Self-MCKD: Enhancing the Effectiveness and Efficiency of Knowledge Transfer in Malware Classification

Hyeon-Jin Jeong , Han-Jin Lee , Gwang-Nam Kim , Seok-Hwan Choi *

Posted Date: 7 March 2025

doi: 10.20944/preprints202501.2257.v2

Keywords: malware classification; lightweight deep learning; knowledge distillation

 doi:10.20944/preprints202501.2257.v2

Article

Self-MCKD: Enhancing the Effectiveness and Efficiency of Knowledge Transfer in Malware Classification

Hyeon-Jin Jeong, Han-Jin Lee, Gwang-Nam Kim and Seok-Hwan Choi *

Division of Software, Yonsei University, Wonju-si, Gangwon-do 26493, Republic of Korea
* Correspondence: sh.choi@yonsei.ac.kr

Abstract: As malware continues to evolve, AI-based malware classification methods have shown significant promise in improving malware classification performance. However, these methods lead to a substantial increase in computational complexity and the number of parameters, increasing computational cost during the training process. Moreover, the maintenance cost of these methods also increase, as frequent retraining and transfer learning are required to keep pace with evolving malware variants. In this paper, we propose an efficient knowledge distillation technique for AI-based malware classification methods, called **Self-MCKD**. **Self-MCKD** transfers output logits that are separated into target class and non-target classes. With the separation of output logits, **Self-MCKD** enables efficient knowledge transfer by assigning weighted importance to the target class and non-target classes. Also, **Self-MCKD** utilizes small and shallow AI-based malware classification methods as both the teacher and student models to overcome the need for using large and deep methods as the teacher model. From the experimental results under various malware datasets, we show that **Self-MCKD** outperforms traditional knowledge distillation techniques in terms of the effectiveness and efficiency of malware classification.

Keywords: malware classification; lightweight deep learning; knowledge distillation

1. Introduction

With the rapid development of the Internet, computers have become an integral part of people's daily lives. However, this growth has also led to a gradual increase in the number of malware, which has become one of the major threats to computer security. Some hackers exploit vulnerabilities in computers to steal data or obtain private information without user's permission. As a representative example, the WannaCry malware caused billions of dollars in damages to millions of computer devices[29].

To address such threats in a timely and effective manner, many countermeasures have been proposed. Traditionally, many studies have focused on malware classification methods using signature-based approach, relying on statistical methods. However, as malware continues to evolve with new variants, these methods have become less effective, leading to a decrease in classification performance.

In response, some studies have shifted towards AI-based methods, which have shown significant promise in improving malware classification performance[30]. Until recently, AI-based malware classification methods have focused on enhancing performance by increasing model size and depth. However, these methods lead to a substantial rise in computational complexity and the number of parameters, significantly raising computational cost during the training process. Moreover, the maintenance cost of AI-based malware classification methods also increases, as frequent retraining and transfer learning are required to keep up with evolving malware variants. These challenges not only increase the training time of AI-based malware classification methods but also increase the inference time. For example, such challenges present practical limitations when applying AI-based

 doi:10.20944/preprints202501.2257.v2

malware classification methods to IoT networks in CPS industrial, which is a resource-constrained environment[32].

To address these challenges, there is a growing need for research on lightweight techniques for AI-based malware classification methods. The existing lightweight techniques for AI-based malware classification methods are grouped into two categories of architectures, i.e., model compaction techniques and model optimization techniques. Model compaction techniques simplify the model structure by replacing layers with more efficient ones for lightweight AI-based malware classification methods. However, these methods suffer from computational complexity and long inference time. On the other hand, model optimization techniques reduce computational complexity and inference time by optimizing computational complexity of AI-based malware classification methods. In particular, knowledge distillation, which is a representative model optimization technique, has been actively studied as an efficient lightweight technique for AI-based malware classification methods.

Knowledge distillation allows small and shallow AI-based malware classification methods, as the student model, to mimic large and deep AI-based malware classification methods, as the teacher model, through knowledge transfer. Unfortunately, previous studies on knowledge distillation techniques for AI-based malware classification methods still face issues of computational complexity and inefficient knowledge transfer. Also, they require large and deep AI-based malware classification methods for the teacher model, which leads to high computational cost during the training process.

To solve these challenges, we propose an efficient knowledge distillation technique for AI-based malware classification methods, called **Self-MCKD**. As shown in Figure 1, different from traditional knowledge distillation techniques for AI-based malware classification methods, which transfer combined output logits composed of target class and non-target classes, **Self-MCKD** transfers output logits that are separated into them. With the separation of output logits, **Self-MCKD** enables efficient knowledge transfer by assigning weighted importance to the target class and non-target classes. Also, **Self-MCKD** only uses small and shallow methods as both the teacher and student models to overcome the need for using large and deep methods as the teacher model in knowledge distillation techniques for AI-based malware classification methods. Since **Self-MCKD** enables efficient knowledge transfer, it not only enhances the performance of small and shallow AI-based malware classification methods but also enables them to surpass large and deep methods. Also, it accelerates convergence speed and reduces the training time of small and shallow AI-based malware classification methods.

Main contributions of this paper can be summarized as follows:

- We propose **Self-MCKD**, a novel knowledge distillation technique for AI-based malware classification methods. **Self-MCKD** separates the output logits into the target class and non-target classes, assigning a weighted importance factor that maximizes knowledge transfer effectiveness.
- To enhance knowledge transfer efficiency, we introduce small and shallow AI-based malware classification methods as both the teacher and student models in **Self-MCKD**.
- From the experimental results using representative benchmark datasets, we show that **Self-MCKD** significantly improves the performance of a small and shallow baseline model, even surpassing that of a large and deep baseline model. Also, we show that Self-MCKD outperforms previous knowledge distillation techniques in both effectiveness and efficiency. Specifically, with the same FLOPs, MACs, and parameters, **Self-MCKD** achieves improvements in accuracy, recall, precision, and F1-score by 0.2% to 1.4%, while reducing training time by 20 seconds.

 doi:10.20944/preprints202501.2257.v2

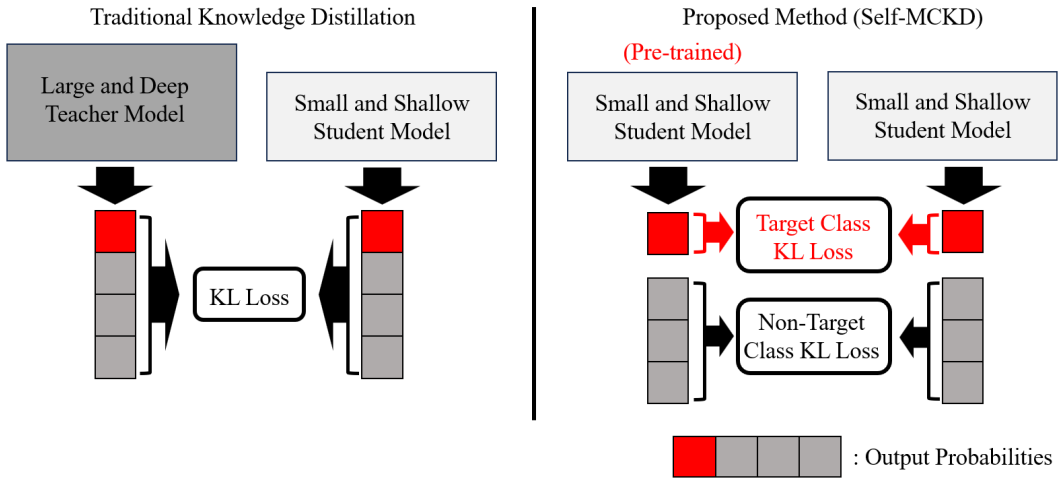

Figure 1. Difference between traditional knowledge distillation and proposed **Self-MCKD**

2. Related Works

In this section, we first overview previous AI-based malware classification methods and lightweight techniques for AI-based malware classification methods, then we introduce the most effective knowledge distillation-based technique for AI-based malware classification methods.

