Meta-Learning for Multi-Family Android Malware Classification

YAO LI, School of Computer Science and Engineering, Macau University of Science and Technology, Taipa, Macao
DAWEI YUAN, School of Computer Science and Engineering, Macau University of Science and Technology, Taipa, Macao
TAO ZHANG, Macau University of Science and Technology, Taipa, Macao
HAIPENG CAI, Washington State University, Pullman, United States
DAVID LO, Singapore Management University, Singapore, Singapore
CUIYUN GAO, Harbin Institute of Technology Shenzhen, Shenzhen, China
XIAPU LUO, Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong
HE JIANG, School of Software, Dalian University of Technology, Dalian, China

With the emergence of smartphones, Android has become a widely used mobile operating system. However, it is vulnerable when encountering various types of attacks. Every day, new malware threatens the security of users' devices and private data. Many methods have been proposed to classify malicious applications, utilizing static or dynamic analysis for classification. However, previous methods still suffer from unsatisfactory performance due to two challenges. First, they are unable to address the imbalanced data distribution problem, leading to poor performance for malware families with few members. Second, they are unable to address the zero-day malware (zero-day malware refers to malicious applications that exploit unknown vulnerabilities) classification problem. In this paper, we introduce an innovative **meta**-learning approach for **m**ulti-family **A**ndroid **m**alware **c**lassification named **Meta-MAMC**, which uses meta-learning technology to learn meta-knowledge (i.e. the similarities and differences among different malware families) of few-family samples and combines new sampling algorithms to solve the above challenges. `Meta-MAMC` integrates (i) the meta-knowledge contained within the dataset to guide models in learning to identify unknown malware, and (ii) more accurate and diverse tasks based on novel sampling strategies, as well as directly adapting meta-learning to a new few-sample and zero-sample task to classify families. We have evaluated `Meta-MAMC` on two popular datasets and a corpus of real-world Android applications. The results demonstrate its efficacy in accurately classifying malicious applications belonging to certain malware families, even achieving 100% classification in some families.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Software and its engineering** → **Software notations and tools**.

[1]Yao Li[#] and Dawei Yuan[#] contributed equally to this work.
[2]Tao Zhang[*] is the corresponding author.

Authors' Contact Information: Yao Li, School of Computer Science and Engineering, Macau University of Science and Technology, Taipa, Macau, Macao; e-mail: 2109853gia30001@student.must.edu.mo; Dawei Yuan, School of Computer Science and Engineering, Macau University of Science and Technology, Taipa, Macau, Macao; e-mail: wu.xiguanghua2014@gmail.com; Tao Zhang, Macau University of Science and Technology, Taipa, Macau, Macao; e-mail: tazhang@must.edu.mo; Haipeng Cai, Washington State University, Pullman, Washington, United States; e-mail: chapering@gmail.com; David Lo, Singapore Management University, Singapore, Singapore, Singapore; e-mail: davidlo@smu.edu.sg; Cuiyun Gao, Harbin Institute of Technology Shenzhen, Shenzhen, Guangdong, China; e-mail: gaocuiyun@hit.edu.cn; Xiapu Luo, Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong; e-mail: csxluo@comp.polyu.edu.hk; He Jiang, School of Software, Dalian University of Technology, Dalian, China; e-mail: jianghe@dlut.edu.cn.

ACM /2024/5-ART
https://doi.org/10.1145/3664806

Additional Key Words and Phrases: Android, malware family, meta-learning, classification

1 INTRODUCTION

The Android operating system has become the world's most widely used mobile platform. As of January 2021, Android held over 70 percent of the smartphone market share, with projections indicating that it will maintain a leading market share of over 87% by 2023 [53]. However, this widespread usage has made Android a target for malicious attacks. According to recent findings by G DATA CyberDefense, cyber criminals released an Android malicious application every eight seconds in the first half of 2020 [8]. Additionally, Kaspersky discovered over 5 million mobile malicious installation packages in 2020 [12]. With the popularity and open-source nature of Android, the operating system is vulnerable to various types of attacks, such as credential theft, privacy disclosure, bank fraud, ransomware, adware, SMS fraud, and more. The risk of Android malware is a significant threat to users, and the situation is escalating as malware becomes increasingly contagious. Therefore, it is crucial to classify malware applications to ensure system security and safeguard user privacy [82].

Many proposals have attempted to classify Android malware families [29, 47, 62, 75, 78] (Android malware families refer to a group of malicious applications that targets the Android operating system and are designed to perform various malicious activities such as violating user privacy and property security) and Android zero-day malware (Android zero-day malware refers to malicious applications that exploit unknown vulnerabilities to perform malicious activities on Android devices) by discovering new attack pattern risks, designing new signatures, or identifying malicious code. For example, *Xu et al.* [78] classify the malware using Deep Learning (DL) techniques. *Martín et al.* [47] classify malware into families using classical machine learning such as Support Vector Machine (SVM) and DL algorithms. *Suarez-Tangil et al.* [62] use a Nearest Neighbor classifier (NN) to classify malware into families. Despite using advanced algorithms like deep learning for malware classification, they fail to address the issues of imbalanced samples within malicious families and incomplete families within the dataset. Additionally, some studies [20, 21, 49, 51] endeavor to leverage malware family information, as zero-day Android malware typically exhibits similar characteristics to existing Android malware. For instance, researchers have employed the approach of constructing an integrated classification framework based on malware family information to classify new malicious software [37]. Although they aim to classify zero-day malware by learning family characteristics, the incomplete datasets they used could not provide a reliable training environment. Meanwhile, since they do not take into account the knowledge from the few-sample families, the classification performance on new and previously unknown families is not optimal. Therefore, two challenges need to be addressed.

Challenge 1: Sample imbalance between families and missing malware families. Previous research endeavors in malware family classification have primarily focused on feature extraction techniques. However, these studies have often overlooked two critical aspects. Firstly, there exists a substantial discrepancy between the number of malicious families available in the dataset and the actual count of known malicious families. As mentioned earlier, the number of Android malware families exceeded 1,000 in 2018 [61], and it continues to rise without any signs of decline. Consequently, the dataset's representation of malware families remains considerably smaller than the total population, leading to limited coverage and a significant absence of a large portion of malware families.

Secondly, the distribution of family samples within the dataset exhibits a high degree of imbalance, primarily attributed to the scarcity of available samples for the majority of Android malware families. To exemplify this observation, consider the renowned Drebin dataset [4], which encompasses 179 malware families. Figure 1 illustrates the substantial variation in sample sizes among the top 20 families. Specifically, the sample sizes within the Drebin dataset exhibit a notable range, spanning from tens to hundreds of samples for the top 20 families alone. Furthermore, our survey reveals that nearly 58% of Android malware families within the Drebin dataset contain

fewer than five samples. This severe scarcity of malicious families, coupled with the imbalanced distribution of samples, poses a significant challenge for models to establish a realistic and reliable training environment.

Consequently, the scarcity of certain malicious families and the disproportionate representation of samples across families present substantial hurdles for models seeking to achieve robust and accurate performance in the domain of malware family classification.

Challenge 2: Difficulty in classifying zero-day malware. Android zero-day malware refers to malicious applications that exploit unknown vulnerabilities in the Android operating system or third-party applications to perform malicious activities on Android devices [56]. These vulnerabilities are unknown to the public and therefore have not been patched by application developers or security researchers. Zero-day malware applications are highly difficult to be classified because they use new and unknown attack methods. Although previous studies [26, 59, 73] classify zero-day malware by using features and information of malware families, it is difficult to achieve good results due to the lack of some malware family samples. Additionally, they do not consider the meta-knowledge in the samples, which leads to the low generalization ability of the models and difficulty in classifying zero-day malware. With the industrialization of Android malware creation, numerous Android malware applications are generated using dozens of commercial tools. However, it is inefficient to classify zero-day malware samples by finding their similarity to known malware samples. For example, VirusTotal [55] assigns known family attributes to zero-day malware samples based on the results of built-in antivirus tools. Therefore, it is difficult for them to classify zero-day malware, and it always takes a long time to learn unknown malware in the latest malware families.

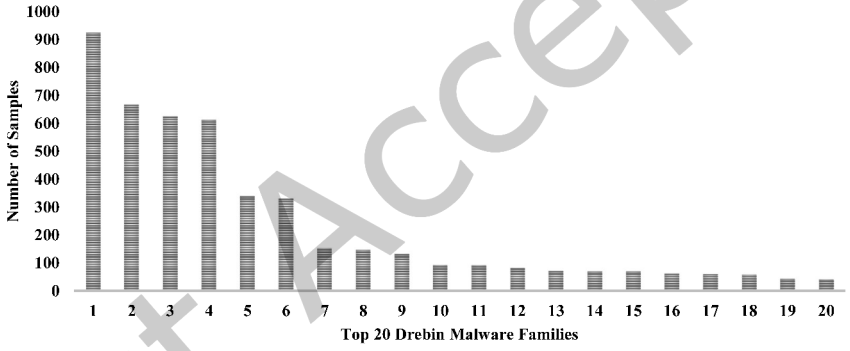

Fig. 1. The sample size of Drebin's top 20 families.

Although, some scholars propose to use meta-learning and few-shot learning to solve the problems encountered in Android malware detection. Zhu et al. [88] proposed a task-adaptive meta-learning approach to address the problem of obfuscated malware classification. But it doesn't work well against new and unknown malware families. Wang et al. [72] proposed a dynamic analysis-based few-shot learning method for unknown malware classification. Although it can better classify unknown samples, it greatly increases the time of early sample processing. Our solution is based on static analysis, which reduces the processing cost in the early stage, and at the same time can classify new malware and unknown malware families well.

Therefore, the above analysis motivates us to investigate: **To what extent, can we classify malware under limited imbalanced datasets (sample imbalance between families and missing some malware families)?**

In order to resolve the aforementioned problems, we propose a multi-family classification.[1] scheme for Android malware based on meta-learning (meta-learning refers to a machine learning technique that involves creating

[1]Multi-family classification is to classify Android malware into multiple families

algorithms or models capable of learning how to learn), called Meta-MAMC. Firstly, reverse engineering is used to decompile the APK files and extract features from the *AndroidManifest.xml* file and *class.dex* file. Then, we design **two new sampling strategies**, an application-based sampling strategy (sample an equal number of instances from the training dataset and split them into two distinct sets: a support set and a query set) and a family-based sampling strategy (uniformly sample a family each time, then randomly select a sample containing that family and add it to the task instance set. Repeat this process several times), to alleviate the sample imbalance. In this way, the model can be utilized to analyze few-sample families, which solves **Challenge 1**. Next, we design an improved meta-learning algorithm, Meta-MAMC algorithm. Meanwhile, we utilize the dual sampling strategies to construct the training tasks and regulate the proportion of the two sampling strategies by setting the hyperparameter p. This can control the model to focus on few-sample and zero-sample families while not ignoring other malicious families. Then a randomly initialized model performs meta-learning on a few samples and multi-family learning tasks sampled from a specific sampling strategy to optimize parameters. The meta-learning model is fine-tuned using traditional supervised learning on the training set of the final task. This process allows the model to learn from multiple low-sample tasks and generalize better to new tasks, making it address **Challenge 2**. Finally, following improved meta-learning training and fine-tuning, a meta-classifier is obtained and subsequently used for classification.

