Adversarially Reprogramming Pretrained Neural Networks for Data-limited and Cost-efficient Malware Detection*

Lingwei Chen† Xiaoting Li‡ Dinghao Wu‡

Abstract

To mitigate evolving malware attacks, machine learning models have been successfully deployed to detect malware. However, these models are often challenged by data scarcity, design efforts and constrained resources. Inspired by the adversarial vulnerability of machine learning, in this paper, we design a novel model *Adv4Mal* to adversarially reprogram an ImageNet classification neural network for malware detection in both white-box and black-box settings. As such, a small or moderate amount of data are sufficient to train a promising malware detection model, the varying software features can be uniformly processed without extra efforts, and the majority of computation can be wisely shared and reused to save the resources. This, to the best of our knowledge, has not yet been explored. Specifically, Adv4Mal proceeds by embedding software features into a host image to construct new data, and learning a universal perturbation to be added to all inputs in an imperceptible manner, such that the outputs of the pretrained model can be accordingly mapped to the final detection decisions for all software. We evaluate Adv4Mal on three software datasets. The experimental results demonstrate that Adv4Mal can successfully exploit ImageNet model's learning capability and limited data to achieve high performance in malware detection, and also yield significant advantages of model flexibility to different features, and cost efficiency in computing resources.

1 Background and Motivation

Software over computers and smartphones plays a vital role in our daily lives [5]. Unfortunately, its ubiquity and benefits immensely attract attackers to disseminate malware onto unsuspecting users to deliberately fulfill their intents, such as unwanted advertising, and user tracing. According to a recent report, the total number of malware has surpassed 1.1 billion [3]. Given such a huge amount of malware, machine learning models have been successfully deployed

*To appear in *SIAM International Conference on Data Mining (SDM22)*.

†Department of Computer Science and Engineering, Wright State University, Dayton, OH 45435, USA.

‡College of Information Sciences and Technology, Pennsylvania State University, University Park, PA 16802, USA.

by many security companies to detect malware earlier than signature-based methods, and thus protect the legitimate users against the threats it poses [1, 16, 10, 33].

However, constructing and maintaining these models represents a significant challenge in terms of data scarcity, design expertise and constrained resources. On one hand, high-quality labeled malware samples are often scarce and costly to acquire, while learning models generally require a large dataset for training. On the other hand, models need to be continually advanced to fit in the evolving features on new malware and its variants as well. Furthermore, complicating model structure may drastically increase the consumption of computation and memory, causing malware detection services (especially for those deployed in clouds) very expensive. With this in mind, we may raise the following question: "can we build up a malware detection model with scarce training data and limited computing resources, which is also applicable to various software features without structure modification?"

Despite their remarkable inference ability, machine learning models themselves are vulnerable to adversarial attacks that can easily add small perturbations to the input data and cause the models to misclassify it with high confidence [15, 20]. Based on this fact, adversarial reprogramming [12] has been proposed to repurpose a machine learning model trained in a source domain to perform a target-domain task through learning a universal perturbation to the target-domain data without modifying the pretrained model parameters, where the domains and tasks can be completely different. For example, some recent works [22, 31, 6] effectively reprogrammed neural networks to solve text classification, medical image classification, and fraud detection tasks.

Inspired by these great successes, if we follow a similar way, malware detection problem can be reduced to a feasible adversarial reprogramming problem over well-trained neural networks, which, to the best of our knowledge, has not yet been explored. Unlike traditional adversarial attacks utilizing the linear approximations of neural networks to mislead the classifications, adversarial reprogramming completely depends on the nonlinear interactions of the input and the perturbation. In other words, given a deep neural network of nonlinear structure, a well-formulated offset added to its input would be sufficient on its own to repurpose the network to a new task without the need of model fine-tuning. Similarly, transfer learning has been also widely used to solve a target-domain task based on the knowledge transferred from a source-domain task [23]. However, it is achieved by fine-tuning a pretrained source-domain model with the target-domain data, which still requires a mass of training data to update model parameters. Training data scarcity thus remains an issue that is easily encountered in transfer learning.

As such, adversarial reprogramming paradigm yields the prospective advantages to address the aforementioned malware detection challenge. (1) Since there is no need to fine-tune the source-domain model and the perturbation is the only trainable parameter, a small or moderate amount of training data are sufficient to repurpose a source-domain model to detect malware. (2) Due to the source-domain model is software-feature-agnostic, adversarial reprogram-

ming can flexibly embed different software features through the uniform input layer without extra efforts for model structure modification. (3) It also enables the resources deployed to the source-domain model reusable for shared compute from different malware detection tasks and benefit the application scenarios with limited computing resources. Considering that ImageNet classification neural networks have been undergoing vibrant evolution in their nonlinear deep structures and learning capability, and a wide range of such pretrained models can be easily available for straightforward leverages, in this paper, we would like to investigate how ImageNet classification neural networks can be adversarially reprogrammed to detect malware in a learning-effective yet cost-efficient manner.

To this end, we present a novel model, called *Adv4Mal*, which **Adv**ersarially reprograms an ImageNet classification neural network to perform **Mal**ware detection task. Given a pretrained model of this kind and a host image randomly selected from ImageNet, Adv4Mal proceeds by injecting the features extracted from software into the host image to construct new image dataset, and learning a universal perturbation to be added to all inputs, such that the outputs of the pretrained model can be accordingly mapped to the final detection decisions (i.e., benign or malicious) for all software. This leads to the following three major goals for Adv4Mal: (1) input transformation to embed software features and perturbation into the host image for the adversarial task, (2) output transformation to map ImageNet classes to malware detection classes, and (3) either white-box or black-box optimization to learn the universal perturbation corresponding to the model scenarios of complete or limited access. In summary, this paper has the following major contributions:

- We explore a novel and practical perspective of malware detection, where we leverage adversarial reprogramming of high-performance and nonlinear deep neural networks trained in ImageNet classification domain to perform malware detection task with limited training software data, software-feature-agnostic model, and less consumption of computing resources.

- We consider a general application scenario, where Adv4Mal (1) conceals the visibility of software features and perturbation to more stealthily reprogram a potential model deployed by self or others, and (2) implements adversarial reprogramming in both white-box and black-box settings with different model access levels. Thus, we propose a black-box optimization to facilitate perturbation formulation. We also investigate different output mapping methods.