2.1. AI-Based Malware Classification Methods

In the field of malware classification, many previous methods have utilized executable files represented by opcodes or assembly codes [18]. However, these methods faced several limitations, such as the difficulty of analyzing obfuscated code, the time-consuming nature of static analysis, and the high resource demands of dynamic analysis[19]. To address these limitations, several studies have applied AI-based methods, specifically machine learning (**ML**) and deep learning (**DL**), after converting executable files into gray-scale images [17]. Nataraj et al. were the first to apply the K-Nearest Neighbors (KNN) algorithm, an **ML** technique, to executable files converted into gray-scale images for malware classification [1]. To achieve improved performance over KNN algorithm, Hardy et al. proposed an SVM-based malware classification method [20]. Also, Drew et al. leveraged modern gene classification tools [21], and Ahmadi et al. leveraged XGBoost techniques [22], to progressively improve the performance of malware classification with more advanced **ML** models. However, as the variety of malware variants increased, the complexity of malware analysis also increased. Moreover, this revealed certain limitations of **ML**-based malware classification methods, especially their inability to automatically extract and learn nuanced features of malware variants.

To address limitations of **ML**-based malware classification methods, **DL**-based malware classification methods, which can automatically extract nuanced features, have emerged. In particular, convolutional neural network (CNN)-based malware classification methods have gained prominence due to their superior image analysis capabilities when applied to executable files converted into gray-scale images. Daniel et al. [2] applied a shallow, simple CNN model for malware classification using gray-scale images of malware. They also showed that **DL**-based malware classification methods outperform previous **ML**-based malware classification methods. Following this study, Mahmoud et al. hypothesized that deeper CNN architectures could further improve the performance of malware classification [3]. Based on this assumption, they enhanced the performance of **DL**-based malware classification methods using the VGG16 model. To achieve even better classification performance, Pratikkumar et al. proposed a migration learning framework for malware classification, utilizing more advanced models such as ResNet52 and VGG19 models [4].

 doi:10.20944/preprints202501.2257.v2

2.2. Lightweight Techniques for **DL**-Based Malware Classification Methods

As **DL**-based malware classification methods have become increasingly complex and deeper, the associated hardware burden has also increased. Also, since this has led to longer training and inference times, several recent studies have focused on developing lightweight **DL**-based malware classification methods. In this section, we summarize and discuss key studies related to the lightweight techniques for **DL**-based malware classification methods. Early lightweight techniques for **DL**-based malware classification methods focused on simplifying the model structure, also known as model compaction techniques. Specifically, Fathurrahman et al. [23] proposed a lightweight malware classification method by replacing the deep structures of the CNN model with residual layers and global average pooling. This approach outperformed the lightweight CNN-based malware classification method proposed by Su et al. [24], offering both reduced complexity and improved performance. However, model compaction techniques, which simply replace layer structures, still suffer from computational complexity and long inference time [5,6,10].

To address these challenges, efficient lightweight techniques, also known as model optimization techniques, have emerged. These techniques, such as weight pruning [7], model quantization [8], and knowledge distillation [9], reduced computational complexity and inference time in **DL**-based malware classification methods. In particular, knowledge distillation has been actively studied as an efficient lightweight technique for **DL**-based malware classification methods. Zhi et al. proposed a knowledge distillation-based technique for recurrent neural network (RNN)-based malware classification methods [11]. Specifically, they enhanced the performance of the RNN-based malware classification method (student model), using a transformer-based malware classification method, as the teacher model. Xia et al. [12] also proposed a lightweight technique for multi-layer perceptron (MLP)-based malware classification methods using knowledge distillation. Different from Zhi et al.'s technique, which uses different families of models for the teacher and student models, Xia et al.'s technique uses the same family of MLP models for both. By using the same MLP family of models, they improved the performance of the student MLP-based malware classification method to a level comparable to the teacher MLP-based model. Building on this foundation, recent studies have further combined effective learning methods with knowledge distillation, such as the two-dimensional Fourier transform and federated learning[33,34].

Previous knowledge distillation-based techniques for **DL**-based malware classification methods showed that small and shallow student models could achieve performance comparable to large and deep teacher models. However, despite the promising performance of these techniques, they still suffered from computational complexity and inefficient knowledge transfer. Moreover, many of these techniques relied on large and deep DL-based methods as the teacher model, which incurred high computational cost during training process. These problems motivate us to seek a new knowledge distillation-based technique for **DL**-based malware classification methods, which enables efficient knowledge transfer with minimal computational cost and complexity.

3. Preliminary

3.1. Knowledge Distillation

Knowledge distillation was first introduced by Bucila et al. for model compression [13] and gained popularity after Hinton et al. generalized it [9]. Knowledge distillation is a model optimization technique that allows the small and shallow student model to mimic the performance of a large and deep teacher model. Specifically, the student model S is trained to mimic the performance of the teacher model T by leveraging the teacher's "Dark Knowledge". Here, "Dark Knowledge" refers to the softened class probabilities predicted by the teacher model, which are distilled into the student model. By utilizing this "Dark Knowledge", the student model improves its performance by learning not only from the original training data but also by incorporating the nuanced knowledge provided by the teacher's predictions. The process of distilling this "Dark Knowledge", which is called knowledge distillation process(KD), can be expressed as follows:

 doi:10.20944/preprints202501.2257.v2

$$KD = \sum_{x_i \in X} \mathrm{KL}\left(softmax\left(\frac{f_T(x_i)}{\tau}\right), softmax\left(\frac{f_S(x_i)}{\tau}\right)\right), \quad (1)$$

where X represents the training dataset, x_i represents individual data points, $f_T(x_i)$ and $f_S(x_i)$ represent the class probabilities predicted by the teacher and student models, respectively. Additionally, τ is the temperature parameter used to soften the class probabilities. The Kullback-Leibler(KL) divergence is used as the loss function to measure the difference between the softened outputs of the teacher and student models. By minimizing KL divergence, the student model is trained to mimic the behavior of the teacher model, effectively absorbing the teacher's "Dark Knowledge" and enhancing its own classification performance. Knowledge distillation can be implemented in two main ways: logit-based and feature-based approaches. In Figure 2, we show the differences between these two knowledge distillation approaches.