We conduct extensive experiments to evaluate the new Meta-MAMC method from several aspects, such as identification accuracy and effectiveness in handling zero-day family malware. We compare our method, Meta-MAMC, with several state-of-the-art baselines, such as Drebin [4], MaMaDroid [46], N-opcode [32], and EC2 [10]. We evaluate both large and small families (with a family size greater than 9 considered as large and less than 10 considered as small [10, 17]), and the results show that the classification effect of Meta-MAMC exceeds that of advanced methods in both large and small families. Significantly, our proposed approach exhibits exceptional performance in classifying small malware families, surpassing that of state-of-the-art methods. Remarkably, some families achieve a remarkable 100% detection rate, underscoring the efficacy of our approach. The main contributions of this paper are as follows:

- We propose a meta-learning algorithm Meta-MAMC for multi-family Android malware classification tasks. The Meta-MAMC consists of two phases: the meta-learning phase and the fine-tuning phase. To our best knowledge, we are the first study to address these challenges in multi-Family Android malware classification tasks from the meta-learning point of view.
- Considering the imbalance among malware families and missing malware families, we innovatively use zero-sample scenarios to represent missing and unknown malware families, and design two new sampling strategies as well as control the number of families to alleviate the imbalance between different families.
- We conduct extensive experiments to assess the effectiveness of Meta-MAMC. The experimental results demonstrate that Meta-MAMC outperforms the state-of-the-art schemes. We have made the datasets, code, and collected unknown malware publicly available at the following repository[2]

The remainder of this article is structured as follows. Section 2 introduces the relevant information on meta-learning and Android malware families. The proposed approach is presented in Section 3. Section 4 describes the research question, experimental setup, and evaluation metrics. Experimental results are shown in Section 5. The proof of zero-day malware classification is presented in Section 6. The further analysis of the proposed approach is discussed in Section 7. Section 8 presents the related work. Finally, Section 9 provides a summary of the proposed approach in this paper.

[2]https://figshare.com/articles/dataset/meta_learning_for_malware_classification/25036160

2 BACKGROUND

2.1 Meta-Learning and Model-Agnostic Meta-Learning

Meta-learning, also known as "learning to learn", is a subset of machine learning in computer science [27, 66]. It is used to improve the results and performance of a learning algorithm by changing some aspects of the learning algorithm based on experiment results. Meta-learning helps researchers understand which algorithm(s) generate the best/better predictions from datasets. Meta-learning consists of two layers, a base learner and a meta-learner. The base learner, the model in the base layer, considers the data set on a single task each time the base learner is trained. The Meta-Learner, a model in the meta-layer, summarizes the training experience on all tasks. Each time the base learner is trained, the meta-learners synthesize new experiences and update the parameters in the meta-learners.

Model-Agnostic Meta-Learning (MAML) [22] is an algorithm in meta-learning. MAML is a technique for few-shot learning, which is the ability to learn new concepts from a small number of examples. Its core idea is to leverage a set of auxiliary tasks to search for a good parameter initialization from which learning a target task would require only a handful of training samples. In the MAML algorithm, the data is organized into tasks, and each task is further divided into a support set and a query set. These divisions play a crucial role in enabling meta-learning and few-shot learning capabilities.

Task Division. In the case of Android malware families classification, the task division involves dividing the dataset into distinct tasks, with each task representing a specific malware family. Each malware family is considered as an individual classification task.

Support Set. Within each malware family task, the support set consists of labeled malware samples used for model adaptation. The support set includes a limited number of labeled examples from each malware family. These labeled samples serve as the training data for the model to update its parameters during the adaptation process. The model learns to adapt to the specific characteristics and behavior of each malware family through the support set.

Query Set. In each malware family task, the query set is a subset of examples used for evaluating the model's performance. The query set consists of both labeled and unlabeled malware samples. It is used to assess the model's ability to generalize and accurately classify unseen malware samples. The model's adaptation to the support set helps improve its performance on the query set by capturing the specific patterns and characteristics of each malware family.

Formally, MAML first meta-learns the initialization of model parameters θ_0 with auxiliary tasks $\{\mathcal{T}_1, \cdots, \mathcal{T}_i\}$ and continues to learn the optimized parameters θ^* for the target task:

$$\theta^* = \text{Learn}\left(\mathcal{T}_t; \text{MetaLearn}\left(\mathcal{T}_1, \cdots, \mathcal{T}_i; \theta_0\right)\right) \tag{1}$$

Once the initialization is learned, the model can be quickly adapted to new tasks by performing a few additional gradient updates on the task-specific data. By fine-tuning the model in this way, it can achieve high performance on new tasks with only a small number of training examples. MAML has been successfully applied to a variety of tasks, including image classification [28] and object detection [39]. Its ability to learn quickly from a few examples makes it a promising technique for applications where labeled data is scarce, such as in Android malware family classification.

Android Malware and Malware Family. Android malware is malicious software that targets smartphone devices running the Android operating system. It is like other malware samples that run on desktops or laptop computers. Android malware is alternatively called mobile malware which is any piece of malicious software intended to harm the mobile device by performing some illegitimate activities. It can be classified into different malware categories such as adware, backdoor, file infector, potentially unwanted application (PUA), ransomware, riskware, scareware, spyware, trojan, etc. Each malware category has some unique characteristics that differentiate

it from the other malware categories. Every malware category has several malware families associated with it. The prominent Android malware categories include *adware*, *backdoor*, *file infector*, *PUA*, *ransomware*, *riskware*, *scareware*, *spyware*, *trojan*, *trojan-sms*, *TrojanSMS.Denofow*, *TrojanSMS.Stealer*, and *TrojanSMS.Hippo*.

3 METHODOLOGY

3.1 Overview

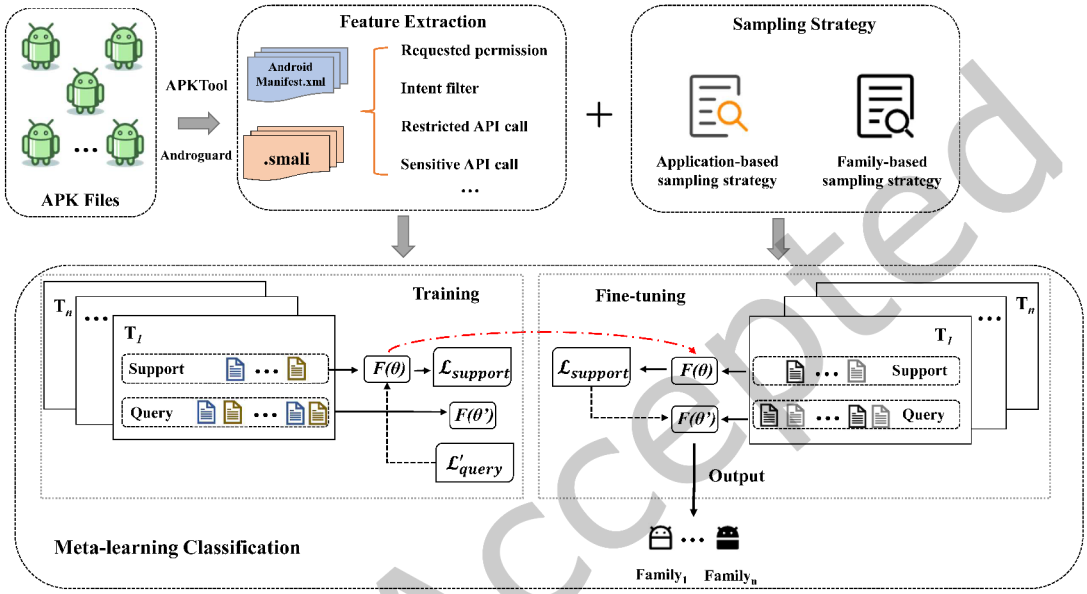

Fig. 2. Main flowchart of our proposed Meta-MAMC for Android malware family classification.

As shown in Figure 2, the proposed approach for Android malware family classification in this paper involves four steps. 1) `Feature Extraction`, reverse engineering is used to decompile the APK files and extract features from the *AndroidManifest.xml* file and *class.dex* file. Eight categories of features are extracted from these files; 2) `Sampling Strategy`, to address the issue of sample imbalance in the training dataset, we have devised two novel sampling strategies: the application-based sampling strategy and the family-based sampling strategy. In the application-based sampling strategy, we randomly select an equal number of instances from the training dataset. These instances are then divided into two distinct sets: the support set and the query set. This approach ensures that both sets contain a balanced representation of instances from different applications. On the other hand, the family-based sampling strategy aims to achieve a more balanced representation at the family level. In this strategy, we uniformly sample a family from the available families in the training dataset. Subsequently, we randomly select a sample that belongs to the chosen family and add it to the task instance set. This process is repeated multiple times to ensure the inclusion of various samples from different families; 3) `Improved Meta-Learning`, `Meta-MAMC` adopts a dual sampling strategy to generate training tasks that encompass both few-sample family tasks and zero-sample family tasks, thereby increasing the realism of the training environment. We utilize the dual sampling strategies to construct the training tasks and regulate the proportion of the two sampling strategies by setting the hyperparameter p. This can control the model to focus on few-sample and zero-sample families while not ignoring other malicious families. A randomly initialized model performs meta-learning on few-sample

Table 1. Features of eight categories

Source	Category	Description
Manifest.xml	Hardware & software resources	Application requirements for system resources, including hardware and software resources
	Requested permission	If an application must execute some specified operations, it must request corresponding permissions in the manifest.xml
	Application components	Application components are essential building blocks of an application; the four components are activities, services, providers, and receivers
	Intent filter	An intent filter declares the capabilities of its parent component—what an activity or service can do and what types of broadcasts a receiver can handle
Dalvik bytecode files (.smali)	Restricted API call	Some key API calls are restricted by system permissions
	Used permission	Access to restricted API calls requires some appropriate permissions to be granted
	Sensitive API call	Some API calls can access sensitive data and resources on mobile devices
	Sensitive shell command	Some shell commands can gain administrative privileges and execute dangerous operations

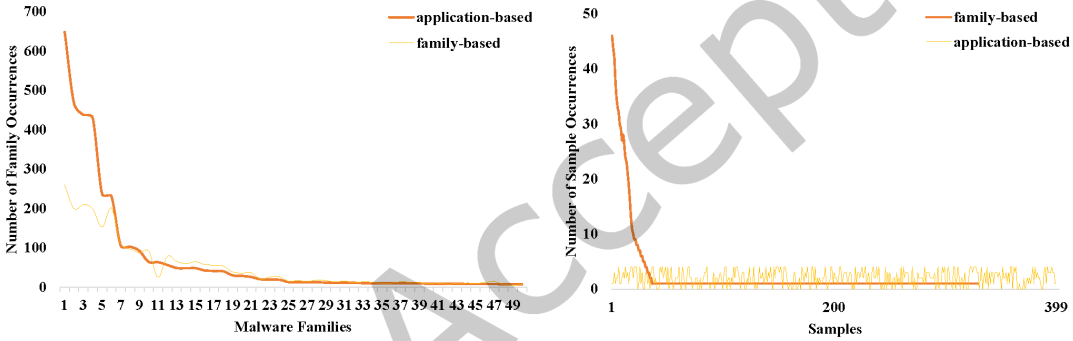

Fig. 3. The number of times each family and sample occurring in the tasks sampled from Drebin according to the application-based or family-based sampling strategy.

and multi-family learning tasks sampled from a specific sampling strategy to optimize parameters. Then the model is fine-tuned using traditional supervised learning on the training set of the final task; 4) `Classification`, after improved meta-learning training and fine-tuning, a meta-classifier is obtained and subsequently used for classification.

3.2 Feature Extraction

The APK file is the installation package of the Android application, which contains the program code and resources which is necessary for program execution. We decompile the APK file with the reverse engineering tools Apktool [67] and Androguard [15] to obtain an *AndroidManifest.xml* file and *Class.dex* files, and we convert the *Class.dex* files into readable *Dalvik bytecode* files (i.e., *.smali*). The *AndroidManifest.xml* file is a configuration file, which contains information, such as permissions and communication components required for program execution. Each *.smali* file is equivalent to a Java *.class* file. Referring to the existing studies [38], [34], eight categories of features are extracted from these files, as listed in Table 1.