- We conduct extensive experiments on three software datasets, including Windows PE files, Android apps, and Drebin apps, to evaluate Adv4Mal on malware detection effectiveness and cost efficiency.

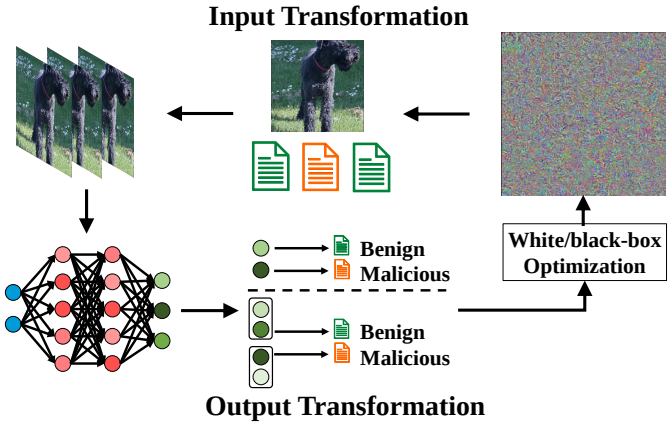

Figure 1: The overview of Adv4Mal.

2 Methods and Technical Solutions

2.1 Problem Statement

We consider that we have either complete or limited access to a deployed ImageNet classification neural network, and use white-box or black-box adversarial reprogramming of this model for malware detection, so as to reduce the cost of developing and maintaining malware detection systems. The source-domain task for our application is ImageNet classification, and the target-domain task is malware detection. Accordingly, we define $\mathbf{x}$ to be software features, $t(\mathbf{x})$ a malware detection function, $\tilde{\mathbf{X}}$ ImageNet images, and $s(\tilde{\mathbf{X}})$ a pretrained ImageNet classification model (e.g., DenseNet [18], Google Cloud Vision API). Based on these definitions, the input transformation function $h_s(\cdot;\boldsymbol{\theta})$ comprises embedding $\mathbf{x}$ to a host image to formulate new data $\tilde{\mathbf{X}}$, and learning a universal perturbation $\boldsymbol{\theta}$ added to $\tilde{\mathbf{X}}$ such that $\tilde{\mathbf{X}} = h_s(\mathbf{x};\boldsymbol{\theta})$; the output transformation function $h_t(\cdot)$ further maps ImageNet classes to malware detection classes such that $t(\mathbf{x}) = h_t(s(\tilde{\mathbf{X}}))$. Given a set of software training samples, Adv4Mal proceeds with input and output transformations, while the only trainable perturbation $\boldsymbol{\theta}$ is updated through evaluating the difference between the output and the corresponding input's ground truth using white-box or black-box optimization. The overview of Adv4Mal is illustrated in Figure 1.

2.2 Motivation on Adversarial Reprogramming

Adversarial reprogramming is effected on the nonlinear interactions of the input $\mathbf{x}$ and the perturbation $\boldsymbol{\theta}$, which can be satisfied by an ImageNet classification neural network of nonlinear deep structure [25]. Specifically, let an ImageNet classification model $s(\tilde{\mathbf{X}})$ receive an input $\tilde{\mathbf{X}}$, which is transformed from a software $\mathbf{x}$ with the perturbation $\boldsymbol{\theta}$ such that the input $\tilde{\mathbf{X}}$ can be rewrit-

ten as $\tilde{\mathbf{X}} = [\mathbf{x}, \boldsymbol{\theta}]$, and let the weights of $s(\tilde{\mathbf{X}})$ be partitioned into two sets $\tilde{\mathbf{W}} = [\mathbf{w}_{\mathbf{x}}, \mathbf{w}_{\boldsymbol{\theta}}]$. The output of ImageNet classification model can be approximated as

$$\begin{aligned} s(\tilde{\mathbf{X}}) &= g(\tilde{\mathbf{W}}^T \tilde{\mathbf{X}}) = g(\mathbf{w}_{\mathbf{x}}^T \mathbf{x} + \mathbf{w}_{\boldsymbol{\theta}}^T \boldsymbol{\theta}) \\ &= g_1(\mathbf{w}_{\mathbf{x}}^T \mathbf{x}) + g_2(\mathbf{w}_{\boldsymbol{\theta}}^T \boldsymbol{\theta}) + g_3(\boldsymbol{\theta}^T \mathbf{x}) \end{aligned} \tag{1}$$

where $g(\cdot)$, $g_1(\cdot)$, $g_2(\cdot)$, and $g_3(\cdot)$ are abstracted functions here for the sake of simplicity that can be decided by the model's nonlinear structure and activation functions. Eq. (1) implies that the perturbation $\boldsymbol{\theta}$ can adapt not only the effective bias $\mathbf{w}_{\boldsymbol{\theta}}^T \boldsymbol{\theta}$, but also the weights applied to the input $\mathbf{x}$ through $\boldsymbol{\theta}^T \mathbf{x}$ without even touching the original weights $\mathbf{w}_{\mathbf{x}}$ to change the way the input is processed. Therefore, an additive perturbation to an ImageNet model's input would be sufficient on its own to repurpose the network to output the results for malware detection task [12]. When revisit transfer learning [23, 9], we can observe that it is necessary to fine-tune the pretrained model to transfer the knowledge to the target task. Differently, adversarial reprogramming keeps all model parameters unchanged and solely learns the perturbation $\boldsymbol{\theta}$ to perform the adversarial tasks. From the application point of view, transfer learning is not as feasible as adversarial reprogramming, since sometimes we may not have sufficient labeled training data for model fine-tuning, and may not have complete access to the pretrained models to tamper with parameters and intermediate layers.