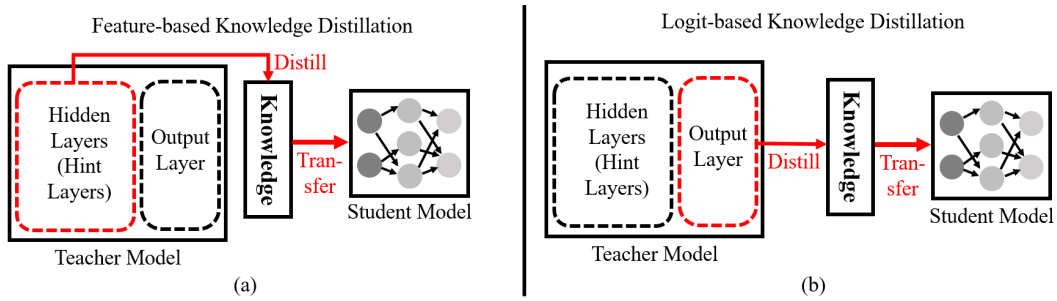

Figure 2. the differences between feature-based and logit-based distillation approaches

As shown in Figure 2 (a), the feature-based approach, which was first proposed by Romero et al. [31], distills knowledge to the student model from the teacher model's hint layers. This approach focuses on training the student to mimic the output of the intermediate layers. Specifically, the feature-based approach transfers the rich knowledge of the teacher to the student, enabling the student to achieve performance similar to that of the teacher. However, since the knowledge distilled from the hint layers has high computational complexity, the feature-based approach requires significant training time for the student model. Also, the feature-based approach is inefficient for lightweight techniques of **DL**-based malware classification methods because **DL**-based malware classification methods require frequent training and transfer learning.

On the other hand, the logit-based approach distills knowledge to the student model from the teacher model's output layer, which produces softened class probabilities as shown in Figure 2 (b). Different from the feature-based approach, this approach focuses on training the student to mimic the output logits of the teacher. Also, since the logit-based approach has low computational complexity, it can be a solution to the long training time problem of the feature-based approach. For example, traditional knowledge distillation, which is a representative logit-based approach, enables the student model to reduce the training time compared to the feature-based approach. Unfortunately, the existing logit-based approaches suffer from inefficient knowledge transfer, which limits the performance of the student model. Therefore, we propose a new logit-based approach, which can transfer knowledge more efficiently to the student model compared to the existing logit-based approaches.

4. Proposed Method

In this section, we overview the operation of **Self-MCKD** in detail. First, we introduce the difference between traditional knowledge distillation and **Self-MCKD**, then we reformulate traditional knowledge distillation for **Self-MCKD**. Next, we describe the network architecture in **Self-MCKD** used for both the teacher and student models.

 doi:10.20944/preprints202501.2257.v2

4.1. Revisiting Traditional Knowledge Distillation

Traditional knowledge distillation transfers combined knowledge of target and non-target classes to the student model. However, this approach suffers from inefficient knowledge transfer, as combined knowledge is often emphasized in an unbalanced manner, reducing its effectiveness [14]. Also, traditional knowledge distillation tends to use high-performing large and deep teacher models, which require high computational cost during the training process. In the field of malware classification, effective knowledge transfer is essential for better generalization against evolving malware variants. However, traditional knowledge distillation has limitations in this domain due to its unbalanced knowledge transfer. Moreover, its reliance on large and deep teacher models incurs high computational costs, making frequent retraining challenging and limiting adaptability to newly emerging malware threats.

To address these problems in the field of malware classification, we propose a novel knowledge distillation technique called **Self-MCKD**. Different from traditional knowledge distillation, **Self-MCKD** transfers the knowledge of target and non-target classes independently. Also, **Self-MCKD** does not rely on a large and deep teacher model. Instead, it utilizes a pre-trained small and shallow student model to serve as the teacher, transferring knowledge to the untrained student model during the training process.

4.2. Notations for ***Self-MCKD***

In this section, we define two types of output probabilities—target malware class probabilities and non-target malware class probabilities—to reformulate knowledge transfer process of traditional knowledge distillation. When a malware sample of the i-th class is given, the output probabilities of the malware classification model are defined as $\mathbf{p} = [p_1, p_2, p_3, \ldots, p_i, \ldots, p_M] \in \mathbb{R}^{1\times M}$, where p_i represents the output probabilities of the i-th malware class, and M is the number of malware families. Here, p_i is calculated as follows:

$$p_i = \frac{\exp(z_i)}{\sum_{j=1}^{M} \exp(z_j)}, \tag{2}$$

where z_i is the output logit for the i-th malware class in the malware classification model.

Next, to separate the output probabilities related to the target malware class and non-target malware classes, we define binary probabilities as $\mathbf{bm} = [p_{tm}, p_{ntm}] \in \mathbb{R}^{1\times 2}$. Here, p_{tm} and p_{ntm} can be expressed into:

$$p_{tm} = \frac{\exp(z_{tm})}{\sum_{j=1}^{M} \exp(z_j)}, p_{ntm} = \frac{\sum_{k=1,k\neq tm}^{M} \exp(z_k)}{\sum_{j=1}^{M} \exp(z_j)}, \tag{3}$$

where p_{tm} represents the probabilities of the target malware class, and p_{ntm} represents the probabilities of all the other non-target malware classes. Meanwhile, we defined $\hat{\mathbf{p}} = [\hat{p}_1, \ldots, \hat{p}_{i-1}, \hat{p}_{i+1}, \ldots, \hat{p}_{M-1}] \in \mathbb{R}^{1\times(M-1)}$ that means the output probabilities of non-target malware classes, i.e., excluding the i-th malware class. Here, $\hat{p}_i$ is calculated as follows:

$$\hat{p}_i = \frac{\exp(z_i)}{\sum_{j=1,j\neq tm}^{M} \exp(z_j)}. \tag{4}$$

4.3. Reformulating Knowledge Distillation for ***Self-MCKD***

Based on above notations, we reformulate the knowledge distillation process(KD) from Equation (1) as follow:

 doi:10.20944/preprints202501.2257.v2

$$\begin{aligned} KD &= KL(p^{\mathcal{T}}||p^{\mathcal{S}}) \\ &= p^{\mathcal{T}}_{tm}\log(\frac{p^{\mathcal{T}}_{tm}}{p^{\mathcal{S}}_{tm}}) + \sum_{i=1,i\neq tm}^{M} p^{\mathcal{T}}_{i}\log(\frac{p^{\mathcal{T}}_{i}}{p^{\mathcal{S}}_{i}}), \end{aligned} \tag{5}$$

where $p^{\mathcal{T}}$ represents the output probabilities of the teacher model and $p^{\mathcal{S}}$ represents the output probabilities of the student model.

