing more than 20 samples and we randomly take 20 samples from each family to form final dataset. The splitting result for VirusShare_00177 is $D^{train} : D^{val} : D^{test} = 73 : 20 : 20$ families, presented in Table A.6.

A2. *APIMDS*

APIMDS is a public malware API invocation dataset[5] that was first proposed in Ki et al. (2015) and has been a bench-

Table A.6 – Detailed dataset splitting result for VirusShare_00177.

Subset	Owned family name	Count
D^{train}	directdownloader,hiloti,sytro,downloadadmin,softcnapp,buterat, kykymber,zbot,psyme,badur,zapchast,toggle,simbot,qqpass,outbrowse, blacole,soft32downloader,qhost,loring,softonic,webprefix,getnow,startp, lunam,ibryte,egroupdial,darkkomet,pirminay,reconyc,lineage,dealply, darbyen,cpllnk,ipamor,antavmu,kovter,linkular,jeefo,banload,firseria, hidelink,scrinject,vilsel,downloadassistant,installmonetizer,sefnit, instally,autoit,staser,extenbro,urelas,installerex,zeroaccess,airinstaller, jyfi,gator,goredir,icloader,lipler,bundlore,ircbot,nimda,gepys,inor,iframeref, wonka,urausy,bettersurf,adclicer,refresh,domaiq,fsysna,c99shell	73
D^{val}	xtrat,faceliker,somoto,dlhelper,fujacks,includer,1clickdownload,trymedia, patchload,acda,scarsi,windef,shipup,decdec,loadmoney,microfake,msposer, refroso,yoddos,redir	20
D^{test}	llac,fearso,kido,zvuzona,xorer,fakeie,hicrazyk,iframeinject,fosniw, downloadsponsor,vittalia,hijacker,mydoom,fbjack,wabot,4shared,mikey, black,zzinfor,midia	20

[5] http://ocslab.hksecurity.net/apimds-dataset.

Table A.7 – Detailed dataset splitting result for APIMDS.

Subset	Owned family name	Count
Train	pasta,wapomi,waledac,zbot,bublik,buzus,toggle,scar,bladabindi, 1clickdownload,soft32downloader,qhost,loring,simda,zusy,sality, fareit,darkkomet,vobfus,lollipop,cycbot,fesber,archsms,poison, banload,zegost,mediafinder,shipup,vilsel,fakesysdef,ganelp,mydoom, hacyayu,smartfortress,crytex,autoit,shylock,megasearch,4shared, agentb,zeroaccess,onlinegames,loadmoney,tdss,gator,hupigon,hlux, ircbot,farfli,fraudpack,gepys,bifrose,parite,high,delf,urausy,chir, winlock,daws,shiz,domaiq,hotbar,	62
Validate	xtrat,multiplug,somoto,mywebsearch,swisyn,primecasino,hala,rebhip, koutodoor,winwebsec,mabezat,yakes,palevo,turkojan,firseria,pykspa, coolmirage,expiro,magania,dapato	20
Test	vundo,directdownloader,sytro,perion,installcore,beebone,gamevance, medfos,ramnit,usteal,navattle,allaple,renos,virut,airinstaller, installbrain,klez,barys,staget,securityshield	20

mark dataset to conduct malware classification experiments (Tran et al., 2020). It contains 23146 malware API invocation sequences recored by hook library but some family labels of sequences are missing. Thus we relabel this dataset in the same way as VirusShare_00177. Finally, we obtain 102 families with more than 20 samples and we take 20 randomly samples out from each family. The splitting result for VirusShare_00177 is $D^{train} : D^{val} : D^{test} = 62 : 20 : 20$ families, presented in Table A.7.

CRediT authorship contribution statement

Peng Wang: Conceptualization, Validation, Formal analysis, Investigation, Writing - original draft, Writing - review & editing. **Zhijie Tang:** Conceptualization, Validation, Formal analysis, Data curation, Software, Investigation, Writing - original draft, Writing - review & editing. **Junfeng Wang:** Project administration, Supervision, Writing - review & editing.