The above theoretical analysis indicates that adversarial reprogramming of ImageNet neural networks can benefit our malware detection problem in four aspects. (1) Only the perturbation $\boldsymbol{\theta}$ is trainable that takes less training efforts regarding labeled data and time cost than transfer learning and other models trained from scratch. (2) As the input format of $s(\tilde{\mathbf{X}})$ is fixed while software features $\mathbf{x}$ always need to be transformed to $\tilde{\mathbf{X}}$, $s(\tilde{\mathbf{X}})$ is software-feature-agnostic and applicable to varying malware detection tasks without impact on model structure and computes. (3) ImageNet neural networks with pretrained $\mathbf{w}_{\mathbf{x}}$ and $\mathbf{w}_{\boldsymbol{\theta}}$ provide powerful learning capability to extract expressive patterns from subtle software features and achieve high performance in malware detection. (4) Since its original weights are unchanged, ImageNet model can be shared for major computes, and the resources deployed to it are thus reusable for different malware detection tasks whose individual computations are less-costly data transformations.

2.3 Input Transformation

Input transformation is to convert the software data to the input space of ImageNet classification. Normally, feature distribution and perturbation magnitude need not be constrained for adversarial reprogramming to work. However, the idea is more general here: Adv4Mal could repurpose any ImageNet model deployed by self or others, while those provided by others may be aware of potential attacks to avoid computing resource abuse. Thus, we would like to construct

our imperceptible reprogramming attack, such that the visibility of software features and perturbation are concealed to the deployed models.

Software feature embedding. Without loss of generality, each given software $\mathbf{x} \in \mathcal{X}$ can be represented by a k-dimensional feature vector $\mathbf{x} = \langle x_1, x_2, x_3, \cdots, x_k \rangle$, where the value of x_i depends on the possible features derived from software. For example, API calls can effectively reflect the behaviors of software [5, 7], which have been thus often extracted from Windows PE files or Android apps to represent them. In this case, $x_i = \{0, 1\}$ indicting the absence or presence of API call i in software $\mathbf{x}$. In addition, software can be also represented using representation learning methods [24, 2]. Considering that each pixel value for ImageNet images is initially scaled to $[0, 1]$ before being further transformed, we need to normalize x_i as $0 \leq x_i \leq 1$ to perform the software feature embedding.

To conceal their visibility, we embed software features in a host image $\tilde{\mathbf{X}} \in \mathbb{R}^{n \times n \times 3}$ randomly selected from ImageNet in a dispersed manner. Generally, high-frequency areas in images with sharp transients may more significantly enhance the invisibility of features embedded in the host image than low-frequency areas. We hence introduce Discrete Wavelet Transform (DWT) [29] to mark high-frequency areas by dividing each host image channel into two quadrants:

$$c\mathbf{A}_i, (c\mathbf{H}_i, c\mathbf{V}_i, c\mathbf{D}_i) = \mathrm{dwt}(\tilde{\mathbf{X}}_i), \quad i \in \{0, 1, 2\} \tag{2}$$

where $c\mathbf{A}_i$ is low-frequency quadrant and $c\mathbf{H}_i$, $c\mathbf{V}_i$, $c\mathbf{D}_i$ are high-frequency quadrants. After that, we zero $c\mathbf{A}_i$ to hide low-frequency quadrant, perform inverse DWT over all quadrants, and normalize it to the range 0 to 1:

$$\mathbf{C}_i = \mathrm{Norm}(\mathrm{idwt}(c\mathbf{A}_i * 0, (c\mathbf{H}_i, c\mathbf{V}_i, c\mathbf{D}_i))) \tag{3}$$

such that we can approximately mark those high-frequency areas in the spatial domain as $\mathbf{C} \in \mathbb{R}^{n \times n \times 3}$ by zeroing out the pixels of value less than 0.5. Some examples are displayed in Figure 2(b)-(d) obtained from host images in Fig. 2(a). Given $\mathbf{C}$, k pixels are then randomly selected from $\tilde{\mathbf{X}}$ to store k feature values from $\mathbf{x}$:

$$\tilde{\mathbf{X}} = \tilde{\mathbf{X}}^k_{\mathbf{C}} \oplus \mathbf{x} \tag{4}$$

where $\oplus$ is feature embedding operation. A mask $\mathbf{M} \in \mathbb{R}^{n \times n \times 3}$ is accordingly formulated with 0 for k locations of the embedded features and 1 for others to avoid software features being perturbed. Once k pixels are designated, pixel i will be firmly associated with the same feature x_i for all software samples. Some feature embedding examples using binary API calls from an Android app are shown in Figure 2(e), where software features can be hardly perceived in host images.

Perturbation formulation. The perturbation in adversarial reprogramming is formulated to be added to all inputs [12], which can be defined as:

$$\tilde{\boldsymbol{\theta}} = \epsilon \cdot \tanh(\boldsymbol{\theta} \odot \mathbf{M}) \tag{5}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{n \times n \times 3}$ is the universal perturbation to be learned, $\mathbf{M}$ is a mask as defined above, $\odot$ denotes the element-wise product, and $\tanh(\cdot)$ bounds the

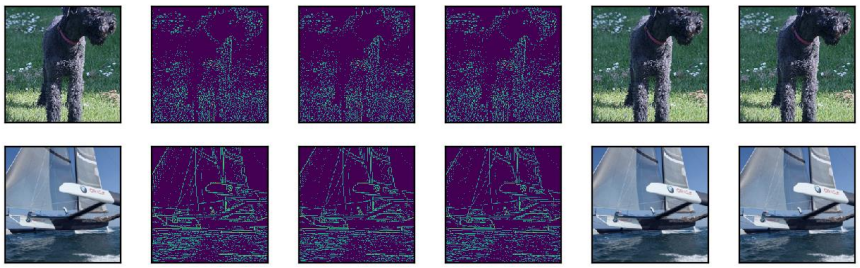

Figure 2: (a) host images, (b-d) high-frequency areas marked in R-G-B channels, (e) input with software features, (f) input with perturbation.

perturbation to be in $(-1, 1)$. Here we also introduce an adjustable hyper-parameter ϵ to govern the magnitude of the perturbation. The transformed new input data sample can thus be presented as:

$$h_s(\mathbf{x}; \boldsymbol{\theta}) = \text{clip}(\tilde{\mathbf{X}} + \epsilon \cdot \tanh(\boldsymbol{\theta} \odot \mathbf{M})) \tag{6}$$

where $\text{clip}(\cdot)$ performs per-pixel clipping of the image to limit each pixel value to the range of $[0, 1]$. In our method, both ϵ and $\text{clip}(\cdot)$ contribute to the perturbation imperceptibility. Some perturbed image examples are shown in Figure 2(f) with $\epsilon = 0.3$, where the perturbations are imperceptible.