Different from the loss function of traditional knowledge distillation, which aims to simply reduce the overall difference in probabilities between the teacher and student models, **Self-MCKD** aims to reduce the difference for both the target and non-target malware classes separately between the teacher and student models. Also, since **Self-MCKD** replaces the large and deep teacher model with a pre-trained small and shallow student model, $p^{\mathcal{T}}$ can be replaced with $p^{t\mathcal{S}}$. Finally, considering that $\hat{p}$ equals p_i/p_{ntm}, we can reformulate Equation (5) as follows:

$$\begin{aligned} KD &= p^{t\mathcal{S}}_{tm}\log\left(\frac{p^{t\mathcal{S}}_{tm}}{p^{\mathcal{S}}_{tm}}\right) + \sum_{i=1,i\neq tm}^{M} p^{t\mathcal{S}}_{ntm}\hat{p}^{t\mathcal{S}}_{i}\left(\log\left(\frac{\hat{p}^{t\mathcal{S}}_{i}}{\hat{p}^{\mathcal{S}}_{i}}\right) + \log\left(\frac{p^{t\mathcal{S}}_{ntm}}{p^{\mathcal{S}}_{ntm}}\right)\right) \\ &= p^{t\mathcal{S}}_{tm}\log\left(\frac{p^{t\mathcal{S}}_{tm}}{p^{\mathcal{S}}_{tm}}\right) + p^{t\mathcal{S}}_{ntm}\log\left(\frac{p^{t\mathcal{S}}_{ntm}}{p^{\mathcal{S}}_{ntm}}\right) + p^{t\mathcal{S}}_{ntm}\sum_{i=1,i\neq tm}^{M}\hat{p}^{t\mathcal{S}}_{i}\log\left(\frac{\hat{p}^{t\mathcal{S}}_{i}}{\hat{p}^{\mathcal{S}}_{i}}\right) \\ &= KL(\mathbf{bm}^{t\mathcal{S}}||\mathbf{bm}^{\mathcal{S}}) + p^{t\mathcal{S}}_{ntm}KL(\hat{\mathbf{p}}^{t\mathcal{S}}||\hat{\mathbf{p}}^{\mathcal{S}}). \end{aligned} \tag{6}$$

After reformulating the Equation (6), it can be written as follows:

$$KD = KL(\mathbf{bm}^{t\mathcal{S}}||\mathbf{bm}^{S}) + p^{t\mathcal{S}}_{ntm}KL(\hat{\mathbf{p}}^{t\mathcal{S}}||\hat{\mathbf{p}}^{S}). \tag{7}$$

As shown in Equation (7), the loss function of traditional knowledge distillation can be reformulated as a weighted sum of two terms. Here, $KL(\mathbf{bm}^{t\mathcal{S}}||\mathbf{bm}^{S})$ represents the knowledge of the target malware class between the teacher and student models, which we refer to as (Target Malware Class Knowledge Distillation) **TMKD**. On the other hand, $KL(\hat{\mathbf{p}}^{t\mathcal{S}}||\hat{\mathbf{p}}^{S})$ represents the knowledge of the non-target malware classes between the teacher and student models, which we refer to as (Non-Target Malware Class Knowledge Distillation) **NMKD**. These separated knowledges of **Self-MCKD** are shown in Figure 3. According to Zhao et al.'s work [14], the $p^{t\mathcal{S}}_{ntm}$ factor, derived from traditional knowledge distillation, enforces an emphasis on non-target malware classes. This enforced emphasis prevents from autonomously adjusting the importance of target and non-target malware classes. As a result, it restricts the effectiveness of knowledge transfer to student models. To address the restrictive effect of the $p^{t\mathcal{S}}_{ntm}$ factor, we introduce a factor β. Also, to proper emphasis on **TMKD**, we introduce a factor α. Consequently, these factors enable **Self-MCKD** to emphasize **TMKD** and **NMKD**, respectively. By applying these factors and making the necessary substitutions to Equation (7), the loss function of **Self-MCKD** can be defined as Equation (8) and calculated using Algorithm 1.

$$\textbf{Self-MCKD} = \alpha\textbf{TMKD} + \beta\textbf{NMKD}. \tag{8}$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 March 2025 doi:10.20944/preprints202501.2257.v2

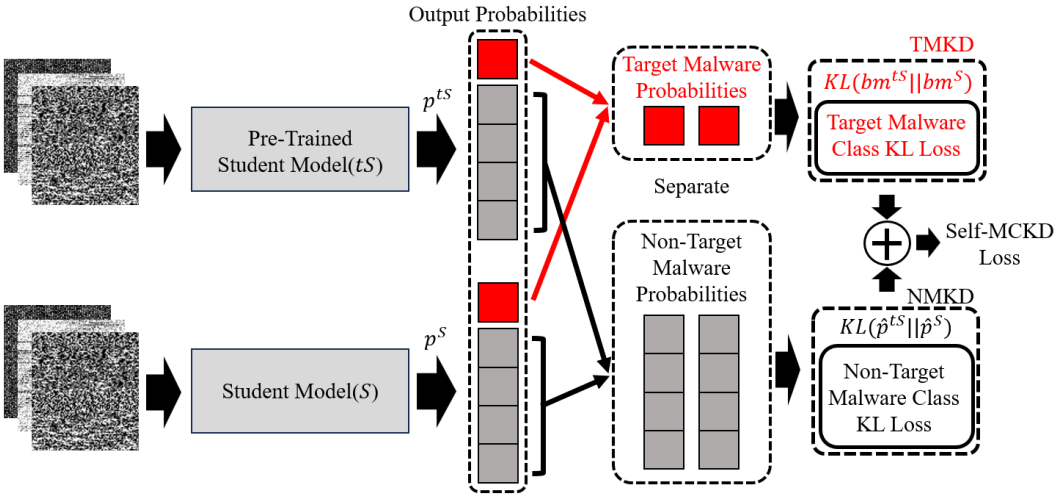

Figure 3. Structure of the Self-MCKD

Algorithm 1 Self-MCKD Loss Function Pseudo Code

Input: Student logits $\mathbf{z}^S$, Pre-trained Student logits $\mathbf{z}^{tS}$, Target Malware $\mathbf{y}$, Factors α, β, Temperature τ
Output: Self-MCKD Loss

1. Compute target malware and non-target malware masks:

$$\text{GT\_Mask} \leftarrow \text{One-hot}(\mathbf{y})$$

$$\text{Other\_Mask} \leftarrow 1 - \text{GT\_Mask}$$

2. Normalize logits with temperature:

$$p^S \leftarrow \text{Softmax}(\mathbf{z}^S/\tau)$$

$$p^{tS} \leftarrow \text{Softmax}(\mathbf{z}^{tS}/\tau)$$

3. Concatenate probabilities:

$$p^s \leftarrow \text{Concat}(p^s \cdot \text{GT\_Mask}, p^s \cdot \text{Other\_Mask})$$

$$p^{tS} \leftarrow \text{Concat}(p^{tS} \cdot \text{GT\_Mask}, p^{tS} \cdot \text{Other\_Mask})$$

4. Compute **TMKD** Loss:

$$\textbf{TMKD} \leftarrow \text{KL}(\log(p^s), p^{tS}) \cdot \tau^2$$

5. Compute **NMKD** Loss:

$$\textbf{NMKD} \leftarrow \text{KL}(\log(p^s - \text{GT\_Mask})$$
$$p^{tS} - \text{GT\_Mask}) \cdot \tau^2$$

6. Return total loss:

$$\textbf{Self-MCKD} \leftarrow \alpha\textbf{TMKD} + \beta\textbf{NMKD}$$

5. Evaluation Results

To demonstrate the effectiveness and efficiency of the proposed **Self-MCKD** in the field of malware classification, we evaluated its performance across various datasets. Specifically, we aimed to evaluate the lightweighting effect of **Self-MCKD** when applied to representative **DL**-based malware classification methods in different dataset environments. To this end, we tried to answer the following questions:

- How do factors α of **TMKD** and β of **NMKD** influence on the performance of **Self-MCKD**?

 doi:10.20944/preprints202501.2257.v2

- Does **Self-MCKD** show good performance on small and shallow malware classification model under various malware datasets?
- Does **Self-MCKD** show the better performance than the other knowledge distillation method?