3.3 Sampling Strategy

As mentioned earlier, the extension of meta-learning algorithms for multi-family classification problems needs to consider two challenges: 1) Sample imbalance between families and missing malware families; 2) Difficulty in classifying zero-day malware. **However, meta-learning algorithms do not take into account specific data distributions. The tasks constructed through their naive task sampling strategy only consider a limited number of frequent labels, reducing task diversity.** Meanwhile, there are too many Android malware families to collect all samples for each family, and new malware is added every day. This leads to zero-sample scenarios not considered by existing algorithms, providing less faithful training conditions. Therefore, we design two sampling strategies, an application-based sampling strategy, and a family-based sampling strategy. These two sampling strategies can uniformly extract each sample and take into account families with fewer samples and zero samples. **We utilize the dual sampling strategies to construct the training tasks and regulate the proportion of the two sampling strategies by setting the hyperparameter** p (For details about the hyperparameter p, please refer to Section 5.3). We incorporate both sampling strategies because using only one strategy can result in the model ignoring certain families during training. For instance, when only the application-based sampling strategy is adopted, it becomes challenging to select few-sample families, leading to inadequate learning of these families by the model. Thus, we use both strategies simultaneously.

Application-based sampling strategy: A handful of samples are uniformly sampled from the original Android malware dataset D. Each sample x has a corresponding label y. This strategy can make the sampling results more consistent with the final task, that is, include both few-sample and zero-sample families. Then D is partitioned into two disjoint sets, i.e., the support set $D^S_{\mathcal{T}_i}$ and the query set $D^Q_{\mathcal{T}_i}$, as shown in Equations 2 and 3:

$$D^S_{\mathcal{T}_i} = \left\{(\mathbf{x}_n, \mathbf{y}_n) \mid \mathbf{y}_n \subset \mathcal{F}^S_{\mathcal{T}_i}\right\}_{n=1}^{N} \tag{2}$$

$$D^Q_{\mathcal{T}_i} = \left\{(\mathbf{x}_k, \mathbf{y}_k) \mid \mathbf{y}_k \subset \mathcal{F}^Q_{\mathcal{T}_i}\right\}_{k=1}^{K} \tag{3}$$

where T_i represents multi-family classification tasks, N and K are the number of instances in the support set and query set respectively. Then, $\mathcal{F}^S_{\mathcal{T}_i} = \bigcup_{n=1}^{N} \mathbf{y}_n$ and $\mathcal{F}^Q_{\mathcal{T}_i} = \bigcup_{k=1}^{K} \mathbf{y}_k$ are the corresponding labels of support set and query set in T_i.

We have found empirically that this strategy can construct more faithful tasks in which few- and zero-shot scenarios coexist, i.e., $\mathcal{F}^Q_{\mathcal{T}_i} \cap \mathcal{F}^S_{\mathcal{T}_i} \neq \emptyset$ and $\mathcal{F}^Q_{\mathcal{T}_i} - \mathcal{F}^S_{\mathcal{T}_i} \neq \emptyset$, with a high probability. However, this strategy is still affected by the long-tailed label distribution of Android malware families, as shown in the upper part of the Figure 3: the few-sample families in the training set have fewer chances to appear in the tasks and models are more susceptible to overfitting to a handful of frequent families.

Family-based sampling strategy: A family is first sampled from the family space F_{space}, and then an instance annotated with this label is selected. We repeat this process $N+K$ times to construct $\tau_i = (D^{tr}_{\mathcal{T}_i}, D^{val}_{\mathcal{T}_i})$. The left part of Figure 3 shows that the family-based one is fairer than the application-based one from the family dimension. On the other hand, the right part of Figure 3 reveals that the application-based one shows no biases to samples, while the family-based one pays too much attention to those samples mostly annotated with few-sample families.

The sampling ratio p allows us to balance between the application-based and family-based strategies and provides a way to control the diversity of the tasks. If p is set to 0, all tasks will be constructed using the family-based strategy, which can lead to less diversity in the tasks. On the other hand, if p is set to 1, all tasks will be constructed using the application-based strategy, which can also lead to less diversity in the tasks. By setting an appropriate value for p, we can strike a balance between the two strategies and generate tasks that are both diverse and faithful to the real-world scenario of Android malware family classification. To ensure comprehensive model training without sacrificing efficiency, we initially set a relatively elevated threshold, ϵ, at 0.01. Because,

when ϵ is less than 0.01, the performance improvement of the model is not obvious after each iteration and the convergence is very slow. Therefore, we decided to set ϵ to 0.01 to train the model more thoroughly. During the training process, we find that models converge quite quickly, and the loss could converge in a relatively few epochs, and when the number of iterations exceeded 20, the model performance no longer improved significantly. Therefore, to avoid wasting resources, we set N_{max} to 20, which helps prevent ineffective repeated training.

Algorithm 1 Meta-MAMC

Input: Dataset D, learning rates α, β and sampling strategy S

Initialization: $\theta \leftarrow \theta_0$
// Meta-learning
while the difference in θ over iterations is greater than a threshold ϵ or the number of iterations is less than N_{max} **do**
  Simulate a batch of few-sample multi-family Android malware classification tasks from D using strategy S
  for each task T_i, with Support Set S_i and Query Set Q_i drawn from D **do**
    Compute local parameters $\theta_i{}'$ using learning rate α and a specified update rule
  end for
  Update model parameters θ using learning rate β and a global update rule
end while
// Fine-Tuning
Fine-tune the model with $\theta_i{}'$ on D

Output: Model M_{meta}

The *Meta-MAMC* algorithm 1 is a sophisticated meta-learning framework tailored for classifying Android malware across various families. It commences with the initialization of model parameters and progresses through an iterative meta-learning phase. This phase involves simulating classification tasks with few samples, updating model parameters locally for each task, and then globally to capture overarching patterns. The algorithm uniquely adapts to new malware types with minimal data, balancing between task-specific learning and generalization. Finally, it fine-tunes the model on a given dataset, culminating in a robust, adaptable model for Android malware classification.

3.4 Improved Meta-Learning and Classification

The original meta-learning algorithm, MAML, does not consider both few-sample and zero-sample families. Therefore, we design an improved MAML algorithm (`Meta-MAMC`) for meta-learning (by combining a designed sampling strategy) to address this issue. Algorithm 1 shows an overall procedure of `Meta-MAMC`, which consists of a meta-learning phase and a fine-tuning phase. We describe the meta-learning stage in detail here. Suppose there is a model F_θ with parameters θ and a task sampling strategy S which generates tasks T_i. For each task $T_i = \left(D^S_{\mathcal{T}_i}, D^Q_{\mathcal{T}_i}\right)$, we first update the model parameters using one-step gradient descent as

$$\theta_i' = \theta - \alpha \nabla_\theta L_\theta \left(D^S_{\mathcal{T}_i}\right) \tag{4}$$

where α is the local learning rate and L is the loss function. After that, the loss of local parameters on the corresponding query set is computed, i.e., $L_{\theta_i'}\left(D^Q_{\mathcal{T}_i}\right)$. Finally, the global parameters are obtained using the loss across multiple tasks, i.e.,

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{D^Q_{\mathcal{T}_i}} L_{\theta'_i} \left(D^Q_{\mathcal{T}_i} \right) \tag{5}$$

where β is the global learning rate. In short, `Meta-MAMC` explicitly simulates the few-sample multi-family Android malware classification tasks and directly incorporates the objective of adapting to these tasks into the meta-learning optimization phases. This encourages models to learn metaknowledge, i.e., how to obtain maximal performance on these rare/unseen families with little training data. Therefore, our enhanced MAML algorithm can effectively address this issue. By integrating the two sampling strategies, we generate tasks that include both few-sample and zero-sample samples, thereby creating a more realistic training environment.

Finally, with the improved meta-learning algorithm `Meta-MAMC`, we obtain a meta-classifier and classify malware families and classify zero-day malware.

4 EXPERIMENT SETUP

This section encompasses the introduction of the state-of-the-art (SOTA) method, the formulation of the research question, the description of the experimental setup, and the delineation of the evaluation metrics utilized in this study.

4.1 SOTA Method

To conduct a comprehensive evaluation of our proposed scheme, we have carefully selected four popular methods, namely Drebin [4], MaMaDroid [46], N-opcode [32], and EC2 [10]. These models have been chosen due to their shared focus on security and privacy-related features for classifying malware samples. The primary objective of our evaluation is to assess the effectiveness of our approach in accurately assigning malware samples to their respective families.

In our evaluation, each malware sample is assigned a family label based on its closest resemblance to the capability vectors associated with the various malware families. Subsequently, the predicted family labels are compared against their corresponding capability labels, and the performance is quantified by evaluating the concordance between the ground truth capability and the predicted capability.

This evaluation framework enables us to gauge the efficacy of our proposed scheme in accurately classifying malware samples and associating them with their corresponding capability labels, thereby providing a robust assessment of its performance in comparison to the selected state-of-the-art methods. The following are the details of these methods.

- Drebin [4] is a lightweight method proposed to address the security threat posed by malicious applications on the Android platform. As the proliferation of such applications hampers conventional defense mechanisms, Android smartphones often lack adequate protection against emerging malware. To address this issue, Drebin performs an extensive static analysis by gathering a wide range of application features. These features are then embedded in a unified vector space, allowing for the identification of typical patterns indicative of malware.
- MaMaDroid [46] is a static analysis-based system designed to detect malware targeting the Android platform. MaMaDroid takes a behavioral approach to malware detection by focusing on the sequence of abstracted API calls performed by an app. The system abstracts the API calls into their corresponding class, package, or family, and constructs a model using the sequences obtained from the call graph of an app, treating them as Markov chains. This approach enhances the resilience of the model to changes in the underlying APIs and results in a manageable feature set.
- N-opcode [32] is a method developed for the detection and classification of Android malware, addressing the increasing problem of malware on the popular Android mobile platform. With the accessibility of

numerous third-party app markets and the adoption of sophisticated detection avoidance techniques by emerging malware families, more effective techniques are needed. The approach utilizes automated feature discovery, eliminating the need for manual expert or domain knowledge to define the required features.

- EC2 [10] is a novel algorithm developed for the discovery of Android malware families of varying sizes, ranging from large to small families, including previously unseen ones. With the increasing number of Android malware variants, the detection of malware families plays a crucial role in enabling security analysts to identify situations where signatures of a known malware family can be adapted instead of manually inspecting the behavior of all samples. Experimental results demonstrate that EC2 accurately detects both small and large malware families, surpassing several comparative baselines.

4.2 Research Questions

We conduct a comprehensive evaluation of our program by answering three research questions (RQ). The RQs are as follows:

- **RQ 1**: *How does* `Meta-MAMC` *perform on Android malware classification?*
- **RQ 2**: *How does the performance of* `Meta-MAMC` *compare to that of the original MAML? And how does it compare to state-of-the-art methods?*
- **RQ 3**: *Can* `Meta-MAMC` *be effective in combating the evolution of malware and classifying zero-day malware?*

For **RQ 1**, we want to evaluate whether an improved meta-learning algorithm, `Meta-MAMC`, can effectively classify Android malware. It not only includes multiple categories of Android malware families but also includes two categories of malware and benign applications, so as to comprehensively evaluate the classification effect of `Meta-MAMC`. Section 5.1 describes the detailed evaluation process and results. Regarding **RQ 2**, we compare the performance of `Meta-MAMC` and the original MAML algorithm in family classification. Furthermore, we select several state-of-the-art methods to compare with `Meta-MAMC` to evaluate its performance. For details, please refer to Section 5.2. Finally, for **RQ 3**, we collect unknown malware for testing. We then evaluate whether the improved `Meta-MAMC` can generalize effectively to new unknown malware. Section 6 describes the details.