2.4 Output Transformation

Output transformation is to map ImageNet classes back to malware detection classes. For the new input $\tilde{\mathbf{X}} = h_s(\mathbf{x}; \boldsymbol{\theta})$, the ImageNet model outputs $\tilde{y} = s(\tilde{\mathbf{X}})$ with $\tilde{y} \in \{0, 1, \cdots, 999\}$, while for the software $\mathbf{x}$, the malware detection function outputs $y = t(\mathbf{x})$ with $y \in \{0, 1\}$. To avoid extra training effort, we leverage non-parametric mapping methods to formulate output transformation function $h_t(\cdot)$ by using either hard coded mapping or soft coded mapping method. Specifically, hard coded mapping simply assigns two randomly-selected class outputs of ImageNet to predict benign and malicious classes respectively:

$$h_t(\tilde{y}) = \langle \tilde{y}_i, \tilde{y}_j \rangle, \quad i \neq j. \tag{7}$$

Soft coded mapping forms two class sets of ImageNet with the same number of non-repeated classes each, and averages the outputs for both sets, which are used to report malware detection decisions respectively:

$$h_t(\tilde{y}) = \langle \frac{1}{|I|} \sum_{i \in I} \tilde{y}_i, \frac{1}{|J|} \sum_{j \in J} \tilde{y}_j \rangle, \quad |I| = |J| \tag{8}$$

where I and J are two non-repeated index sets used to locate the ImageNet classes.

2.5 Optimization

White-box setting. To achieve our cost-efficient malware detection goal, a more feasible solution is to reprogram self-deployed ImageNet models for complete access without extra expenses on large model query. The optimization of Adv4Mal can be directly formulated as:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}(-\log p(h_t(\tilde{y})|h_s(\mathbf{x};\boldsymbol{\theta})) + \lambda\|\boldsymbol{\theta}\|_F^2) \tag{9}$$

where $p(h_t(\tilde{y})|h_s(\mathbf{x};\boldsymbol{\theta}))$ indicates the probability that an image input $\tilde{\mathbf{X}}$ transformed from software $\mathbf{x}$ being classified as $\tilde{y}$, which can be mapped to malware; λ is the regularization parameter. Eq. (9) shows that $\boldsymbol{\theta}$ is the only parameter for our problem, which can be updated by optimizing this loss using Adam.

Black-box setting. For the adversarial reprogramming scenarios that Adv4Mal attempts to repurpose models deployed by others (e.g., online image classification APIs or public facing services), only input-output responses can be queried and retrieved without knowing the model architecture. As such, optimizing the loss function in Eq. (9) to compute the gradient via back-propagation is infeasible, since $\nabla_{\boldsymbol{\theta}} s(h_s(\mathbf{x};\boldsymbol{\theta}))$ is inadmissible. To update the perturbation $\boldsymbol{\theta}$, it is necessary to use black-box optimization techniques to enable the model training. Recent examples have demonstrated the effectiveness of gradient-free optimizations [26, 21, 19, 8]. As a particular set, Evolution Strategies have exhibited very high parallelizability and good exploration behaviors [4, 27].

To this end, in this paper, following the Evolution Strategies, we propose a black-box optimization to facilitate our problem. More specifically, let $L(\boldsymbol{\theta})$ be the objective function acting on perturbation $\boldsymbol{\theta}$, and $p_\psi(\boldsymbol{\theta})$ be the population distribution. Our black-box optimization proceeds to maximize the average objective value $\mathbb{E}_{\boldsymbol{\theta}\sim p_\psi} L(\boldsymbol{\theta})$ over the population by searching for ψ with stochastic gradient ascent [26]. Accordingly, based on the definitions, we can instantiate the objective function $L(\boldsymbol{\theta})$ as the classification accuracy of malware detection, and the population distribution p_ψ as an isotropic multivariate Gaussian with mean ψ and fixed covariance $\sigma^2 I$, such that $\mathbb{E}_{\boldsymbol{\theta}\sim p_\psi} L(\boldsymbol{\theta})$ can be rewritten as

$$\mathbb{E}_{\boldsymbol{\theta}\sim p_\psi} L(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\rho}\sim N(0,I)} L(\boldsymbol{\theta} + \sigma\boldsymbol{\rho}) \tag{10}$$

In this way, the original objective $L(\boldsymbol{\theta})$ can be cast as a Gaussian-blurred version $L(\boldsymbol{\theta} + \sigma\boldsymbol{\rho})$. To perform the optimization of $\mathbb{E}_{\boldsymbol{\rho}\sim N(0,I)} L(\boldsymbol{\theta} + \sigma\boldsymbol{\rho})$ over $\boldsymbol{\theta}$ using stochastic gradient ascent, we further deploy REINFORCE algorithm [32] to formulate the new gradient estimator:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{\rho}\sim N(0,I)} L(\boldsymbol{\theta} + \sigma\boldsymbol{\rho}) &= \frac{1}{\sigma} \mathbb{E}_{\boldsymbol{\rho}\sim N(0,I)} \{L(\boldsymbol{\theta} + \sigma\boldsymbol{\rho})\boldsymbol{\rho}\} \\ &= \frac{1}{n\sigma} \sum_{i=1}^{n} L(\boldsymbol{\theta} + \sigma\boldsymbol{\rho}_i)\boldsymbol{\rho}_i \end{aligned} \tag{11}$$

Eq. (11) demonstrates four repeated executions for this optimization: (1) performing n stochastic manipulations on perturbation $\boldsymbol{\theta}$, (2) evaluating malware

Table 1: Statistics of software datasets

Dataset	#Total	#Ben	#Mal	#APIs	#Perms	#Ints
Windows PEs	10,000	5,000	5,000	3,503	-	-
Android apps	7,670	4,367	3,303	329	104	204
Drebin apps	15,036	9,476	5,560	215	-	-

detection accuracy for each manipulation $L(\boldsymbol{\theta} + \sigma\boldsymbol{\rho}_i)$, (3) estimating gradient based on these n manipulations and evaluation results, and then (4) updating $\boldsymbol{\theta}$ using the estimated gradient. Here we take advantage of mirrored sampling [14] to reduce the variance by manipulating $\boldsymbol{\theta}$ with $(\boldsymbol{\rho}, -\boldsymbol{\rho})$ pairs, such that the impact of outliers on each population can be mitigated and the possibility to fall into local optima in training may also be decreased.