5.1. Experimental Environments

In this section, we describe experimental environments including dataset, model configuration, evaluation metric, and so on.

Dataset: We conducted the experiments using the Malimg dataset(Malimg) [28] and the Microsoft Malware Classification Challenge Dataset(MMCC) [27], both of which are representative datasets for malware classification. The Malimg, originally compiled by Nataraj et al. [1], consists of 9,339 malware samples represented as grayscale images, corresponding to 25 malware families. These families include "Adialer", "Allaple", "Lolyda", "C2LOP", and "Swizzor", and several others. The MMCC dataset, provided by Microsoft Malware Classification Challenge, consists of 21,741 malware samples categorized into 9 malware families, namely "Ramni", "Lollipop", "Kelihos_ver3", "Vundo", "Simda", "Tracur", "Kelihos_ver1", "Obfuscator.ACY", and "Gatak". As with Malimg, the distribution of malware samples across classes in the MMCC dataset is not uniform, with some families having significantly more samples than others. In our experiments, we randomly select 90% of malware samples from each family for training and used the remaining 10% for testing.

Model Configuration: To demonstrate the effectiveness and efficiency of the proposed **Self-MCKD**, we selected representative **DL**-based malware classification methods, such as ResNet and VGGNet. In previous malware classification studies [3,35], ResNet and VGGNet achieved promising classification performance. Thus, we used ResNet110 and VGG16 models as large and deep baseline models for malware classification. Also, to apply small and shallow baseline models in the proposed **Self-MCKD**, we selected much lighter homogeneous models, such as ResNet32 and VGG8 models. For the implementation of these models, we followed the original ResNet and VGGNet architecture [16,35] without modifications. Also, we set these models' output layer to 9 or 25, depending on the number of malware families in the dataset, and used the stochastic gradient descent(SGD) optimization function to train them. Lastly, we set hyperparameters, such as batch size, epoch, learning rate, momentum, and weight decay to 8, 25, 0.001, 0.9, and 0.0005, respectively.

Evaluation Metric: To evaluate the performance of **Self-MCKD**, we measured accuracy, recall, precision, and F1-score under different malware datasets. These metrics are derived from the confusion matrix. In the confusion matrix, True Positive (TP) represents correctly predicted samples of the target malware class, while True Negative (TN) represents correctly predicted samples of the non-target malware class. False Positive (FP) represents non-target malware class samples that are incorrectly predicted as belonging to the target malware class. On the other hand, False Negative (FN) represents target malware class samples incorrectly predicted as belonging to the non-target malware class. Based on the confusion matrix, accuracy, recall, precision, and F1-score are calculated as follows:

$$Accuracy = \frac{TP + TN}{FP + TP + FN + TN}. \tag{9}$$

$$Recall = \frac{TP}{TP + FN}. \tag{10}$$

$$Precision = \frac{TP}{TP + FP}. \tag{11}$$

$$F1\text{-}score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}. \tag{12}$$

Also, to consider the computational efficiency and complexity of malware classification baseline models, we measured floating point operations (FLOPs), multiply-accumulate operations (MACs), and the number of model parameters. Here, FLOPs refer to the total number of floating-point

 doi:10.20944/preprints202501.2257.v2

operations required during a forward pass and MACs represent the number of multiply-accumulate operations. such metrics reflect the computational cost of the convolutional and fully connected layers. In knowledge distillation, both the student and teacher models contribute to the FLOPs and MACs during the inference process. However, the number of parameters, which represents the total trainable parameters of malware classification baseline models, is measured only for the student model during the training process. We conducted experiments five times and reported the average results to ensure a reliable evaluation based on these metrics.

Implementation Environment: We implemented malware classification baseline models, knowledge distillation methods using PyTorch version 2.2.2 and Python version 3.11.9. For efficient experiments, the performance was measured on Windows OS with PyTorch-CUDA version 12.1, Intel Core i9-12900KF processor, NVIDIA RTX 3060 GPU, and 64 GB of memory.

5.2. Experimental Analysis

5.2.1. How Do Factors α of **TMKD** and β of **NMKD** Influence on the Performance of **Self-MCKD**?

To evaluate the impact of the factors α and β on the performance of **Self-MCKD**, we measured the accuracy of the malware classification model while adjusting these factors. Simultaneously, we aimed to verify that these factors provide better performance compared to the $p_{ntm}^{t\mathcal{S}}$ factor derived from the traditional knowledge distillation reformulation. To determine the appropriate values for α and β, we conducted experiments by setting their sum to 1 to ensure stability during the training process and prevent degradation of knowledge transfer effectiveness. Also, we varied the α and β values from (0.1, 0.9) to (0.9, 0.1).

In Table 1, we show the experimental results of the VGG8 model applied to **Self-MCKD** using MMCC and Malimg datasets. According to Table 1, it is observed that too much emphasis on **NMKD** by increasing the factor β reduced the effectiveness of knowledge transfer. For example, in the experimental results on both datasets, while the VGG8 model with traditional knowledge distillation achieved an accuracy of 98.44% and 99.36%, the VGG8 model with **Self-MCKD** achieved an accuracy of 98.07% and 99.25%, when we set α and β at 0.2 and 0.8, respectively. Also, we observed that too much emphasis on **TMKD** by increasing the factor α also negatively affected knowledge transfer. For example, in the experimental results on MMCC dataset, the VGG8 model with **Self-MCKD**, where α and β were set to 0.9 and 0.1, achieved an accuracy of 98.25%, which is lower than an accuracy of 98.44% achieved by the VGG8 model with traditional knowledge distillation.

Table 1. Performance variation with changes in α and β factors of **Self-MCKD**

MMCC	(α, β)	(0.1, 0.9)	(0.2, 0.8)	(0.3, 0.7)	(0.4, 0.6)	(0.5, 0.5)
	Accuracy	98.25%	98.07%	98.44%	98.34%	98.52%
	(α, β)	(0.6, 0.4)	(0.7, 0.3)	(0.8, 0.2)	(0.9, 0.1)	$p_{ntm}^{t\mathcal{S}}$
	Accuracy	98.25%	**98.62%**	98.44%	98.25%	98.44%
Malimg	(α, β)	(0.1, 0.9)	(0.2, 0.8)	(0.3, 0.7)	(0.4, 0.6)	(0.5, 0.5)
	Accuracy	99.25%	99.25%	99.36%	99.25%	99.46%
	(α, β)	(0.6, 0.4)	(0.7, 0.3)	(0.8, 0.2)	(0.9, 0.1)	$p_{ntm}^{t\mathcal{S}}$
	Accuracy	99.57%	**99.57%**	99.57%	99.57%	99.36%

On the other hand, when balancing between **TMKD** and **NMKD**, we observed that **Self-MCKD** transferred knowledge more effectively than traditional knowledge distillation. Specifically, when we set both α and β to 0.5, the VGG8 model with **Self-MCKD** achieved accuracy of 98.52% and 99.46%, respectively, which are higher than that of the VGG8 model with traditional knowledge distillation. Also, it is observed that a proper balance between **TMKD** and **NMKD** maximizes the effectiveness of knowledge transfer. For example, the VGG8 model with **Self-MCKD**, with α and β were set to 0.7 and

 doi:10.20944/preprints202501.2257.v2

0.3, achieved an accuracy of 98.62% and 99.57%, respectively, which outperformed the VGG8 model with traditional knowledge distillation.