4.3 Experimental Setting

Table 2. **Summary of Android Malware Datasets**

Dataset	Malware Families	Malware Samples	Benign Samples
Drebin	179	5,560	123,456
AMD	71	24,553	*N/A*
PlayDrone (Benign)	*N/A*	*N/A*	10,000

To evaluate the effectiveness of our proposed method, we conduct experiments on two publicly available Android malware datasets, which show in Table 2. We follow the previous studies [35, 42, 81] and use the public malware dataset: Drebin dataset [4] and AMD dataset [73]. The Drebin dataset contains a total of 5,560 malware samples from 179 families and 123,456 benign samples. The AMD dataset contains 71 families with a total of 24,553 malware samples. The benign samples are collected from the PlayDrone [69]. The PlayDrone is a measurement study of the Google Play Store. It indexes and analyzes over 1,100,000 applications in the Google Play Store on a daily basis, and is the largest such index of Android applications. We conduct a random download of 10,000 popular benign applications from the platform, which represents a significantly larger sample size compared to

previous studies [10, 46]. This diverse collection of applications spans across multiple categories, such as news, finance, education, games, sports, music, shopping, weather, and more.

We follow the previous studies [19, 44] and randomly split the data into training and testing sets with a ratio of 7:3. We use 70% of the training data to construct tasks for the meta-learning phase and fine-tune the model on the remaining 30%. We employ a five-fold cross-validation approach [25], where the experiments are repeated five times to ensure robustness, and the average results are reported. We compare our method, `Meta-MAMC`, with several state-of-the-art baselines, including Drebin [4], MaMaDroid [46], N-opcode [32], and EC2 [10].

Moreover, we utilize a text-based Convolutional Neural Network (TextCNN) for classification [14]. The model features 3x3 convolutional layers with 64 filters, batch normalization, and ReLU non-linearity. It employs 2x2 max-pooling and has a last hidden layer dimension of 64. The output layer is a softmax layer for classification purposes. Additionally, the model includes a non-convolutional variant with four hidden layers of sizes 256, 128, 64, and 64. The chosen loss function is cross-entropy, and the model is fine-tuned with additional hyperparameters for optimal performance.

The subsequent experiments are conducted on a personal computer integrated with 13th Gen Intel(R) Core(TM) i9-13900K 3.00 GHz and NVIDIA GeForce RTX 4090. The computer has 32GB of memory and 2TB of storage. The deep neural networks are implemented using Scikit-learn, and Pytorch.

4.4 Evaluation Metrics

There are four common metrics adopted to evaluate the performance of classification: accuracy, precision, recall rate, and F-score. The malicious sample is denoted as the positive (P) class and the benign sample is denoted as the negative (N) class. Then, four numbers are obtained:

- True Positive (TP): the number of positive samples that are correctly predicted as positive.
- False Negative (FN): the number of positive samples that are incorrectly predicted as negative.
- False Positive (FP): the number of negative samples that are incorrectly predicted as positive.
- True Negative (TN): the number of negative samples that are correctly predicted as negative.

Based on these designations, the following equations calculate four common metrics:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}} \tag{6}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8}$$

$$F\text{-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9}$$

5 EXPERIMENTAL RESULTS

5.1 Performance of Android Malware Classification

In order to answer RQ 1, we evaluate the effectiveness of `Meta-MAMC` on malware instances from Drebin [4] and AMD [73] containing real malware collected from various sources. All these malicious applications belong to known malware families such as DroidKungFu, Geinimi, and GoldDream among others. To perform this experiment, we extract the eight categories of features in Table 1 to model the training process on these malware applications. The results of our experiment are presented in Table 3 and Table 4. The first column lists the malware families and the second column displays the number of analyzed instances belonging to each family. Tables 3 and 4 show the classification results of large and small families. To conduct a comprehensive evaluation of `Meta-MAMC`, we conduct a random selection of 20 families, encompassing both large and small families. For the

Table 3. Evaluation of Meta-MAMC on malware from Drebin

Malware Family	Family Size	Accuracy	Precision	Recall	F-Score
FakeInstaller	648	0.976	0.964	0.968	0.966
DroidKungFu	467	0.992	0.979	0.980	0.979
Plankton	438	0.963	0.961	0.961	0.961
Opfake	429	0.971	0.968	0.965	0.966
BaseBridge	237	0.945	0.941	0.946	0.943
GinMaster	231	0.966	0.963	0.959	0.961
Kmin	106	0.972	0.968	0.970	0.969
Iconosys	103	0.959	0.955	0.958	0.957
FakeDoc	92	0.978	0.973	0.976	0.975
Adrd	64	0.991	0.982	0.985	0.983
Imlog	30	0.966	0.953	0.962	0.983
Jifake	20	0.909	0.913	0.921	0.954
Fakelogo	13	0.876	0.866	0.884	0.923
FakePlayer	11	0.769	0.833	0.901	0.870
Zitmo	9	0.700	0.875	0.778	0.824
DroidSheep	7	0.857	0.846	0.853	0.923
AccuTrack	7	1	1	1	1
Stiniter	6	0.833	1	0.831	0.910
Maistealer	1	1	1	1	1
SmsSpy	1	1	1	1	1

purpose of this evaluation, a family size greater than 9 is considered as a large family, while a family size smaller than 10 is categorized as a small family [10, 17]. For instance, as shown in Table 3, the performance of Meta-MAMC in classifying malware instances across large families demonstrates an average accuracy of 0.945 and an average F-score of 0.948. These results highlight the effective classification ability of Meta-MAMC for large families of malware. Similarly, when dealing with small families, Meta-MAMC achieves an average accuracy of 0.898 and an average F-score of 0.943.

Moreover, on the AMD dataset, Meta-MAMC exhibits promising performance. For large families, the average accuracy is reported as 0.953, along with an average F-score of 0.952. In the case of small families, the average accuracy is 0.975, and the average F-score is 0.905. These results further validate the effectiveness of Meta-MAMC in accurately classifying both large-family and small-family malicious samples across both datasets.

However, a closer look at the results reveals that Meta-MAMC performs poorly on the BaseBridge malware family. Specifically, out of the 237 samples analyzed, only 223 are classified as instances of BaseBridge. Further inspection of this family reveals that many instances dynamically load code to perform malicious functionality, which is not detectable by static analysis and results in many false negatives. Excluding BaseBridge from the results of our sample achieves an average accuracy of 96.9%.

For other malware families with false negatives, several factors contribute to classification errors. Firstly, our signatures may not capture the essential characteristics of all instances of a family, as they are based on a small number of samples. Secondly, the malware family may have key features that are not expressible in our specification language. Lastly, missing models may cause false negatives as our static analysis relies on method stubs that do not cover all relevant behavior of the Android SDK.

Table 4. Evaluation of Meta-MAMC on malware from AMD

Malware Family	Family Size	Accuracy	Precision	Recall	F-Score
Airpush	2744	0.963	0.958	0.962	0.960
Dowgin	1724	0.965	0.961	0.956	0.959
FakeInst	1520	0.959	0.953	0.954	0.953
Mecor	1274	0.964	0.958	0.962	0.960
Youmi	910	0.953	0.949	0.953	0.951
Fusob	893	0.966	0.964	0.966	0.965
Kuguo	839	0.972	0.968	0.964	0.966
BankBot	453	0.976	0.973	0.974	0.973
Jisut	392	0.981	0.980	0.976	0.978
Mseg	164	0.987	0.984	0.985	0.985
Leech	89	0.976	0.982	0.989	0.988
Koler	48	0.959	0.979	0.979	0.979
Ksapp	25	0.880	0.917	0.957	0.937
Zitmo	16	0.765	0.813	0.929	0.867
Cova	11	0.909	1	0.909	0.952
FakeTimer	8	0.636	0.700	0.875	0.778
Opfake	7	0.875	0.875	1	0.933
Ogel	4	1	1	1	1
Tesbo	2	1	1	1	1
Fobus	1	1	1	1	1

Table 3 and Table 4 show that Meta-MAMC reports very few false positives. In the 30 malicious families of the two datasets, Meta-MAMC achieves an accuracy rate of over 94% for each family, with some families reaching over 99%. This indicates that the static analysis of Meta-MAMC is precise enough, and meta-learning can effectively learn meta-knowledge from the dataset.

From the results presented in Table 3 and Table 4, it is evident that the performance of Meta-MAMC on multi-family classification is outstanding. This observation confirms the effectiveness of our proposed sampling strategy and Meta-MAMC in addressing the issue of imbalance in Android malware families. Moreover, the family-based sampling strategy enhances the generalization ability of Meta-MAMC, making it more suitable for classifying unknown malware and their families. Moreover, the decent F-scores of BaseBridge and the low scores of FakeTimer are largely attributed to the quantity of samples and the distinctness of their features. Specifically, the FakeTimer dataset is characterized by high feature noise and poor differentiation, resulting in lower F-scores. In contrast, BaseBridge, with its clearer features and less noise, demonstrates better performance. The analysis encompasses various aspects, including sample quantity, uniqueness and diversity of features, representativeness of training data, overfitting issues, and challenges related to class imbalance.

Answer to RQ 1: The Meta-MAMC approach demonstrates efficient classification capabilities for Android malware families, irrespective of their size. Notably, on the Drebin dataset, the average accuracy achieved by Meta-MAMC was 0.898, accompanied by an average F-score of 0.943. Similarly, on the AMD dataset, the approach achieved an impressive average accuracy of 0.975, with an average F-score of 0.905. These results highlight the strong performance and effectiveness of the Meta-MAMC approach in accurately classifying Android malware families on both datasets.

5.2 Meta-MAMC Versus Original MAML and SOTA Methods

In order to answer RQ 2, we further verify the effect of Meta-MAMC in multi-family classification. While Meta-MAMC is based on meta-learning, it still diverges from other Android malware family classifiers. One of the notable improvements in Meta-MAMC is its employment of the new sampling techniques we developed to balance the distribution of diverse families and early samples. Moreover, it takes a more innovative approach by considering the zero-sample scenario, which makes it more effective in classifying unknown malware.

To demonstrate the effectiveness of our proposed approach, we conduct a comparison with the original MAML algorithm, and the results are presented in Figure 4 and Figure 5.

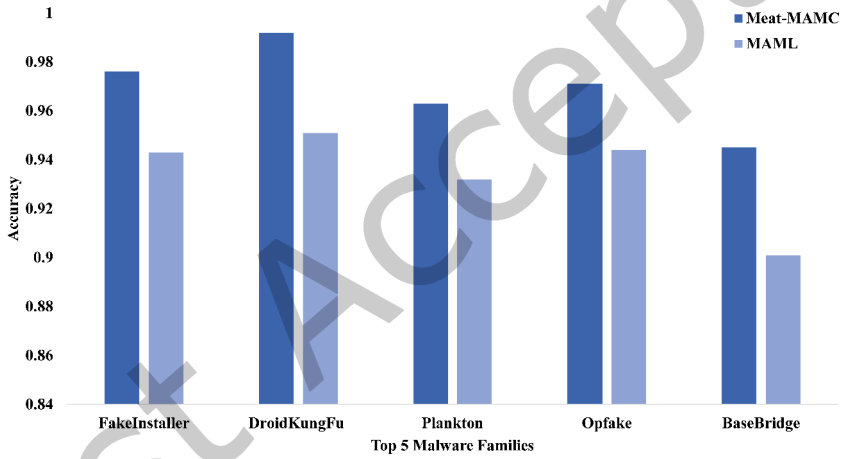

Fig. 4. Performance of Meta-MAMC and MAML in the top five families (top five by number) in the Drebin dataset.