REFERENCES

Abbasi A, Albrecht C, Vance A, Hansen J. Attentive task-agnostic meta-learning for few-shot text classification, 36; 2019. p. 1293–327.

Agrawal R, Stokes JW, Marinescu M, Selvaraj K. Neural sequential malware detection with parameters. In: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, Vol. 2018-April; 2018. p. 2656–60. doi:10.1109/ICASSP.2018.8461583.

Allen KR, Shelhamer E, Shin H, Tenenbaum JB. Infinite mixture prototypes for few-shot learning. In: 36th International Conference on Machine Learning, ICML 2019, Vol. 2019-June; 2019. p. 348–57. arXiv:1902.04552.

Amer E, Zelinka I. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. Comput. Secur. 2020;92. doi:10.1016/j.cose.2020.101760.

Antoniou A, Storkey A, Edwards H. How to train your MAML. 7th International Conference on Learning Representations, ICLR 2019, 2019.

Athiwaratkun B, Stokes JW. Malware Ccassification with LSTM and GRU language models and a character-level CNN. In: IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 2017; 2017. p. 2482–6.

Bhodia N, Prajapati P, Di Troia F, Stamp M. Transfer learning for image-based malware classification. In: ICISSP 2019 - Proceedings of the 5th International Conference on Information Systems Security and Privacy; 2019. p. 719–26. arXiv:1903.11551. doi:10.5220/0007701407190726.

Chen WY, Wang YCF, Liu YC, Kira Z, Huang JB. A closer look at few-shot classification. 7th International Conference on Learning Representations, ICLR 2019, 2019.

Cho IK, Kim T, Shim YJ, Park H, Choi B, Im EG. Malware similarity analysis using API sequence alignments. J. Internet Serv. Inf. Secur. (JISIS) 2014;4(November):103–14.

Devlin J, Chang MW, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding, 1; 2019. p. 4171–86. arXiv:1810.04805.

Du D, Sun Y, Ma Y, Xiao F. A novel approach to detect malware variants based on classified behaviors. IEEE Access 2019;7:81770–82. doi:10.1109/ACCESS.2019.2924331.

Fei-Fei L, Fergus R, Perona P. One-shot learning of object categories. IEEE Trans. Pattern Anal. Mach. Intell. 2006;28(4):594–611. doi:10.1109/TPAMI.2006.79.

Finn C, Abbeel P, Levine S. Model-agnostic meta-learning for fast adaptation of deep networks, 3; 2017. p. 1856–68. arXiv:1703.03400.

Gao T, Han X, Liu Z, Sun M. Hybrid attention-based prototypical networks for noisy few-shot relation classification. In: 33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019; 2019. p. 6407–14. doi:10.1609/aaai.v33i01.33016407.

Geng R, Li B, Li Y, Zhu X, Jian P, Sun J. Induction networks for few-shot text classification. In: EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference; 2020. p. 3904–13. arXiv:1902.10482. doi:10.18653/v1/d19-1403.

Gibert D, Mateu C, Planes J. HYDRA: a multimodal deep learning framework for malware classification. Comput. Secur. 2020;95. doi:10.1016/j.cose.2020.101873.

Huang, Y. T., Chen, T. Y., Sun, Y. S., Chen, M. C., 2019. Learning malware representation based on execution sequences. arXiv:1912.07250

Kang J, Jang S, Li S, Jeong YS, Sung Y. Long short-term memory-based malware classification method for information security. Comput. Electr. Eng. 2019;77:366–75. doi:10.1016/j.compeleceng.2019.06.014.