As the host image $\tilde{\mathbf{X}}$ and software features $\mathbf{x}$ are constants during the reprogramming, we can pre-process feature embedding on $\tilde{\mathbf{X}}$, which significantly decreases the training time cost. Also, the only trainable parameter is the $n \times n \times 3$ perturbation tensor $\boldsymbol{\theta}$; therefore, the complexity of Adv4Mal is consistent for all different software features and different malware detection tasks. Based on the trained model, we can easily perform malware detection that needs only input and output transformations, while leaving the majority of computes to the deployed ImageNet models.

3 Empirical Evaluation

3.1 Experimental Setup

Datasets. We test Adv4Mal on three software datasets: Windows PE files [7], Android apps [5], and Drebin apps [34]. The dataset statistics are summarized in Table 1. Each Windows PE file is represented as $3,503$-dimensional API call vector; each Android app is represented as 329-dimensional API call vector, 104-dimensional permission vector, and 204-dimensional filtered intents respectively; and each Drebin app is represented as 215-dimensional API call vector. We mainly use API calls to evaluate malware detection effectiveness and parameters on PE files and Android apps, while Drebin apps are included in baseline studies, and the comparison among API calls, permissions, and filtered intents is used for cost efficiency evaluation.

Parameter setting. We randomly select 80% of the samples for training, while the remaining 20% is used for testing, and we report the mean accuracy and F1-score of 5 runs on test samples for all the experiments. The training setting is specified as 0.01 initial learning rate, and 5×10^{-4} L2 regularization on the perturbation. We also set $\sigma = 0.01$ and $n = 50$ for black-box optimization.

Pretrained ImageNet models. For adversarial reprogramming, consider-

ing the query expenses on online image classification services, we employ five pretrained ImageNet models for white-box and black-box settings, including DenseNet-121 and DenseNet-161 [18], ResNet-50 and ResNet-101 [17], and Inception-V3 [30].

Baselines. We compare Adv4Mal with some other malware detection models from recent works: linear model from Demontis et al. [11], ensemble on base linear models from Chen et al. [5] which is similar to DroidFusion [34], stacked auto-encoder (SAE) from Hardy et al. [16] with structure of $[k, 128, 64, 10, 2]$, convolutional neural network (CNN) from Demetrio et al. [10] with structure of $[(1, 6, 3), (6, 16, 3), 64, 10, 2]$ (since we do not have raw bytes of the files, we convert software features to 60×60, 18×18, and 16×16 matrices for PEs, Android and Drebin apps as inputs respectively), long short-term memory (LSTM) from Athiwaratkun et al. [2] (we only feed one-hot features to each unit in the feature number order), and graph convolutional network (GCN) from Garcia et al. [13] with two-layer structure over graphs built on software feature differences for all samples. To compare Adv4Mal with models of the comparable amount of parameters, we also include the aforementioned five ImageNet neural networks as baselines, where the output of each network is replaced with a linear layer for binary classification and the networks are fine-tuned through transfer learning. Note that, due to the data limitation, other state-of-the-art models (e.g., [1, 33]) interacting with unavailable features (e.g., hardware or relation features) are not included here.

3.2 Ablation Study on Output Transformation

We first perform an ablation study to decide the mapping method that better benefits Adv4Mal. For hard coded mapping, we randomly choose two ImageNet classes, while for soft coded mapping, we randomly divide 1,000 ImageNet classes into two sets. Based on this setting, we evaluate Adv4Mal using ResNet-101 with $\epsilon = 0.3$ and white-box optimization. Figure 3 shows the comparative results. We can see that hard coded mapping slightly outperforms soft coded mapping on test accuracy and F1-score. When we look into the training procedure, we observe that hard coded mapping reaches a stable loss at epoch 8, while soft coded mapping requires 12–15 epochs to achieve the comparative test performance, which enforces higher training time. Therefore, hard coded mapping leads to faster and better convergence results, and we use it in our model Adv4Mal for the further evaluation.

3.3 Evaluation of Adv4Mal

Malware detection effectiveness. In our experiments, we evaluate Adv4Mal over different perturbation magnitudes ϵ and host images $\tilde{\mathbf{X}}$ for Windows PE files and Android apps under white-box setting Note that, we mainly use white-box setting for parameter evaluations, while the comparison between white-box

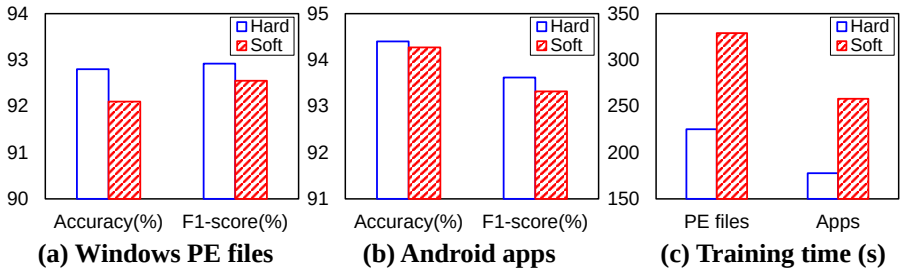

Figure 3: Test performance and training time between hard and soft coded mappings.

and black-box optimizations will be further discussed in the following subsection. In particular, we test Adv4Mal for malware detection with $\epsilon \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$, while the host images $\tilde{\mathbf{X}}$ being randomly selected from ImageNet data, including kerry blue terrier, toucan, home theater, cellphone, and trimaran. The results are shown in Figure 4. As we can see, though different parameter settings contribute to different test results, which will be further discussed later, Adv4Mal successfully achieves the goal of reprogramming five well-trained ImageNet classification models to perform malware detection. Averagely, Adv4Mal obtains 93.75% test accuracy and 93.78% F1-score on Windows PEs, and 94.45% accuracy and 93.56% F1-score on Android apps.