5.2.2. Does **Self-MCKD** Shows Good Performance on Small and Shallow Malware Classification Model Under Various Malware Datasets?

To show the effectiveness and efficiency of **Self-MCKD** on small and shallow malware classification models under various datasets, we measured the computational complexity and accuracy of ResNet32 and VGG8 models, both with and without **Self-MCKD** using MMCC and Malimg datasets. Simultaneously, we aimed to show that the ResNet32 and VGG8 models with **Self-MCKD** outperforms larger and deeper malware classification models by comparing their performance to ResNet110 and VGG16 baseline models. Specifically, we measured FLOPs, MACs and the number of model parameters to assess computational complexity, while evaluating performance in terms of accuracy and training time. In Table 2, we show the experimental results for ResNet and VGGNet under MMCC and Malimg datasets.

From the experimental results, we first observed that the large and deep malware classification models outperformed small and shallow malware classification models under various datasets. For example, on the MMCC dataset, ResNet32 and VGG8 baseline models achieved accuracies of 96.87% and 97.70%, respectively, which are lower than accuracies of 97.79% and 98.25% achieved by the ResNet110 and VGG16 baseline models. Also, on the Malimg dataset, the ResNet110 and VGG16 baseline models achieved accuracies of 99.36% and 99.25%, respectively, while the ResNet32 and VGG8 baseline models both achieved an accuracy of 99.04%. However, deeper models such as the ResNet110 and VGG16 baseline models cost approximately four times more in terms of FLOPs, MACs, and parameters compared to shallower models such as ResNet32 and VGG8. Also, these models required more than twice the training time.

Second, it is observed that the small and shallow malware classification models with **Self-MCKD** achieved better performance than the large and deep malware classification models, while using only minimal additional computational cost. For example, on the MMCC dataset, while the VGG16 baseline model achieved an accuracy of 98.25% using 627.04M FLOPs, 313.52M MACs, and 14.77M parameters, the VGG8 model with **Self-MCKD** achieved an accuracy of 98.62% with only 272.02M FLOPs, 136.01M MACs, and 3.96M parameters. Also, while the VGG16 baseline model took 91 minutes and 8 seconds to train, the VGG8 model with **Self-MCKD** required only 28 minutes and 19 seconds, which is more than three times faster. Similarly, the ResNet110 baseline model required 508.42M FLOPs, 254.21M MACs, and 1.74M parameters to achieve an accuracy of 97.79%. Its performance is higher than the ResNet32 baseline model, which achieved an accuracy of 96.87% with 138.89M FLOPs, 69.45M MACs, and 0.47M parameters. However, the ResNet32 model with **Self-MCKD** outperformed the ResNet110 baseline model with 277.78M FLOPs, 138.90M MACs, and 0.47M parameters. For the Malimg dataset, the ResNet32 and VGG8 models with **Self-MCKD** achieved comparable performance to the ResNet110 and VGG16 baseline models. For example, while the ResNet110 and VGG16 baseline models achieved accuracies of 99.36% and 99.25%, respectively, the ResNet32 and VGG8 models with **Self-MCKD** achieved accuracies of 99.25% and 99.57%, respectively.

Third, we observed that the small and shallow malware classification models with **Self-MCKD** effectively transfers knowledge from the teacher to student models. In other words, **Self-MCKD** not only improved the classification accuracy of the student model but also enhanced its convergence speed. Specifically, the accuracy of the ResNet32 and VGG8 baseline models improved from 96.87% and 97.70% to 98.53% and 98.62%, respectively, using **Self-MCKD** on the MMCC dataset. Similarly, both the ResNet32 and VGG8 baseline models achieved an accuracy of 99.04% without **Self-MCKD**. On the other hand, for both models with **Self-MCKD**, the average accuracy increased by approximately 0.3%. Such results show that the ResNet32 and VGG8 models with **Self-MCKD** maintained the number of parameters while reducing the training time by approximately 1.5 times compared to the ResNet32 and VGG8 baseline models.

 doi:10.20944/preprints202501.2257.v2

Table 2. Comparison results on computational complexity and performance with baseline models under various malware datasets

Dataset	Model		FLOPs	MACs	Params	Accuracy	Training time
MMCC	Baseline	VGG16	627.04M	313.52M	14.77M	98.25%	91m 8s
		VGG8	136.01M	68.00M	3.96M	97.70%	42m 23s
		ResNet110	508.42M	254.21M	1.74M	97.79%	94m 17s
		ResNet32	138.89M	69.45M	0.47M	96.87%	48m 39s
	Self-MCKD + VGG8		272.02M	136.01M	3.96M	**98.62%**	**28m 19s**
	Self-MCKD + ResNet32		277.78M	138.90M	0.47M	**98.53%**	**35m 21s**
Malimg	Baseline	VGG16	627.04M	313.52M	14.77M	99.25%	72m 52s
		VGG8	136.01M	68.00M	3.96M	99.04%	33m 18s
		ResNet110	508.42M	254.21M	1.74M	99.36%	82m 5s
		ResNet32	138.89M	69.45M	0.47M	99.04%	41m 51s
	Self-MCKD + VGG8		272.02M	136.01M	3.96M	**99.57%**	**27m 11s**
	Self-MCKD + ResNet32		277.78M	138.90M	0.47M	**99.25%**	**30m 52s**

5.2.3. Does **Self-MCKD** show the better performance than the other knowledge distillation methods?

To compare the classification performance of **Self-MCKD** with other logit-based knowledge transfer methods, we conducted experiments using Vanilla KD and MLKD. We measured the accuracy, precision, recall, F1-score, and training time of Vanilla KD, MLKD and **Self-MCKD**. Here, Vanilla KD refers to a traditional knowledge distillation method that transfers knowledge, combining both target and non-target classes, from the teacher model to the student model. Also, MLKD refers to Multi-Level Logit Knowledge Distillation (MLKD) [36], a state-of-the-art method that distills knowledge at multiple logit levels. Unlike these approaches, **Self-MCKD** transfers knowledge of target and non-target classes independently. To compare the effectiveness of knowledge transfer, we used the same small and shallow malware classification models as the teacher and the student model in Vanilla KD and MLKD, as with **Self-MCKD**. For all methods, including Vanilla KD, MLKD, and **Self-MCKD**, we set the temperature factor to 4, following common practices in knowledge distillation experiments [9,14,36].