Ki Y, Kim E, Kim HK. A novel approach to detect malware based on API call sequence analysis. Int. J. Distrib. Sens. Netw. 2015. doi:10.1155/2015/659101.
Kim H, Kim J, Kim Y, Kim I, Kim KJ, Kim H. Improvement of malware detection and classification using API call sequence alignment and visualization. Cluster Comput. 2019;22(s1):921–9. doi:10.1007/s10586-017-1110-2.
Kulis B, Jordan MI. Revisiting k-means: New algorithms via Bayesian nonparametrics, 1; 2012. p. 513–20. arXiv:1111.0352.
Lai VD, Dernoncourt F, Nguyen TH. Exploiting the matching information in the support set for few shot event classification. In: LNAI, 12085; 2020. p. 233–45. arXiv:2002.05295. doi:10.1007/978-3-030-47436-2_18.
Li H, Eigen D, Dodge S, Zeiler M, Wang X. Finding task-relevant features for few-shot learning by category traversal. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2019-June; 2019. p. 1–10. arXiv:1905.11116. doi:10.1109/CVPR.2019.00009.
Lim MJ, An JJ, Jun SH, Kwon YM. Efficient algorithm for malware classification: N-gram MCSC. Int. J. Comput. Digit. Syst. 2020;9(2):179–85. doi:10.12785/IJCDS/090204.
Lyda R, Hamrock J. Using entropy analysis to find encrypted and packed malware. IEEE Secur. Privacy 2007;5(2):40–5. doi:10.1109/MSP.2007.48.
Ma X, Guo S, Li H, Pan Z, Qiu J, Ding Y, Chen F. How to make attention mechanisms more practical in malware classification. IEEE Access 2019;7:155270–80. doi:10.1109/ACCESS.2019.2948358.
Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. Malware images: visualization and automatic classification. ACM International Conference Proceeding Series, 2011.
Ndibanje B, Kim KH, Kang YJ, Kim HH, Kim TY, Lee HJ. Cross-method-based analysis and classification of malicious behavior by API calls extraction. Appl. Sci. (Switzerland) 2019;9(2). doi:10.3390/app9020239.
Nichol, A., Achiam, J., Schulman, J., 2018. On first-order meta-learning algorithms. arXiv:1803.02999
Oreshkin BN, Rodriguez P, Lacoste A. Tadam: task dependent adaptive metric for improved few-shot learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems; 2018. p. 719–29.
Pagliardini M, Gupta P, Jaggi M. Unsupervised learning of sentence embeddings using compositional n-gram features, 1; 2018. p. 528–40. arXiv:1703.02507. doi:10.18653/v1/n18-1049.
Park Y, Reeves D, Mulukutla V, Sundaravel B. Fast malware classification by automated behavioral graph matching. In: ACM International Conference Proceeding Series; 2010. p. 1–4. doi:10.1145/1852666.1852716.
Pascanu R, Stokes JW, Sanossian H, Marinescu M, Thomas A. Malware classification with recurrent networks. In: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, Vol. 2015-Augus; 2015. p. 1916–20. doi:10.1109/ICASSP.2015.7178304.
Pennington J, Socher R, Manning CD. GloVe: global vectors for word representation, 31; 2014. p. 1532–43. doi:10.3115/v1/d14-1162.
Ravi C, Manoharan R. Malware detection using Windows API sequence and machine learning. Int. J. Comput. Appl. 2012;43(17):12–16. doi:10.5120/6194-8715.
Ravi S, Larochelle H. Optimization as a model for few-shot learning. In: Proceedings of the 5th International Conference on Learning Representations (ICLR 2017); 2017. p. 1–11.
Roberts, J.-M., 2011. Virus share.(2011). https://virusshare.com. [Online; accessed 28-March-2020].
Santoro A, Bartunov S, Botvinick M, Wierstra D, Lillicrap T. Meta-learning with memory-augmented neural networks, 4; 2016. p. 2740–51. arXiv:1605.06065v1.
Santos I, Penya YK, Devesa J, Bringas PG. N-grams-based file signatures for malware detection. In: ICEIS 2009 - 11th International Conference on Enterprise Information Systems, Proceedings AIDSS; 2009. p. 317–20. doi:10.5220/0001863603170320.
Sebastián M, Rivera R, Kotzias P, Caballero J. Avclass: a tool for massive malware labeling. In: International Symposium on Research in Attacks, Intrusions, and Defenses. Springer; 2016. p. 230–53.