Impact of perturbation magnitudes (ϵ). We empirically investigate the sensitivity of perturbation magnitude ϵ on the performance of Adv4Mal, which is illustrated in Figure 4(a)-(b), and Fig. 4(d)-(e) (note that, all these experiments are run on a host image of kerry blue terrier). Generally, when we enlarge ϵ, the test effectiveness increases. We can observe that the accuracy and F1-score rise to a stable high level at $\epsilon = 0.3$ for all five pretrained neural networks on both datasets and then either slightly increase or drop when ϵ changes from 0.3 to 0.5. On the other hand, the larger ϵ may increase the risk of perturbation being recognized by the pretrained models. In this respect, we use $\epsilon = 0.3$ throughout the following evaluations to keep a good trade-off between the malware detection effectiveness and perturbation imperceptibility for adversarial reprogramming.

Impact of host images ($\tilde{\mathbf{X}}$). It is also interesting to see if different host images affect the performance of our model Adv4Mal. Since Inception-V3 outperforms other pretrained models at $\epsilon = 0.3$, we report the test accuracy and F1-score of Adv4Mal that repurposes Inception-V3 for malware detection over five different host images selected from ImageNet. As shown in Figure 4(c) and Figure 4(f), we find that the evaluation results slightly vary in different host images. However, the standard deviations of accuracy and F1-score on both datasets fall within $[0.001, 0.003]$, which imply that the performance difference is not statistically significant and Adv4Mal is loosely coupled with host images.

Impact of optimizations. To assess the impact of black-box setting on the detection performance, we assume that we are not aware of the ImageNet model

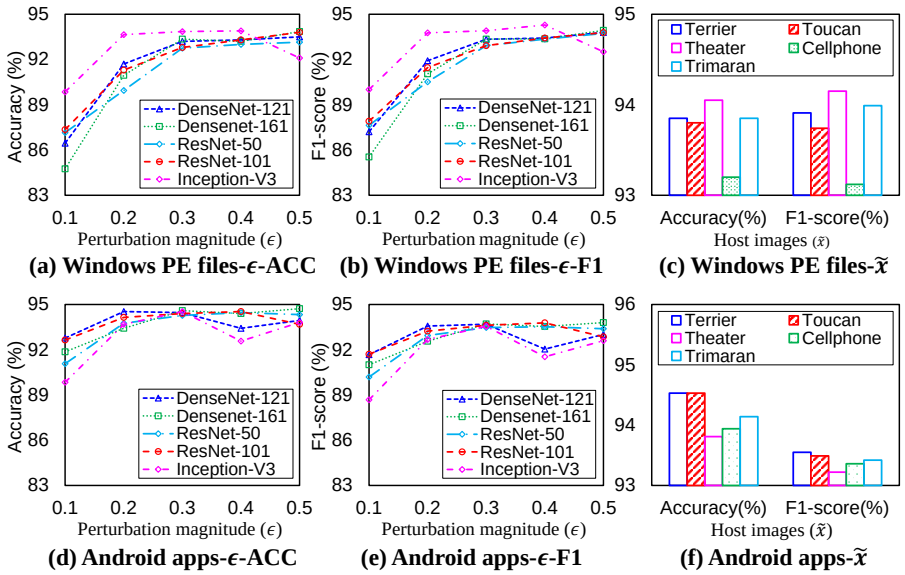

Figure 4: Evaluation on ϵ and host images (white-box): (a), (b), (c) specify the test results on Windows PE files, while (d), (e), (f) give the results on Android apps.

architecture, but query the models with inputs and retrieve the output predictions to facilitate our gradient-free optimization. The results are illustrated in Table 2 (lower table). We can observe that black-box reprogramming still manages to averagely achieve highly promising 91.60% accuracy and 91.83% F1-score on Windows PE files, 92.64% accuracy and 91.70% F1-score on Android apps, and 95.00% accuracy and 91.89% F1-score on Drebin apps. Clearly, white-box reprogramming outperforms the black-box approach in all of our experiments. As will be explained in Section 3.5, black-box optimization takes more training epochs (20-25) to obtain the effective perturbation than white-box (8-10), which results in much higher training time. For our cost efficiency goal, white-box setting enables a more feasible solution than black-box setting for convenient resource self-deployment, complete and free model access, and better performance in practice. But we believe that scaling up black-box optimization to many parallel workers can achieve linear speedups, and further advancing gradient-free optimization techniques can make the training procedure more stable and feasible with less variance. As leveraging online/public services can significantly reduce the self-deployed resources, we would like to leave more investigations on black-box optimization as our future work.

Table 2: Comparisons of different machine learning-based malware detection models

Method	Pretrained	Windows PE files		Android apps		Drebin apps	
		ACC (%)	F1 (%)	ACC (%)	F1 (%)	ACC (%)	F1 (%)
Linear (Demontis et al. [11])	-	87.65	87.61	91.22	90.25	92.48	88.53
Ensemble (Chen et al. [5])	-	89.35	89.41	91.85	90.96	93.67	90.35
SAE (Hardy et al. [16])	-	91.40	91.64	92.62	91.48	94.85	91.67
CNN (Demetrio et al. [10])	-	91.05	91.13	92.36	91.47	94.55	91.15
LSTM (Athiwaratkun et al. [2])	-	88.35	88.42	90.88	90.12	90.56	87.20
GCN (Garcia et al. [13])	-	89.85	89.96	91.45	90.93	94.10	91.30
DenseNet-121	-	89.80	89.85	88.98	88.03	91.63	88.31
DenseNet-161	-	89.60	89.67	88.13	87.35	90.88	87.16
ResNet-50	-	89.85	89.87	89.05	88.21	91.82	88.74
ResNet-101	-	89.55	89.66	88.76	87.83	90.26	87.65
Inception-V3	-	89.70	89.83	87.95	87.17	90.79	87.15
Adv4Mal (White-box)	DenseNet-121	93.20	93.34	94.46	93.69	96.50	93.41
	DenseNet-161	92.35	92.35	**94.59**	**93.71**	97.70	94.37
	ResNet-50	92.75	92.92	94.27	93.47	96.82	93.59
	ResNet-101	92.80	92.92	94.40	93.62	96.90	93.70
	Inception-V3	**93.85**	**93.91**	94.53	93.55	**97.81**	**94.56**
Adv4Mal (Black-box)	DenseNet-121	91.65	91.87	92.77	91.67	94.52	91.75
	DenseNet-161	91.85	91.93	92.86	92.25	95.98	92.70
	ResNet-50	90.95	91.52	92.07	91.19	94.08	91.12
	ResNet-101	91.30	91.46	93.64	92.70	94.56	91.33
	Inception-V3	92.25	92.37	91.84	90.68	95.90	92.56