As shown in Table 3, we observed that Vanilla KD and MLKD also improved small and shallow malware classification models. For example, on the MMCC dataset, the VGG8 model with Vanilla KD achieved an accuracy, recall, precision, and F1-score of 98.44%, 98.34%, 98.36%, and 98.29%, respectively, while the VGG8 baseline model achieved 97.70%, 97.70%, 97.67%, and 97.68%, respectively. Also, for ResNet32, MLKD achieved an accuracy, recall, precision, and F1-score of 97.70%, 97.70%, 97.74%, and 97.57%, respectively, while the ResNet32 baseline model achieved an accuracy, recall, precision, and F1-score of 96.87%, 97.42%, 97.39%, and 97.39%, respectively. Similarly, for the Malimg dataset, it is observed that the VGG8 model with Vanilla KD and MLKD improved its malware classification accuracy, recall, precision, and F1-score by approximately 0.3%. Especially, MLKD outperformed Vanilla KD in recall, precision, and F1-score across both datasets. However, although Vanilla KD and MLKD reduced the training time by approximately 1.5 times compared to the baseline, MLKD required slightly longer training time than Vanilla KD.

 doi:10.20944/preprints202501.2257.v2

Table 3. Comparison results with knowledge distillation methods under various malware datasets

Dataset	Baseline	Technique	Accuracy	Recall	Precision	F1-score	Training Time
MMCC	VGG8	None	97.70%	97.70%	97.67%	97.68%	42m 23s
		Vanilla KD	98.44%	98.34%	98.36%	98.29%	28m 43s
		MLKD	98.43%	98.44%	98.44%	98.37%	33m 34s
		Self-MCKD	**98.62%**	**98.71%**	**98.73%**	**98.69%**	**28m 19s**
	ResNet32	None	96.87%	97.42%	97.39%	97.39%	48m 39s
		Vanilla KD	97.79%	97.06%	96.83%	96.82%	35m 26s
		MLKD	97.70%	97.70%	97.74%	97.57%	38m 20s
		Self-MCKD	**98.53%**	**98.44%**	**98.47%**	**98.41%**	**35m 21s**
Malimg	VGG8	None	99.04%	98.72%	98.81%	98.73%	33m 18s
		Vanilla KD	99.36%	99.04%	99.08%	99.03%	27m 13s
		MLKD	99.25%	99.25%	**99.32%**	99.25%	29m 1s
		Self-MCKD	**99.57%**	**99.25%**	99.29%	**99.25%**	**27m 11s**
	ResNet32	None	99.04%	99.25%	99.28%	99.26%	41m 51s
		Vanilla KD	99.14%	98.82%	98.96%	98.80%	30m 54s
		MLKD	98.92%	98.82%	98.85%	98.82%	31m 39s
		Self-MCKD	**99.25%**	**99.25%**	**99.36%**	**99.26%**	**30m 52s**

Even though Vanilla KD and MLKD improved the performance of small and shallow malware classification models, **Self-MCKD** outperformed both methods in all metrics. For example, on the MMCC dataset, the ResNet32 and VGG8 models with **Self-MCKD** surpassed the accuracy, recall, precision, and F1-score of same models with Vanilla KD and MLKD by about 0.2% to 0.8%. Similarly, on the Malimg dataset, the ResNet32 model with **Self-MCKD** achieved higher accuracy, recall, precision, and F1-score of same models with Vanilla KD and MLKD by about 0.4%. Notably, **Self-MCKD** showed shorter training times compared to baseline and other knowledge distillation methods.

In Figure 4, we showed ROC curves along with AUC scores for each method on the MMCC dataset. As shown for class index 8 (e.g., Gatak), indicated by the light green line in Figure 4, **Self-MCKD** outperformed other methods achieving an AUC score exceeding 0.4. In Figure 5, we also visualized the T-SNE results for each method on the Malimg dataset. Compared to the baseline and other knowledge distillation methods, **Self-MCKD** produced more distinct and well-separated representations.

 doi:10.20944/preprints202501.2257.v2

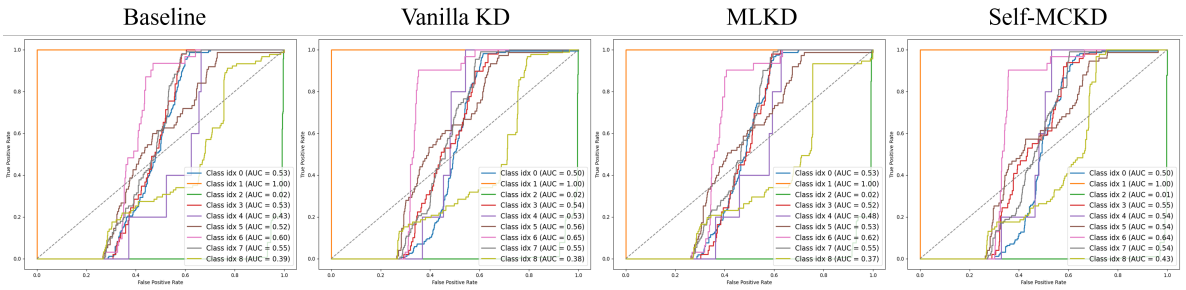

Figure 4. Comparison of ROC curves for knowledge distillation methods under MMCC dataset

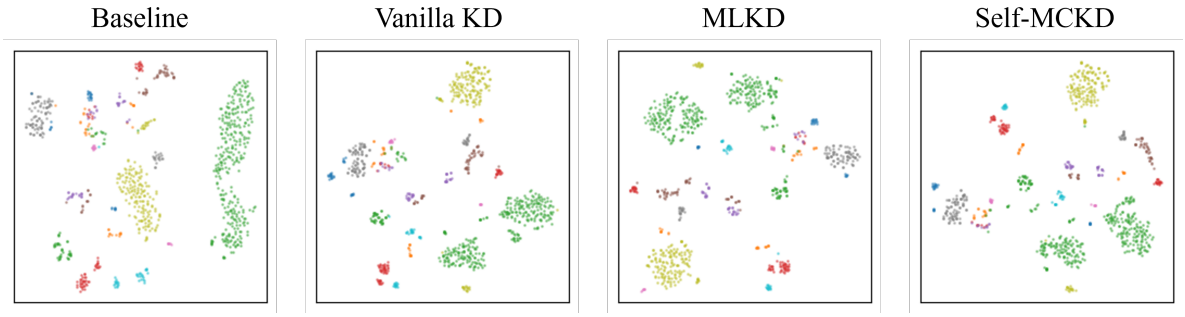

Figure 5. Comparsion of T-SNE visualizations for knowledge distillation methods under Malimg dataset

6. Conclusions

In this paper, we showed that traditional knowledge distillation in AI-based malware classification methods reduces the effectiveness of knowledge transfer by combining the knowledge of the target malware class and non-target classes. Also, this approach often relies on a high-performing, large and deep teacher model, which requires high computational cost during the training process, further reducing the efficiency of knowledge transfer. To address these problems, we proposed a novel knowledge distillation technique, called **Self-MCKD**. Its main idea is to transfer two types of knowledge, **NMKD** and **TMKD**, by separating the output logits of the target class and non-target classes. Also, **Self-MCKD** utilized a pre-trained small and shallow student model to serve as the teacher model. From the experimental results under various malware datasets, we showed that the separation of output logits in **Self-MCKD**, consisting of **NMKD** and **TMKD** increased the effectiveness of malware knowledge transfer. Specifically, **Self-MCKD** outperformed traditional knowledge distillation in all evaluation metrics. Furthermore, we showed that the small and shallow malware classification methods with **Self-MCKD** achieved better performance than the large and deep malware classification methods, while requiring only minimal additional computational cost. Also, we showed that **Self-MCKD** improved the efficiency of malware knowledge transfer by using small and shallow methods as both the teacher and student models. From such experimental results, we believe that **Self-MCKD** can enhance the malware knowledge in terms of the effectiveness and efficiency of malware classification better than existing knowledge distillation techniques. In future work, we will validate **Self-MCKD** under scenarios with highly imbalanced class distributions to assess its robustness and adaptability in real-world malware classification tasks.