Shijo PV, Salim A. Integrated static and dynamic analysis for malware detection. Procedia Comput. Sci. 2015;46(Icict 2014):804–11. doi:10.1016/j.procs.2015.02.149.
Snell J, Swersky K, Zemel R. Prototypical networks for few-shot learning, 2017-Decem; 2017. p. 4078–88. arXiv:1703.05175.
Sung F, Yang Y, Zhang L, Xiang T, Torr PH, Hospedales TM. Learning to compare: relation network for few-shot learning. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition; 2018. p. 1199–208. arXiv:1711.06025. doi:10.1109/CVPR.2018.00131.
Tang Z, Wang P, Wang J. ConvProtoNet: deep prototype induction towards better class representation for few-shot malware classification. Appl. Sci. (Switzerland) 2020;10(8). doi:10.3390/APP10082847.
Total, V., 2012. Virustotal-free online virus, malware and url scanner. https://www.virustotal.com. [Online; accessed 3-July-2020].
Tran K, Sato H, Kubo M. MANNWARE: a malware classification approach with a few samples using a memory augmented neural network. Information (Switzerland) 2020;11(1). doi:10.3390/info11010051.
Tran TK, Sato H. NLP-based approaches for malware classification from API sequences. In: Proceedings - 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems, IES 2017, Vol. 2017-Janua; 2017. p. 101–5. doi:10.1109/IESYS.2017.8233569.
Vasan D, Alazab M, Wassan S, Safaei B, Zheng Q. Image-based malware classification using ensemble of CNN architectures (IMCEC). Comput. Secur. 2020;92(March):101748. doi:10.1016/j.cose.2020.101748.
Vinyals O, Blundell C, Lillicrap T, Kavukcuoglu K, Wierstra D. Matching networks for one shot learning. In: Advances in Neural Information Processing Systems; 2016. p. 3637–45. arXiv:1606.04080.
Vu DL, Nguyen TK, Nguyen TV, Nguyen TN, Massacci F, Phung PH. A convolutional transformation network for malware classification. In: Proceedings - 2019 6th NAFOSTED Conference on Information and Computer Science, NICS 2019; 2019. p. 234–9. arXiv:1909.07227. doi:10.1109/NICS48868.2019.9023876.
Ye HJ, Hu H, Zhan DC, Sha F. Few-shot learning via embedding adaptation with set-to-set functions. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition; 2020. p. 8805–14. arXiv:1812.03664. doi:10.1109/CVPR42600.2020.00883.
Ye Y, Li T, Adjeroh D, Iyengar SS. A survey on malware detection using data mining techniques. ACM Comput. Surv. 2017;50(3). doi:10.1145/3073559.

Peng Wang received the M.S. degree in computer science and technology from Sichuan University, Chengdu, Sichuan, China, in 2010, where he is currently pursuing the Ph.D. degree in cyberspace security. He is currently involved in research work on malware classification and attacking malware detection. Her research interests include software security and network traffic analysis.

Zhijie Tang is pursuing his B.E. degree at the College of Software Engineering of Sichuan University, China. He has published some work about the few-shot malware classification and is also currently involving in the related researches. His research interests

include malware classification, meta learning and few-shot learning.

Junfeng Wang received the M.S. degree in computer application technology from the Chongqing University of Posts and Telecommunications, Chongqing, in 2001, and the Ph.D. degree in computer science from the University of Electronic Science and Technology of China, Chengdu, in 2004. From 2004 to 2006, he held a postdoctoral position with the Institute of Software, Chinese Academy of Sciences. Since 2006, he has been with the College of Computer Science and the School of Aeronautics and Astronautics, Sichuan University, as a Professor. His recent research interests include network and information security, spatial information networks, and data mining. He is currently serving as an Associate Editor of the IEEE ACCESS, the IEEE INTERNET OF THINGS, and Security and Communication Networks.