Figure 4 and Figure 5 depict the performance comparison between the original MAML algorithm (shown in light blue) and our proposed Meta-MAMC algorithm (shown in dark blue) for the top five malware families (top five by number). It is evident from the figure that our improved Meta-MAMC algorithm outperforms the primitive MAML algorithm, particularly in the context of classifying the Android malware family. This is primarily attributed to our enhanced multi-class categorization of the Android malware family and our ability to effectively handle both few-sample and zero-sample scenarios. Our novel sampling strategy also plays a crucial role in achieving superior performance by balancing diverse families and samples. In addition to its incorporation of two sampling strategies, namely application-based sampling strategy, and family-based sampling strategy, Meta-MAMC outperforms the MAML algorithm which uses a native task sampling strategy. By incorporating a new sampling strategy, Meta-MAMC can consider both few-sample families and zero-sample families, thereby improving its generalization capabilities.

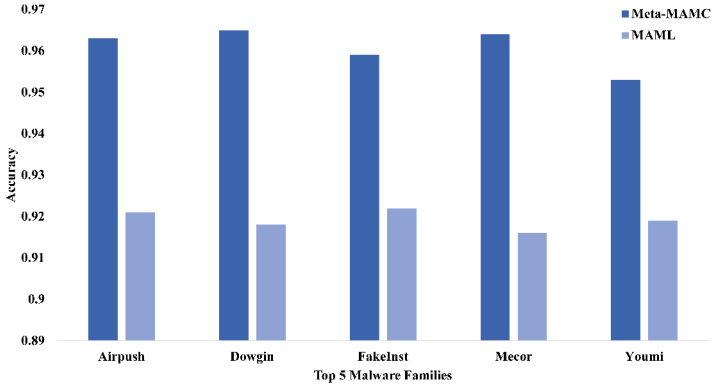

Fig. 5. Performance of Meta-MAMC and MAML on top five families (top five by number) in AMD dataset.

Previously, we validated the effectiveness of Meta-MAMC on multiple malware families. Now, we aim to conduct a comparison with several state-of-the-art Android malware classification schemes further to validate the effectiveness of Meta-MAMC. We have selected four baseline models that can classify malware families: Drebin [4], MaMaDroid [46], N-opcode [32], and EC2 [10].

Table 5. Meta-MAMC Versus State-of-the-Art Methods on Drebin. We show the F-scores of ten random families in the Drebin dataset.

Family	Drebin		MaMaDroid		N-opcode		EC2		**Meta-MAMC**	
	ACC	F-Score	ACC	F-Score	ACC	F-Score	ACC	F-Score	ACC	F-Score
FakeInstaller	0.786	0.773	0.874	0.875	0.864	0.875	0.827	0.819	**0.976**	**0.966**
DroidKungFu	0.769	0.781	0.861	0.849	0.852	0.867	0.843	0.826	**0.992**	**0.979**
Plankton	0.753	0.734	0.844	0.853	0.835	0.854	0.798	0.785	**0.963**	**0.961**
Opfake	0.765	0.769	0.851	0.836	0.847	0.864	0.811	0.805	**0.971**	**0.966**
BaseBridge	0.725	0.732	0.832	0.824	0.816	0.827	0.775	0.792	**0.945**	**0.943**
Imlog	0.700	0.695	0.813	0.806	0.800	0.780	0.787	0.769	**0.966**	**0.983**
Jifake	0.4	0.571	0.577	0.732	0.242	0.391	0.700	0.701	**0.909**	**0.954**
Zitmo	0.231	0.377	0.546	0.706	0.143	0.252	0.417	0.589	**0.700**	**0.824**
Stiniter	0.167	0.154	0.5	0.429	0.166	0.153	0.5	0.356	**0.833**	**0.910**
SmsSpy	0	0	0	0	0	0	0	0	**1**	**1**

The superiority of the proposed Meta-MAMC scheme over other methods is demonstrated in Tables 5 and 6. A total of ten families, comprising both large and small families, were randomly selected for evaluation. The results indicate that Meta-MAMC outperforms state-of-the-art methods on both the Drebin dataset and the AMD dataset. Significantly, our scheme exhibits exceptional performance in classifying small-scale household samples. As evidenced in Table 5, Meta-MAMC achieves a remarkable 100% detection rate for the SmsSpy family, surpassing the detection capabilities of other schemes. Similarly, for the Opfake family presented in Table 6, while other schemes struggle to surpass a 60% detection rate, Meta-MAMC achieves an impressive rate of 87.5%. These results can be attributed to the utilization of meta-knowledge encoded within the dataset, allowing Meta-MAMC to surpass the limitations of solely relying on the presence of features.

Table 6. Meta-MAMC Versus State-of-the-Art Methods on AMD. We show the F-Score of the ten random families in the AMD dataset.

Family	Drebin		MaMaDroid		N-opcode		EC2		**Meta-MAMC**	
	ACC	F-Score	ACC	F-Score	ACC	F-Score	ACC	F-Score	ACC	F-Score
Airpush	0.769	0.758	0.851	0.838	0.781	0.775	0.836	0.841	**0.960**	**0.960**
Dowgin	0.784	0.776	0.861	0.847	0.783	0.792	0.840	0.837	**0.965**	**0.959**
Youmi	0.751	0.737	0.836	0.824	0.765	0.773	0.827	0.818	**0.953**	**0.951**
Kuguo	0.803	0.794	0.786	0.791	0.738	0.751	0.875	0.886	**0.972**	**0.966**
BankBot	0.763	0.745	0.802	0.807	0.814	0.807	0.824	0.836	**0.976**	**0.973**
Mseg	0.719	0.774	0.843	0.826	0.634	0.647	0.917	0.913	**0.987**	**0.985**
Koler	0.791	0.787	0.836	0.848	0.708	0.703	0.786	0.785	**0.959**	**0.979**
Cova	0.364	0.286	0.818	0.789	0.456	0.462	0.727	0.706	**0.909**	**0.952**
Opfake	0.286	0.316	0.571	0.522	0.143	0.182	0.571	0.571	**0.875**	**0.933**
Ogel	0	0	0.550	0.190	0	0	0	0	**1**	**1**

In addition, we introduce two innovative sampling strategies, namely application-based sampling strategy and family-based sampling strategy, to facilitate meta-learning training. These strategies ensure a balanced representation of both few-shot and many-shot families during training, thus providing a realistic and reliable training scenario. By considering the unique characteristics of different families, our approach achieves improved performance compared to existing schemes. The distinction between traditional machine learning and meta-learning is significant in the context of malware classification. Traditional methods require extensive data to effectively train models, especially when encountering new or rare samples. Meta-learning, however, trains across various tasks, allowing models to swiftly adapt to new scenarios, even with limited data. Moreover, we apply data augmentation techniques [74] to solve the limited samples, particularly aimed at addressing the challenges posed by datasets with limited samples. These techniques are vital for enhancing the robustness and generalizability of machine learning models through the artificial expansion of the dataset, which can improve the performance of the sampling strategies. This adaptability is particularly beneficial for identifying novel and unknown malware types, such as zero-day attacks. Meta-learning's ability to discern subtle differences and unique patterns makes it more effective in distinguishing between new and existing malware. Our method employs meta-learning not bound to any specific model. It's proven superior to traditional methods in scenarios with limited data. We've developed application-level and family-based sampling strategies for mobile malware classification. If these strategies are applied in conjunction with meta-learning in other methods, it could enhance their performance. Our research demonstrates, through Figures 4 and 5, that combining meta-learning with our sampling strategies improves performance over traditional methods. We've compared the effectiveness of our sampling strategies against conventional methods, revealing significant improvements in malware detection.

Table 7 showcases our unique sampling strategy applied to the SVM method. Contrasting traditional SVM that employs random sampling without accounting for the distribution of malware family samples, our approach, with a mere 0.4 sampling ratio, outperformed the standard SVM on validation tests. Moreover, we extended our methodology to the XGBoost model, where a 0.35 sampling ratio sufficed to surpass XGBoost's results with random sampling, as detailed in Table 8. To further underscore our strategy's effectiveness, we compare it against diverse XGBoost [13] sampling strategies (random, over and under sampling, editing distance-based) in Table 9. Moreover, XGBoost is a type of boosting algorithm, which integrates many weak classifiers to form a strong classifier. Since XGBoost is a tree boosting model, it integrates many tree models to create a robust classifier. Afterwards, we employ the SVM classifier [79], renowned for its binary classification capability, which not only

Table 7. SVM for Our (SS: Sampling Strategies) and (B: Benign, S: Malware) and Comparison with Random Sampling

SS (ratio)	B Class (Benign)			S Class (Malware)			Macro Avg			Accuracy
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	
0.05	0.95	0.9896	0.9694	0.9811	0.9123	0.9455	0.9656	0.9509	0.9574	0.9608
0.10	0.9333	**0.9949**	0.9631	**0.9896**	0.8716	0.9268	0.9615	0.9332	0.9449	0.9510
0.15	0.9404	0.9930	0.9660	0.9873	0.8960	0.9394	0.9638	0.9445	0.9527	0.9564
0.20	0.9711	0.9762	0.9736	0.9614	0.9532	0.9573	0.9662	0.9647	0.9654	0.9674
0.25	0.9735	0.9917	0.9825	0.9855	0.9544	0.9697	0.9795	0.9730	0.9761	0.9778
0.30	0.9492	0.9912	0.9697	0.9848	0.9153	0.9488	0.9670	0.9532	0.9592	0.9619
0.35	0.9713	0.9908	0.9810	0.9853	0.9550	0.9699	0.9783	0.9729	0.9754	0.9767
0.40	**0.9795**	**0.9961**	**0.9877**	**0.9933**	**0.9651**	**0.9790**	**0.9864**	**0.9806**	**0.9834**	**0.9845**
0.45	0.9729	**0.9942**	0.9835	**0.9898**	0.9530	0.9711	0.9814	0.9736	0.9773	0.9790
0.50	0.9719	0.9883	0.9800	0.9807	0.9540	0.9672	0.9763	0.9712	0.9736	0.9752
SVM	0.9768	0.9930	0.9848	0.9883	0.9616	0.9748	0.9825	0.9773	0.9798	0.9811

Table 8. XGBoost for Our (SS: Sampling Strategies) and (B: Benign, S: Malware) and Comparison with Random Sampling

SS (ratio)	B Class (Benign)			S Class (Malware)			Macro Avg			Accuracy
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score	
0.05	0.9635	0.9764	0.9699	0.9607	0.9397	0.9501	0.9621	0.9581	0.9600	0.9624
0.10	0.9724	0.9850	0.9787	0.9750	0.9546	0.9647	0.9737	0.9698	0.9717	0.9734
0.15	0.9781	0.9823	0.9802	0.9710	0.9642	0.9676	0.9745	0.9732	0.9739	0.9754
0.20	**0.9808**	0.9893	0.9850	0.9823	0.9686	0.9754	0.9816	0.9789	0.9802	0.9814
0.25	0.9830	0.9914	0.9872	0.9858	0.9721	0.9789	0.9844	0.9817	0.9830	0.9840
0.30	0.9829	0.9871	0.9850	0.9789	0.9721	0.9755	0.9809	0.9796	0.9802	0.9814
0.35	**0.9883**	**0.9936**	**0.9909**	**0.9894**	**0.9808**	**0.9851**	**0.9888**	**0.9872**	**0.9880**	**0.9887**
0.40	**0.9835**	0.9919	0.9877	0.9867	0.9729	0.9798	0.9851	0.9824	0.9837	0.9847
0.45	**0.9941**	0.9925	**0.9933**	**0.9878**	**0.9904**	**0.9891**	**0.9909**	**0.9914**	**0.9912**	**0.9917**
0.50	**0.9925**	**0.9952**	**0.9938**	**0.9921**	**0.9878**	**0.9899**	**0.9923**	**0.9915**	**0.9919**	**0.9924**
XGBoost	0.9872	0.9925	0.9898	0.9877	0.9790	0.9833	0.9874	0.9858	0.9866	0.9874

distinguishes between categories but also optimizes the margin between them, enabling a more precise and dependable classification for in-depth analysis. Additionally, the Edited Nearest Neighbors [2] (ENN) sampling technique is an under-sampling method based on the nearest neighbor rule. Its basic idea is to remove samples that are misclassified by their nearest neighbors, thus reducing the number of majority class samples and balancing the dataset. Over-sampling and under-sampling [31] adjust the number of samples in each class in the dataset to address class imbalance. Meanwhile, we employ the Synthetic Minority Oversampling Technique (SMOTE) [45], a widely recognized oversampling method, to generate synthetic instances of minority class samples, thereby augmenting the dataset with an increased number of samples. We do not show results such as TextCNN because TextCNN is sensitive to data quality and structure, which may lead to unstable performance of various sampling techniques. In contrast, XGBoost and SVM show greater robustness to data changes. Furthermore, the design of Meta-MAMC may tend to overfit certain families of malicious families, whereas XGBoost and SVM show superior generalization capabilities on the dataset. Therefore, XGBoost and SVM perform better on some families. While our method showcased superior recall in malware detection, it slightly lagged in precision but was more effective in predicting potential malware compared to other sampling techniques. The random sampling strategy, in particular, performed poorly due to its neglect in addressing sample size disparities across categories. Our

method's balanced approach is crucial for capturing features from both minority and majority samples, enhancing the model's overall capability to identify malware accurately.