3.4 Comparisons with Baselines

Comparisons with regular models. We compare Adv4Mal with other baselines that can directly feed feature vectors for malware detection, including linear model [11], ensemble model [5], SAE [16], CNN [10], LSTM [2], and GCN [13]. The comparative results are illustrated in Table 2. We can observe that compared to linear and ensemble models, SAE has greatly boosted the test performance; whereas, CNN, LSTM and GCN, as other types of neural networks, yield less advantage from deep structure to detect malware due to the facts that (1) converting feature vectors to matrices enforces either information padding (for PE files and Drebin apps) or lost (for Android apps), which degrades the feature expressiveness learned by convolutions; (2) features without sequential relations and graphs built on feature vectors appear to introduce some noisy information. Obviously, Adv4Mal leverages the powerful feature extraction capability of ImageNet classifiers, whose structures are more sophisticated and much deeper than regular SAEs and CNNs, to achieve state-of-the-art accuracy and F1-score for malware detection. Most of black-box methods enjoy either comparative or better performance than baselines (except for SAE and CNN). By contrast, white-box models on either Inception-V3 or DenseNet-161 obtain the best results on Windows PEs, Android and Drebin apps respectively, while five pretrained models reprogrammed by Adv4Mal under the same setting all outperform other baselines.

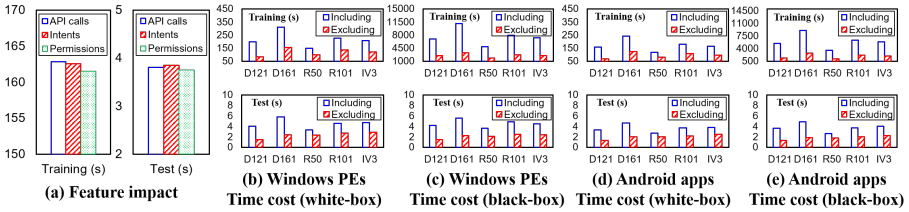

Figure 5: Evaluation of cost efficiency: (a) time cost on different features, (b) and (c) time cost for Window PE files in white/black-box settings, and (d) and (e) time cost for Android apps in white/black-box settings.

Table 3: Training time (transfer learning and Adv4Mal)

Method	Model	PEs (s)	Android (s)	Drebin (s)
Transfer Learning	DenseNet-121	545	431	830
	DenseNet-161	912	708	1,379
	ResNet-50	380	311	590
	ResNet-101	636	495	963
	Inception-V3	550	443	847
Adv4Mal (White)	DenseNet-121	199	156	303
	DenseNet-161	308	242	468
	ResNet-50	147	116	224
	ResNet-101	225	177	342
	Inception-V3	207	162	314

It is worth remarking that we can surely increase the parameters of regular SAE and CNN to a similar number that ImageNet neural networks have and further improve their malware detection performance. However, the increase of parameters will come with the huge growth of computations and resource consumption as well, since they are trained from scratch and all parameters need to be updated. Differently, the only trainable parameter for Adv4Mal is $224 \times 224 \times 3$ perturbation tensor, which is consistent to all inputs and tasks, while its training cost is comparable to regular SAE and CNN used in baselines.

Comparisons with transfer learning models. As aforementioned, we would like to see if we directly train a deep neural network with similar structure to the pretrained model reprogrammed by Adv4Mal for malware detection, how the performance difference will be. To this end, we replace the output of five ImageNet neural networks with a linear layer for binary classification, and fine-tune these models using transfer learning. All the parameters for these networks are updated. The malware detection results are shown in Table 2. Surprisingly, though the structure of these models are much deeper than regular SAE and CNN, their accuracy and F1-score perform apparently worse, which are not even close to Adv4Mal. Due to the large amount of parameters to update for trans-

fer learning, limited data might be the main reason for these underperforming results. This indicates that Adv4Mal is applicable to the application scenario with limited training data. In addition, as demonstrated in Table 3, the burdensome model fine-tuning operation also enforces 2.5-3 times higher training cost for transfer learning than Adv4Mal. By contrast, Adv4Mal merely updates a single perturbation tensor and directly leverages the learning power of the pretrained model, such that it is less influenced by the limited data setting, and enjoys better detection performance and less training cost.

3.5 Evaluation of Cost Efficiency

In addition to less training cost and less training data demand than transfer learning, here we demonstrate the practicality of Adv4Mal for malware detection with cost efficiency in model design and computing resources. We evaluate Adv4Mal using Inception-V3 on Android apps with three completely different feature representations: 329-dimensional API calls, 204-dimensional filtered intents, and 104-dimensional permissions, and also report computing time of Adv4Mal including/excluding the consumption by the ImageNet neural networks. All the results are illustrated in Figure 5.

Model flexibility to different features. From Figure 5(a), we can see that different from the models constructed from scratch or transfer learning that require more training and test time as the feature dimension increases, the time cost of Adv4Mal is consistent to features of different types and dimensions: the training time on API calls, filtered intents, and permissions is ranging in $(161.5, 162.8)$s and the test time is varying in $(3.7, 3.8)$s. This implies that Adv4Mal is software-feature-agnostic and flexible to different software features without imposing extra time cost and design effort, which yields an advantage of applicability and scalability for different malware detection scenarios.