Author Contributions: Methodology, data collection, analysis and interpretation of results and writing, H.J.Jeong, review and editing, H.J.Lee and G.N.Kim. All authors have read and agreed to the published version of the manuscript.

Acknowledgments: This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2024-RS-2023-00259967) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation) and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2023-00243075).

Conflicts of Interest: The authors declare no conflicts of interest.

 doi:10.20944/preprints202501.2257.v2

References

1. Nataraj, L.; Yegneswaran, V.; Porras, P.; Zhang, J. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*; ACM: New York, NY, USA, 2011; pp. 21–30.
2. Gibert, D. Convolutional neural networks for malware classification. *University Rovira i Virgili, Tarragona*; Spain, 2016.
3. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.D.B.; Wang, Y.; Iqbal, F. Malware classification with deep convolutional neural networks. In *Proceedings of the 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*; IEEE: Paris, France, 2018; pp. 1–5.
4. Prajapati, P.; Stamp, M. An empirical analysis of image-based learning techniques for malware classification. In *Malware Analysis Using Artificial Intelligence and Deep Learning*; Springer: Cham, Switzerland, 2021; pp. 411–435.
5. JIANG, K.; Bai, W.; Zhang, L.; CHEN, J.; PAN, Z.; GUO, S. Malicious code detection based on multi-channel image deep learning. *Journal of Computer Applications* **2021**, *41*, 1142.
6. Runzheng, W.; Jian, G.; Xin, T.; Mengqi, Y. Research on malicious code family classification combining attention mechanism. *Journal of Frontiers of Computer Science & Technology* **2021**, *15*, 881.
7. Liu, Z.; Sun, M.; Zhou, T.; Huang, G.; Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* **2018**.
8. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*; Chapman and Hall/CRC: Boca Raton, FL, USA, 2022; pp. 291–326.
9. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* **2015**.
10. Zhu, D.; Xi, T.; Jing, P.; Wu, D.; Xia, Q.; Zhang, Y. A transparent and multimodal malware detection method for android apps. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*; ACM: New York, NY, USA, 2019; pp. 51–60.
11. Zhi, Y.; Xi, N.; Liu, Y.; Hui, H. A lightweight android malware detection framework based on knowledge distillation. In *Network and System Security: 15th International Conference, NSS 2021, Tianjin, China, 23 October 2021*; Springer: Cham, Switzerland, 2021; pp. 116–130.
12. Xia, M.; Xu, Z.; Zhu, H. A Novel Knowledge Distillation Framework with Intermediate Loss for Android Malware Detection. In *Proceedings of the IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, Bangkok, Thailand, 2022; pp. 1–6.
13. Buciluǎ, C.; Caruana, R.; Niculescu-Mizil, A. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*; ACM: New York, NY, USA, 2006; pp. 535–541.
14. Zhao, B.; Cui, Q.; Song, R.; Qiu, Y.; Liang, J. Decoupled knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: New Orleans, LA, USA, 2022; pp. 11953–11962.
15. Yuan, L.; Tay, F.E.H.; Li, G.; Wang, T.; Feng, J. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: Seattle, WA, USA, 2020; pp. 3903–3911.
16. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* **2014**.
17. Cui, Z.; Xue, F.; Cai, X.; Cao, Y.; Wang, G.; Chen, J. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics* **2018**, *14*, 3187–3196.
18. Idika, N.; Mathur, A.P. A survey of malware detection techniques. *Purdue University* **2007**, *48*, 32–46.
19. You, I.; Yim, K. Malware obfuscation techniques: A brief survey. In *Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications*; IEEE: Fukuoka, Japan, 2010; pp. 297–300.
20. Hardy, W.; Chen, L.; Hou, S.; Ye, Y.; Li, X. DL4MD: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Science (ICDATA)*; The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp): Las Vegas, NV, USA, 2016.
21. Drew, J.; Moore, T.; Hahsler, M. Polymorphic malware detection using sequence classification methods. In *Proceedings of the 2016 IEEE Security and Privacy Workshops (SPW)*; IEEE: San Jose, CA, USA, 2016; pp. 81–87.

 doi:10.20944/preprints202501.2257.v2

22. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*; ACM: New Orleans, LA, USA, 2016; pp. 183–194.
23. Fathurrahman, A.; Bejo, A.; Ardiyanto, I. Lightweight convolution neural network for image-based malware classification on embedded systems. In *Proceedings of the 2021 International Seminar on Machine Learning, Optimization, and Data Science (ISMODE)*; IEEE: Bali, Indonesia, 2022; pp. 12–16.
24. Su, J.; Vasconcellos, D.V.; Prasad, S.; Sgandurra, D.; Feng, Y.; Sakurai, K. Lightweight classification of IoT malware based on image recognition. In *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*; IEEE: Tokyo, Japan, 2018; Volume 2, pp. 664–669.
25. Yim, J.; Joo, D.; Bae, J.; Kim, J. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*; IEEE: Honolulu, HI, USA, 2017; pp. 4133–4141.
26. Chen, P.; Liu, S.; Zhao, H.; Jia, J. Distilling knowledge via knowledge review. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: Virtual Conference, 2021; pp. 5008–5017.
27. Ronen, R.; Radu, M.; Feuerstein, C.; Yom-Tov, E.; Ahmadi, M. Microsoft malware classification challenge. *arXiv preprint arXiv:1802.10135* **2018**.
28. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B.S. Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*; ACM: Pittsburgh, PA, USA, 2011; pp. 1–7.
29. Kan, Z.; Wang, H.; Xu, G.; Guo, Y.; Chen, X. Towards light-weight deep learning based malware detection. In *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*; IEEE: Tokyo, Japan, 2018; Volume 1, pp. 600–609.
30. Moon, J.; Kim, S.; Song, J.; Kim, K. Study on machine learning techniques for malware classification and detection. *KSII Transactions on Internet & Information Systems* **2021**, *15*, 12.
31. Romero, A.; Ballas, N.; Kahou, S.E.; Chassang, A.; Gatta, C.; Bengio, Y. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* **2014**.
32. Wang, Z.; Li, Z.; He, D.; Chan, S. A lightweight approach for network intrusion detection in industrial cyber-physical systems based on knowledge distillation and deep metric learning. *Expert Systems with Applications* **2022**, *206*, 117671.
33. Wang, L.H.; Dai, Q.; Du, T.; Chen, L. Lightweight intrusion detection model based on CNN and knowledge distillation. *Applied Soft Computing* **2024**, *165*, 112118.
34. Shen, J.; Yang, W.; Chu, Z.; Fan, J.; Niyato, D.; Lam, K.Y. Effective intrusion detection in heterogeneous internet-of-things networks via ensemble knowledge distillation-based federated learning. *arXiv preprint arXiv:2401.11968* **2024**.
35. Huang.; Dong.; Li.; Tsz On.; Xie.; Xiaofei; Cui.; Heming. Themis: Automatic and Efficient Deep Learning System Testing with Strong Fault Detection Capability. In *2024 IEEE 35th International Symposium on Software Reliability Engineering(ISSRE)*; IEEE: 2024; pp. 451–462.
36. Jin, Y.; Wang, J.; Lin, D. Multi-level logit distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: Virtual Conference, 2023; pp. 24276–24285.