Table 9. Comparison of Sampling Strategies in XGBoost. M: Macro, RS: Random Sampling, ODS: Over and under Sampling, ENN: Edited Nearest Neighbors, OS: Our Sampling (fraction 0.5)

Method	Precision		Recall		F1-Score		M Avg Prc	M Avg Rcl	M Avg F1
	Benign	Malware	Benign	Malware	Benign	Malware			
RS	0.9802	0.9762	0.9855	0.9677	0.9829	0.9719	0.9782	0.9766	0.9774
ODS	0.9876	0.9914	0.9917	0.9871	0.9896	0.9893	0.9895	0.9894	0.9894
ENN	0.9878	**0.9981**	**0.9989**	0.9789	0.9933	0.9884	**0.9930**	0.9889	0.9909
OS (0.5)	**0.9925**	0.9921	0.9952	**0.9878**	**0.9938**	**0.9899**	0.9923	**0.9915**	**0.9919**

Answer to RQ 2: In comparison to the MAML algorithm, the proposed Meta-MAMC method exhibits superior performance. Furthermore, when compared to the current state-of-the-art methods, our approach outperforms them on both the Drebin dataset and the AMD dataset. These findings highlight the effectiveness of the Meta-MAMC method and its ability to achieve improved performance in the field of malware detection.

5.3 Analysis of Hyperparameter

In the previous section, we fully evaluate the classification ability of Meta-MAMC. However, a hyperparameter is set in our scheme to control the two sampling strategies. We delve deeper into how the choice of hyperparameters affects the performance of the Meta-MAMC model. Specifically, we examine the impact of the hyperparameter p on the distribution of meta-learning tasks and the resulting diversity, which in turn affects the generalization ability of the models.

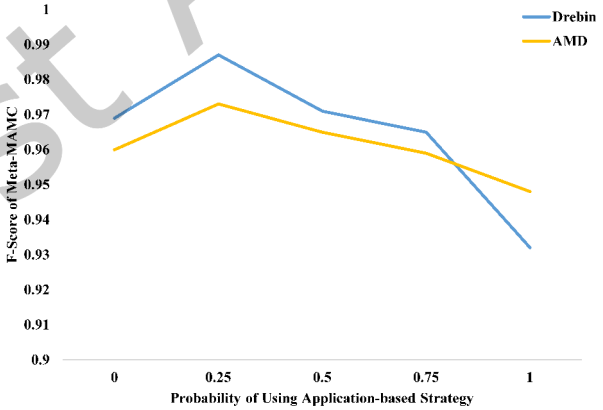

Fig. 6. Performance with different hyperparameter p.

The impact of the hyperparameter p on the performance of Meta-MAMC is shown in Figure 6. We construct the training tasks by setting the different p values. The hyperparameter p is used to control the participation ratio of

the sampling strategy. p is chosen from the set [0.00, 0.25, 0.50, 0.75, 1.00]. It is important to note that $p = 0.00$ represents the pure family-based task sampling strategy, while $p = 1.00$ represents the pure application-based strategy. The results show the value of p has a significant impact on the performance of Meta-MAMC. As can be seen from Figure 6, Meta-MAMC works better when $p = 0.25$. Generally, the performance first improves and then decreases as the value of p increases. Pure application-based sampling strategies are inferior because they do not consider the long-tailed distribution of label frequency in datasets and reduce the diversity of sampled tasks. Meanwhile, pure family-based sampling strategies were less effective as they disregarded samples from other families. Thus, we use a combination of both strategies simultaneously and regulate the parameter p to adapt to various situations.

6 EFFECTIVENESS AGAINST ANDROID NEW MALWARE

To answer RQ 3, we further validate Meta-MAMC's ability to resist malware evolution and zero-day malware.

Table 10. Performance of schemes against Android evolution

Method	Accuracy & F-Score									
	Scenario A		Scenario B		Scenario C		Scenario D		Scenario E	
EFIMDetector	0.889	0.890	0.886	0.902	0.905	0.911	0.884	0.879	0.860	0.856
MaMaDroid	0.823	0.818	0.864	0.883	0.854	0.863	0.875	0.867	0.854	0.861
Drebin	0.838	0.846	0.913	0.921	0.903	0.918	0.646	0.668	0.650	0.655
SEDMDroid	0.854	0.861	0.863	0.871	0.868	0.890	0.877	0.858	0.843	0.838
MMN	0.867	0.868	0.866	0.875	0.873	0.889	0.874	0.872	0.863	0.857
Meta-MAMC	**0.945**	**0.939**	**0.933**	**0.938**	**0.946**	**0.952**	**0.951**	**0.960**	**0.946**	**0.951**

6.1 Performance of Robustness against Android Evolution

As the Android operating system and its ecosystem continue to evolve, the corresponding Android malware is also constantly evolving and becoming more sophisticated. The Android malware family is expanding, and new malware variants are emerging every day. Based on this fact, the classification of Android malware should not only focus on the existing datasets but also consider the continuous evolution of Android malware and new unknown malware that may arise in the future. Therefore, we simulate five evaluation scenarios to verify the ability of Meta-MAMC to resist malware evolution. The five scenarios, namely A, B, C, D, and E, are created, and experiments are conducted on each of them. The F-Score and accuracy of EFIMDetector [37], MaMaDroid [46], Drebin [4], SEDMDroid [86], MMN [33], and Meta-MAMC are presented in Table 10. In scenario A, each scheme is trained using the 2012 and 2013 datasets, and the samples from 2014 to 2022 are classified. For scenario B, samples randomly selected from datasets before 2015 are used as the training data, and the samples from 2015 to 2022 are tested. Similarly, the training samples in scenario C are randomly selected from datasets before 2017, and the testing samples are from 2017 to 2022. Scenario D included randomly chosen training samples from datasets before 2019, and classification is carried out on the samples from 2019 to 2022. Scenario E randomly selects training samples from the dataset before 2021 to classify samples from 2021-2022. To conduct these five experiments, we crawl malicious samples from 2012 to 2019 in VirusShare [23] and collect benign samples from AndroZoo [3]. Malware datasets from 2020 to 2022 are collected from GitHub [33, 60]. We ensure that the malicious families in the testing and training sets do not overlap.

Table 10 presents the F-Score and accuracy values of Meta-MAMC, MaMaDroid, Drebin, SEDMDroid, and MMN in classifying zero-day malware. Overall, Meta-MAMC performs slightly better than the others in the five scenarios. This is mainly because of the meta-knowledge of the dataset, which is more resilient to the zero-day samples

Table 11. Performance of schemes against Zero-day Malware

Method	EFIMDetector	Drebin	MaMaDroid	N-opcode	EC2	SEDMDroid	MMN	Meta-MAMC
Number	23	6	34	11	28	22	26	65
Detection Rate	0.323	0.084	0.478	0.154	0.394	0.309	0.366	0.915

and Android evolution. Compared with the four state-of-the-art schemes, Meta-MAMC can obtain a good effect in classifying newer samples using an old dataset.

6.2 Performance of Zero-day Evaluation

We obtain a new set of malware samples to evaluate the effectiveness of our scheme. We collect new malware from January 1 to March 22, 2023, and have collected a total of 71 so far. These malware are not included in the dataset we are currently using. These samples are gathered from the MalwareBazaar website. Furthermore, to verify the accuracy of our experiment, we submit these 71 samples to multiple detection tools, such as VirusTotal [55]. These tools also identify these applications as malicious. Therefore, we add these 71 samples to the testing set for classification. Table 11 showcases the extensive detection results attained by each scheme, shedding light on the number of malicious samples identified and the corresponding detection rates achieved by each respective method. Notably, Meta-MAMC exhibits exceptional accuracy, successfully classifying 65 of these samples as malicious. In contrast, alternative state-of-the-art methods such as EFIMDetector [37], MaMaDroid [46], Drebin [4], SEDMDroid [86], EC2 [10], N-opcode [32], and MMN [33] fall short, exhibiting detection rates below 50%. After analyzing the remaining 6 samples, we find that the feature sets extracted by these malware are empty. This is because the feature selection algorithm we designed is relatively strict, resulting in an insufficient number of features extracted from these samples to support the classifier classification (only two or three features are retained). To address this issue, it is suggested to relax the degree of feature selection during actual deployment. Meta-MAMC controls the two sampling strategies through the hyperparameter p, so that it can learn the knowledge of few-sample families and zero-sample families to effectively deal with zero-day malware.

Answer to RQ 3: Meta-MAMC can help alleviate the issues caused by Android version iterations and technology updates. Additionally, by combining new sampling algorithms and zero-sample scenarios based on meta-learning, Meta-MAMC can effectively classify zero-day malware.

7 DISCUSSION

7.1 Analysis of Meta-MAMC

Meta-MAMC improves the original MAML algorithm, making it more suitable for multi-family classification of Android malware. One challenge we address is dealing with family sample imbalance and malicious family identities. We utilize the dual sampling strategies to construct the training tasks and regulate the proportion of the two sampling strategies by setting the hyperparameter p. We incorporate both sampling strategies because using only one strategy can result in the model ignoring certain families during training. For instance, when only the application-based sampling strategy is adopted, it becomes challenging to select few-sample families, leading to inadequate learning of these families by the model. Thus, we use both strategies simultaneously. It is the simultaneous adoption of two sampling strategies that enable Meta-MAMC to take into account few-sample families, zero-sample families, and other families (non-few-sample families and non-zero-sample families). In this way, Meta-MAMC can better learn the characteristics of few-sample families and improve the generalization of the final meta-classifier, thereby improving the ability to classify zero-day malware.

In our research, we tackle another important challenge of enhancing the resilience of malware detection systems against zero-day malware while considering the dynamic nature of malware evolution. Consequently, sustainability emerges as a highly relevant performance indicator, reflecting the ability of a classifier to maintain its effectiveness over time. A pertinent problem concept associated with this challenge is concept drift, which denotes the occurrence of changes in the underlying data distribution.