Training and test time costs. Figure 5(b)-(e) show that black-box training requires much higher time cost than white-box training due to its large model query and gradient exploration. In both settings, when we exclude the time cost consumed by the ImageNet models, all the costs enjoy a drastic drop. For example, the training time for Adv4Mal using DenseNet121 on PE files is 199.0s: 116.5s are consumed by DenseNet121 while only 82.5s are used on data transformation and perturbation update; under the same data setting, regular SAE and transfer learning on DenseNet121 cost 71.1s and 545.0s for model training. This implies that the major cost of Adv4Mal is spent on shared computes in ImageNet models; also, Adv4Mal costs the comparable time to regular networks and much less than transfer learning. Adv4Mal yields another advantage to enable reusable resources, reduce the computing cost for individual malware detection tasks, and thus address the real-world challenge for such models with limited resources.

4 Significant Impact and Applicability

Our solution impacts on malware detection as follows: (1) Adv4Mal leverages the powerful learning ability of ImageNet neural networks, which enjoys better performance than a malware detection model constructed from scratch, and more potentials than transfer learning for less training data and cost; (2) Adv4Mal provides scalable application interfaces and flexible resource managements for different malware detection tasks, as they need only store and update the data, and enforce major computation to the ImageNet models.

In this respect, Adv4Mal holds the applicability of reprogramming high-performance neural networks to perform malware detection tasks. Similar to the recent work that focused on building dynamically connected networks with reusable components [28], in practice, we may consider the ImageNet models as our reusable components to perform the shared compute disentangled out of different tasks, while these components can be self-deployed in local or cloud servers for complete and easy access, or reached through pay-per-use online services. This especially benefits the scenarios with many tasks yet limited resources. Though we demonstrate adversarial reprogramming of image-domain neural networks on malware detection, both source-domain models and target-domain tasks can be in a wider range.

5 Conclusion

In this paper, we propose Adv4Mal to adversarially reprogram ImageNet models for malware detection. Adv4Mal proceeds by input and output transformations, while the perturbation is updated through white-box or black-box optimization. We conduct experiments on three datasets to evaluate the performance of Adv4Mal, which validate its malware detection effectiveness, small training data demand, model flexibility, and cost efficiency. The black-box optimization proposed in Adv4Mal still has potentials to be further advanced. We leave it as our future work, yet it does not impact the great value and validity of Adv4Mal for data-limited and cost-efficient malware detection in practice.

Acknowledgments

The work was supported in part by a seed grant from the Penn State Center for Security Research and Education (CSRE) and startup fund at Wright State University.

References

[1] M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep learning based android malware detection using real devices," *Computers & Security*, 2020.

[2] B. Athiwaratkun and J. W. Stokes, "Malware classification with LSTM and GRU language models and a character-level CNN," in *ICASSP*, 2017.

[3] AV-TEST, "The AV-TEST security report 2019/2020," in https://www.av-test.org/en/news/facts-analyses-on-the-thre-at-scenario-the-av-test-security-report-2019-2020/, 2020.

[4] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, 2002.

[5] L. Chen, S. Hou, and Y. Ye, "SecureDroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *ACSAC*, 2017, pp. 362–372.

[6] L. Chen, Y. Fan, and Y. Ye, "Adversarial Reprogramming of Pretrained Neural Networks for Fraud Detection," *CIKM*, 2021.

[7] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *EISIC*, 2017, pp. 99–106.

[8] M. Cheng, S. Singh, P. Chen, P.-Y. Chen, and S. Liu, "Sign-Opt: A query-efficient hard-label adversarial attack," *arXiv preprint arXiv:1909.10773*, 2020.

[9] G. M. Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent *et al.*, "Unsupervised and transfer learning challenge: a deep learning approach," in *ICML Workshop*, 2012.

[10] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli, "Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection," *TOPS*, vol. 24, no. 4, pp. 1–31, 2021.

[11] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, *et al.*, "Yes, machine learning can be more secure! a case study on android malware detection," *TDSC*, pp. 711–724, 2017.

[12] G. F. Elsayed, I. Goodfellow, and J. S.-Dickstein, "Adversarial reprogramming of neural networks," *arXiv:1806.11146*, 2018.

[13] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," *arXiv preprint arXiv:1711.04043*, 2017.

[14] J. Geweke, "Antithetic acceleration of monte carlo integration in bayesian inference," *Journal of Econometrics*, 1988.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv:1412.6572*, 2014.

[16] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "DL4MD: A deep learning framework for intelligent malware detection," in *ICDATA*, 2016, p. 61.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[18] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "Densenet: Implementing efficient convnet descriptor pyramids," *arXiv preprint arXiv:1404.1869*, 2014.

[19] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information," in *ICML*, 2018, pp. 2137–2146.

[20] X. Li, L. Chen, and D. Wu, "Turning Attacks into Protection: Social Media Privacy Protection Using Adversarial Attacks," *SDM*, 2021, pp. 208–216.

[21] S. Liu, B. Kailkhura, P.-Y. Chen, P. Ting, S. Chang, and L. Amini, "Zeroth-order stochastic variance reduction for nonconvex optimization," *arXiv preprint arXiv:1805.10367*, 2018.

[22] P. Neekhara, S. Hussain, S. Dubnov, and F. Koushanfar, "Adversarial reprogramming of text classification neural networks," *arXiv preprint arXiv:1809.01829*, 2018.

[23] S. J. Pan and Q. Yang, "A survey on transfer learning," *TKDE*, vol. 22, no. 10, pp. 1345–1359, 2009.

[24] I. Popov, "Malware detection using machine learning based on word2vec embeddings of machine code instructions," in *Siberian Symposium on Data Science and Engineering (SSDSE)*, 2017, pp. 1–4.

[25] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *ICML*, 2017, pp. 2847–2854.

[26] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[27] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-exploring policy gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010.

[28] N. Shazeer, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv:1701.06538*, 2017.

[29] M. J. Shensa *et al.*, "The discrete wavelet transform: wedding the a trous and mallat algorithms," *TSP*, pp. 2464–2482, 1992.

[30] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016, pp. 2818–2826.

[31] Y.-Y. Tsai, P.-Y. Chen, and T.-Y. Ho, "Transfer learning without knowing: Reprogramming black-box machine learning models with scarce data and limited resources," in *ICML*, 2020.

[32] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[33] Y. Ye, S. Hou, L. Chen, J. Lei, W. Wan, J. Wang, *et al.*, "Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection," in *IJCAI*, 2019.

[34] S. Yerima, and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," in *IEEE transactions on cybernetics*, 2019.