To address concept drift and its impact on unknown software detection, previous studies have proposed various techniques. For instance, *Barbero et al.* [30] introduced a fully parametric statistical framework that evaluates classifier decisions to identify instances of concept drift. Similarly, *Xu et al.* [76] presented DroidEvolver, an approach that detects drift and utilizes online techniques to adaptively update the detection model. These approaches leverage the detection of drift to effectively identify unknown software instances.

However, it is important to note that our focus is not on recognizing drift itself. Instead, we improve the meta-learning algorithm by incorporating labeled data representing unknown malware families during the training process. This enhancement aims to enhance the classification performance of the model on unknown families and, consequently, reinforce its capability to accurately identify new instances of malware. By addressing the challenge of zero-day malware and leveraging the power of meta-learning, our proposed approach, named `Meta-MAMC`, effectively leverages the meta-knowledge extracted from limited malware samples. This allows us to mitigate the issues associated with imbalanced data distribution and improve the classification effectiveness for unknown malware families. As a result, our method demonstrates enhanced resilience against zero-day malware and improves the overall effectiveness of malware identification.

Moreover, our sampling method is particularly proposed for malware detection in Android software. Strategies like Stratified Sampling [54], SMOTE [11], and Over/Under-sampling techniques [50] are integrated to address the challenges of small sample sizes prevalent in this domain. Our approach, which uniquely merges application-specific and family-related features, harmonized through a "P value" strategy, ensures a balanced consideration of both domain-specific and common family features. This dual focus represents an advantage of our method, as it emphasizes application layer features while also capturing broader familial characteristics. Additionally, in the context of malware detection, the primary challenge is dealing with small sample sizes. Our goal is to retain pertinent features from various categories while extracting features from limited samples, thereby maintaining a comprehensive dataset for effective malware identification.

7.2 Threats to Validity

Internal threats to validity. Like other static solutions, the features we use inevitably encounter code confusion. This makes it difficult for us to guarantee the authenticity of the extracted features, which in turn affects the training and classification of subsequent models. To overcome this issue, we use some unpacking systems [80, 83] to recover the actual files, and then apply static analysis to extract features. In contrast, `Meta-MAMC` is a classification solution based on meta-learning, which can learn unique metaknowledge from the dataset to alleviate this problem.

External threats to validity. Authoritative datasets such as Drebin and AMD do not include all malware families and are subject to a long-tailed distribution of family sample sizes, which can affect model training due to class imbalance. To mitigate this issue, for example, we have collected unknown malware from MalwareBazaar and submitted them to VirusTotal to provide a more comprehensive analysis that can assign family tags to samples.

8 RELATED WORK

Static analysis and dynamic analysis: Android malware classification techniques can be categorized into two distinct forms: static analysis and dynamic analysis. Static analysis involves decompiling the APK file to obtain the source code and metadata files, followed by extraction of significant features for malware classification and

analysis such as the application programming interface (API) [41, 65], permissions [1, 16, 34, 84], intent [64], function call graph [18, 24, 40], control flow [43], and grayscale [7]. Dynamic analysis [6, 48, 58, 63, 71, 85] involves the extraction of features by monitoring the application's execution in real time. Dynamic monitoring is time-consuming and labor-intensive, and there is a risk of losing genuine malicious code segments. However, our method is based on static analysis, extracting eight categories of features from APK files for classification.

ML-based: Machine learning-based malware classification relies on classifiers to distinguish malicious from benign applications and to classify malware families. Many relevant studies focus on common classifiers, such as SVM [40, 57, 68], k-nearest neighbors (KNN) [84], and decision tree (DT) [52], RF [34, 87] to classify malware. In addition, even deep learning methods [5, 86] and meta-learning methods [36, 89] are directly used for classification. *Cai et. al.* [9] develop DroidSpan, a novel classification system based on a new behavioral profile for Android apps that captures sensitive access distribution from lightweight profiling. But DroidSpan is not as comparable because it is a dynamic technique (using features extracted from run-time app execution traces). CDGDroid [77] is proposed to classify and characterize Android malware families automatically [78]. GRAMMC, a novel graph signature-based malware classification mechanism is proposed. The proposed graph signature uses sensitive API calls to capture the flow of control which helps to find a caller–callee relationship between the sensitive API and the nodes incident on them [75]. In addition, another interesting work [70] explores the key features that can classify and describe the behaviors of Android malware families to enable fingerprinting the malware families with these features. Unlike these approaches, our `Meta-MAMC` scheme does not simply classify malware families based on extracted features. Instead, it utilizes meta-learning to acquire meta-knowledge from the data, which can be used to improve the accuracy of the classification task.

9 CONCLUSION AND FUTURE WORK

In this paper, we propose a novel meta-learning-based approach for multi-family Android malware classification, called `Meta-MAMC`. We introduce two new sampling strategies, namely the application-based and family-based sampling strategies. The former can effectively mitigate the issue of imbalanced family sample distributions, while the latter addresses the problem of incomplete family datasets by leveraging zero-sample learning scenarios to aid the model's understanding of novel families and new malware samples. We control the two sampling algorithms by adjusting the hyperparameters, specifically selecting the family-based sampling and application-based sampling methods for task sampling purposes. Finally, we conduct several experiments and compare our approach to state-of-the-art solutions, showing that `Meta-MAMC` achieves better results.

In the field of Few-Shot Learning (FSL) applied to malware classification, there are several existing challenges. One primary issue is the difficulty in distinguishing malware samples with similar features, especially when these samples belong to different families. This similarity can confound classifiers. Furthermore, there is a notable gap in testing scenarios, particularly those involving test sets that include basic malware families, which has not been adequately explored. Addressing this gap is crucial for advancing the field. A key area for future research is dataset scale augmentation. Currently, datasets in this domain often contain only a few samples per malware family, which significantly hampers the model's ability to generalize. Expanding these datasets will be vital in overcoming this limitation. Another promising direction involves employing more advanced embedding architectures. For example, if the dataset scale is sufficiently large, using robust structures like BERT and its variants could be beneficial. These models are adept at extracting contextual information and providing improved sequence embeddings, which can enhance classification accuracy. Moreover, existing few-shot learning methods often show limited transferability when the task domain changes, leading to subpar performance in new domains. Future research should focus on improving the generalization capacity of these models. This includes adapting models trained in other fields to effectively address few-shot malware detection challenges. It is well known that detecting unknown malware in real-time is essential for real-world applications. A malware detection system

must be capable of identifying not only known malware but also unknown variants as they emerge. Future research should therefore explore the implementation of incremental few-shot malware detection methods to meet real-life requirements. This approach would enable systems to continually adapt and recognize new malware types effectively.

DATA AVAILABILITY

The data and code underlying this article are available at the following repository[3]

ACKNOWLEDGMENTS

The authors would like to appreciate all anonymous reviewers for their insightful comments and constructive suggestions to polish this paper in high quality.

REFERENCES

[1] Moutaz Alazab, Mamoun Alazab, Andrii Shalaginov, Abdelwadood Mesleh, and Albara Awajan. 2020. Intelligent mobile malware detection using permission requests and API calls. *Future Generation Computer Systems* 107 (2020), 509–521.
[2] Roberto Alejo, José Martínez Sotoca, Rosa Maria Valdovinos, and P Toribio. 2010. Edited nearest neighbor rule for improving neural networks classifications. In *Advances in Neural Networks-ISNN 2010: 7th International Symposium on Neural Networks, ISNN 2010, Shanghai, China, June 6-9, 2010, Proceedings, Part I 7*. Springer, 303–310.
[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting Millions of Android Apps for the Research Community. In *International Conference on Mining Software Repositories*. 468––471.
[4] Daniel Arp, M. Spreitzenbarth, M Hübner, H. Gascon, and K. Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Network & Distributed System Security Symposium*. 1–15.
[5] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. PScout: Analyzing the Android Permission Specification. In *ACM Conference on Computer and Communications Security*. 217–228.
[6] Chanwoo Bae and Seungwon Shin. 2016. A collaborative approach on host and network level android malware detection. *Security and Communication Networks* 9 (2016), 5639–5650.
[7] K Bakour and H. M. Nver. 2021. VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques. *Neural Computing and Applications* 33 (2021), 3133–3153.
[8] Kathrin Beckert-Plewka, Hauke Gierow, Vera Haake, and Stefan Karpenstein. 2020. *G DATA Mobile Malware Report: Harmful Android apps every eight seconds.* https://www.gdatasoftware.com/news/1970/01/-36401-g-data-mobile-malware-report-harmful-android-apps-every-eight-seconds
[9] Haipeng Cai. 2020. Assessing and Improving Malware Detection Sustainability through App Evolution Studies. *ACM Transactions on Software Engineering and Methodology* 29, 2 (2020).
[10] Tanmoy Chakraborty, Fabio Pierazzi, and V. S. Subrahmanian. 2020. EC2: Ensemble Clustering and Classification for Predicting Android Malware Families. *IEEE Transactions on Dependable and Secure Computing* 17, 2 (2020), 262–277.
[11] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.
[12] Victor Chebyshev. 2021. *Mobile malware evolution 2020.* Kaspersky. https://securelist.com/mobile-malware-evolution-2020/101029/
[13] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.
[14] Yahui Chen. 2015. *Convolutional neural network for sentence classification.* Master's thesis. University of Waterloo.
[15] Anthony Desnos. 2023. *Androguard, a full python tool to play with Android files.* https://github.com/androguard/androguard
[16] V. P. Dharmalingam and V. Palanisamy. 2020. A novel permission ranking system for android malware detection—the permission grader. *Journal of Ambient Intelligence and Humanized Computing* 12 (2020), 5071–5081.
[17] Karim O. Elish, Mahmoud O. Elish, and Hussain M. J. Almohri. 2022. Lightweight, Effective Detection and Characterization of Mobile Malware Families. *IEEE Trans. Comput.* 71, 11 (2022), 2982–2995.
[18] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu. 2018. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis. *IEEE Transactions on Information Forensics and Security* 13 (2018), 1890–1905.

[3]https://figshare.com/articles/dataset/meta_learning_for_malware_classification/25036160

[19] Ruitao Feng, Sen Chen, Xiaofei Xie, Lei Ma, Guozhu Meng, Yang Liu, and Shang-Wei Lin. 2019. MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform. In *2019 24th International Conference on Engineering of Complex Computer Systems*. 61–70.
[20] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantics-Based Detection of Android Malware through Static Analysis. In *Proceedings of ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 576–587.
[21] Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. 2017. Automated Synthesis of Semantic Malware Signatures using Maximum Satisfiability. 1–16.
[22] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. 1126–1135.
[23] Corvus Forensics. 2023. *VirusShare*. https://virusshare.com
[24] H. Gao, S. Cheng, and W. Zhang. 2021. GDroid: Android Malware Detection and Classification with Graph Convolutional Network. *Computers & Security* 20 (2021), 102264–102278.
[25] Seymour Geisser. 1975. The Predictive Sample Reuse Method with Applications. *J. Amer. Statist. Assoc.* 70, 350 (1975), 320–328.
[26] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. RiskRanker: Scalable and Accurate Zero-Day Android Malware Detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. 281–294.
[27] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. 2022. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44 (2022), 5149–5169.
[28] Taewon Jeong and Heeyoung Kim. 2020. OOD-MAML: Meta-Learning for Few-Shot out-of-Distribution Detection and Classification. In *Proceedings of the International Conference on Neural Information Processing Systems*.
[29] Jianguo Jiang, Song Li, Min Yu, Gang Li, Chao Liu, Kai Chen, Hui Liu, and Weiqing Huang. 2019. Android Malware Family Classification Based on Sensitive Opcode Sequence. In *2019 IEEE Symposium on Computers and Communications*. 1–7.
[30] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *USENIX Security Symposium*. 625–642.
[31] Nutthaporn Junsomboon and Tanasanee Phienthrakul. 2017. Combining over-sampling and under-sampling techniques for imbalance dataset. In *Proceedings of the 9th international conference on machine learning and computing*. 243–247.
[32] Boojoong Kang, Suleiman Y. Yerima, Kieran Mclaughlin, and Sakir Sezer. 2016. N-opcode analysis for android malware classification and categorization. In *International Conference On Cyber Security And Protection Of Digital Services*. 1–7.
[33] Taeguen Kim, Boojoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. 2019. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security* 14 (2019), 773–788.
[34] Cai L., Li Yao, and Xiong Zhi. 2021. JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Computers & Security* 100 (2021), 102086–102100.
[35] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. 2021. A Framework for Enhancing Deep Neural Networks Against Adversarial Malware. *IEEE Transactions on Network Science and Engineering* 8 (2021), 736–750.
[36] Wenhao Li, Huaifeng Bao, Xiao-Yu Zhang, and Lin Li. 2022. AMDetector: Detecting Large-Scale and Novel Android Malware Traffic with Meta-learning. In *Computational Science*, Derek Groen, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot (Eds.). 387–401.
[37] Yao Li, Zhi Xiong, Tao Zhang, Qinkun Zhang, Ming Fan, and Lei Xue. 2022. Ensemble Framework Combining Family Information for Android Malware Detection. *Comput. J.* (2022).
[38] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. 2020. Software Vulnerability Detection Using Deep Neural Networks: A Survey. *Proc. IEEE* 108, 10 (2020), 1825–1848.
[39] Hao Liu, Richard Socher, and Caiming Xiong. 2019. Taming MAML: Efficient unbiased meta-reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. 4061–4071.
[40] P. Liu, W. Wang, X. Luo, H. Wang, and C. Liu. 2021. NSDroid: efficient multi-classification of android malware using neighborhood signature in local function call graphs. *International Journal of Information Security* 20 (2021), 59–71.
[41] Zhen Liu, Ruoyu Wang, Nathalie Japkowicz, Deyu Tang, Wenbin Zhang, and Jie Zhao. 2021. Research on unsupervised feature learning for Android malware detection based on Restricted Boltzmann Machines. *Future Generation Computer Systems* 120 (2021), 91–108.
[42] Duoyuan Ma, Yude Bai, Zhenchang Xing, Lintan Sun, and Xiaohong Li. 2020. A Knowledge Graph-based Sensitive Feature Selection for Android Malware Classification. In *Asia-Pacific Software Engineering Conference*. 188–197.
[43] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma. 2019. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. *IEEE Access* 7 (2019), 21235–21245.
[44] Samaneh Mahdavifar, Dima Alhadidi, and Ali. A. Ghorbani. 2022. Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder. *Journal of Network and Systems Management* 30 (2022), 22–56.
[45] Hadi Mansourifar and Weidong Shi. 2020. Deep synthetic minority over-sampling technique. *arXiv preprint arXiv:2003.09788* (2020).
[46] E. Mariconti, L. Onwuzurike, P. Andriotis, E Cristofaro, G. Ross, and G. Stringhini. 2019. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. *ACM Transactions on Privacy and Security* 22 (2019), 1–34.

[47] Alejandro Martín, Víctor Rodríguez-Fernández, and David Camacho. 2018. CANDYMAN: Classifying Android Malware Families by Modelling Dynamic Traces with Markov Chains. *Eng. Appl. Artif. Intell.* 74 (2018), 121–133.
[48] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, and R. Baldoni. 2020. AndroDFA: Android Malware Classification Based on Resource Consumption. *Information (Switzerland)* 11 (2020), 326–346.
[49] Guozhu Meng, Yinxing Xue, Chandramohan Mahinthan, Annamalai Narayanan, Yang Liu, Jie Zhang, and Tieming Chen. 2016. Mystique: Evolving Android Malware for Auditing Anti-Malware Tools. In *Proceedings of ACM on Asia Conference on Computer and Communications Security*. 365–376.
[50] Roweida Mohammed, Jumanah Rawashdeh, and Malak Abdullah. 2020. Machine learning with oversampling and undersampling techniques: overview study and experimental results. In *2020 11th international conference on information and communication systems (ICICS)*. IEEE, 243–248.
[51] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. 2018. A Multi-View Context-Aware Approach to Android Malware Detection and Malicious Code Localization. *Empirical Software Engineering* 23 (2018), 1222–1274.
[52] M. Nisa, J. H. Shah, S. Kanwal, M. Raza, and T. Blaauskas. 2020. Hybrid Malware Classification Method Using Segmentation-Based Fractal Texture Analysis and Deep Convolution Neural Network Features. *Applied Sciences* 10, 14 (2020), 4966–4989.
[53] S. O'Dea. 2020. *Android - Statistics & Facts*. Statista. https://www.statista.com/topics/876/android/
[54] Van L Parsons. 2014. Stratified sampling. *Wiley StatsRef: Statistics Reference Online* (2014), 1–11.
[55] Antivirus products. 2023. *VIRUSTOTAL*. https://www.virustotal.com
[56] Kiran Radhakrishnan, Rajeev R Menon, and Hiran V Nath. 2019. A survey of zero-day malware attacks and its detection methodology. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*. 533–539.
[57] A. Salah, E. Shalabi, and W. Khedr. 2020. A Lightweight Android Malware Classifier Using Novel Feature Selection Methods. *Symmetry* 12 (2020), 858–874.
[58] A. Saracino, D Sgandurra, G. Dini, and F. Martinelli. 2018. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Transactions on Dependable & Secure Computing* 15 (2018), 83–97.
[59] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling, Vol. 9854. 230–253.
[60] sk3ptre. 2023. *Malware datasets from 2021 to 2022*. https://github.com/sk3ptre
[61] G. Suarez-Tangil and G. Stringhini. 2022. Eight Years of Rider Measurement in the Android Malware Ecosystem. *IEEE Transactions on Dependable and Secure Computing* 19 (2022), 107–118.
[62] Guillermo Suarez-Tangil, Juan E. Tapiador, Pedro Peris-Lopez, and Jorge Blasco. 2014. Dendroid: A Text Mining Approach to Analyzing and Classifying Code Structures in Android Malware Families. *Expert System Application* 41 (2014), 1104–1117.
[63] R. Surendran, T. Thomas, and S. Emmanuel. 2020. On Existence of Common Malicious System Call Codes in Android Malware Families. *IEEE Transactions on Reliability* 70 (2020), 248–260.
[64] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti. 2020. On Defending Against Label Flipping Attacks on Malware Detection Systems. *Neural Computing and Applications* 32 (2020), 14781–14800.
[65] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu. 2018. MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. *IEEE Transactions on Reliability* 67 (2018), 355–369.
[66] Sebastian Thrun and Lorien Pratt. 1998. *Learning to Learn: Introduction and Overview*. 3–17.
[67] Connor Tumbleson and RyszardWiśniewski. 2023. *APKtool*. https://ibotpeaches.github.io/Apktool/
[68] P. D. Varna and P. Visalakshi. 2020. Detecting android malware using an improved filter based technique in embedded software. *Microprocessors and Microsystems* 76 (2020), 103115–103127.
[69] Nicolas Viennot, Edward Garcia, and Jason Nieh. 2014. A measurement study of Google Play. In *ACM international conference on Measurement and modeling of computer systems*. 221–233.
[70] Devyani Vij, Vivek Balachandran, Tony Thomas, and Roopak Surendran. 2020. GRAMAC: A Graph Based Android Malware Classification Mechanism. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*. 156–158.
[71] A Vp, B Az, and C Mc. 2019. A machine learning based approach to detect malicious android apps using discriminant system calls. *Future Generation Computer Systems* 94 (2019), 333–350.
[72] Peng Wang, Zhijie Tang, and Junfeng Wang. 2021. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Computers & Security* 106 (2021), 102273.
[73] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep Ground Truth Analysis of Current Android Malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. 252–276.
[74] Jason Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196* (2019).
[75] Nannan Xie, Xing Wang, Wei Wang, and Jiqiang Liu. 2018. Fingerprinting Android malware families. *Frontiers of Computer Science* 13 (2018), 637–646.

[76] Ke Xu, Yingjiu Li, Robert Deng, Kai Chen, and Jiayun Xu. 2019. DroidEvolver: Self-Evolving Android Malware Detection System. In *IEEE European Symposium on Security and Privacy*. 47–62.
[77] Zhiwu Xu, Kerong Ren, Shengchao Qin, and Florin Craciun. 2018. CDGDroid: Android Malware Detection Based on Deep Learning Using CFG and DFG. In *IEEE International Conference on Formal Engineering Methods*.
[78] Zhiwu XU, Kerong Ren, and Fu Song. 2019. Android Malware Family Classification and Characterization Using CFG and DFG. In *International Symposium on Theoretical Aspects of Software Engineering*. 49–56.
[79] Hui Xue, Qiang Yang, and Songcan Chen. 2009. SVM: Support vector machines. In *The top ten algorithms in data mining*. Chapman and Hall/CRC, 51–74.
[80] Lei Xue, Xiapu Luo, Le Yu, Shuai Wang, and Dinghao Wu. 2017. Adaptive Unpacking of Android Apps. In *IEEE/ACM International Conference on Software Engineering*. 358–369.
[81] Jian Yu, Yuewang He, Qiben Yan, and Xiangui Kang. 2021. SpecView: Malware Spectrum Visualization Framework With Singular Spectrum Transformation. *IEEE Transactions on Information Forensics and Security* 16 (2021), 5093–5107.
[82] Le Yu, Xiapu Luo, Jiachi Chen, Hao Zhou, Tao Zhang, Henry Chang, and Hareton K. N. Leung. 2021. PPChecker: Towards Accessing the Trustworthiness of Android Apps' Privacy Policies. *IEEE Transactions on Software Engineering* 47, 2 (2021), 221–242.
[83] Yueqian Zhang, Xiapu Luo, and Haoyang Yin. 2015. DexHunter: Toward Extracting Hidden Code from Packed Android Applications. In *European Symposium on Research in Computer Security*. 293–311.
[84] X. Zhi, T. Guo, Q. Zhang, C. Yu, and X. Kai. 2018. Android Malware Detection Methods Based on the Combination of Clustering and Classification. In *International Conference on Network and System Security*. 411–422.
[85] Q. Zhou, F. Feng, Z. Shen, R. Zhou, M. Y. Hsieh, and K. C. Li. 2019. A novel approach for mobile malware classification and detection in Android systems. *Multimedia Tools and Applications* 78 (2019), 3529–3552.
[86] Huijuan Zhu, Yang Li, Ruidong Li, Jianqiang Li, Zhuhong You, and Houbing Song. 2021. SEDMDroid: An Enhanced Stacking Ensemble Framework for Android Malware Detection. *IEEE Transactions on Network Science and Engineering* 8, 2 (2021), 984–994.
[87] H. J. Zhu, T. H. Jiang, B. Ma, Z. H. You, W. L. Shi, and L. Cheng. 2017. HEMD: a highly efficient random forest-based malware detection framework for Android. *Neural Computing & Applications* 30 (2017), 3353–3361.
[88] Jinting Zhu, Julian Jang-Jaccard, Amardeep Singh, Paul Watters, and Seyit Camtepe. 2023. Task-Aware Meta Learning-Based Siamese Neural Network for Classifying Control Flow Obfuscated Malware. *Future Internet* 15 (2023), 214.
[89] Jinting Zhu, Julian Jang-Jaccard, Amardeep Singh, Ian Welch, Harith AI-Sahaf, and Seyit Camtepe. 2022. A few-shot meta-learning based siamese neural network using entropy features for ransomware classification. *Computers & Security* 117 (2022), 102691–102720.

Received 3 July 2023; revised 4 May 2024; accepted 4 May 2024