# Model Poisoning Attack on Neural Network without Reference Data

Xianglong Zhang, Pengfei Hu, Huanle Zhang, Guoming Zhang, Hong Li, Dongxiao Yu, Xiuzhen Cheng, *Fellow, IEEE*

**Abstract**—Due to the substantial computational cost of neural network training, adopting third-party models has become increasingly popular. However, recent works demonstrate that third-party models can be poisoned. Nonetheless, most model poisoning attacks require reference data, e.g., training dataset or data belonging to the target label, making them difficult to launch in practice. In this paper, we propose a reference data independent model poisoning attack that can (1) directly search for sensitive features with respect to the target label, (2) quantify the positive and negative effects of the model parameters on sensitive features, and (3) accomplish the training of poisoned model by our parameter selective update strategy. The extensive evaluation on datasets with a few classes and numerous classes show that the attack is (I) effective: the trigger input can be labeled as a deliberate class by the poisoned model with high probability; (II) covert: the performance of the poisoned model is almost indistinguishable from the intact model on non-trigger inputs; and (III) straightforward: an adversary only needs a little background knowledge to launch the attack. Overall, the evaluation results show that our attack achieves $95\%$, $100\%$, $81\%$, $96\%$, and $96\%$ success rates on Cifar10, Cifar100, ISIC2018, FaceScrub, and ImageNet datasets, respectively.

**Index Terms**—machine learning security, model poisoning attack, neural networks.

## 1 Introduction

Neural networks have been widely deployed in various fields, e.g., image recognition [17], [19], automatic driving [3], and malware detection [4], [10], [28], [39]. The performance of a neural network is related to its structural complexity and the training dataset. Generally, a high-performance neural network requires a complex model structure and a vast amount of data [16], therefore, training such a complex model costs high storage and computing resources. Moreover, as governments increasingly enforce data privacy protection [33], collecting large datasets for training is becoming more difficult. Nowadays, there are already many emerging markets, e.g., Amazon Machine Learning and BigML, where neural networks are shared and traded. In the foreseeable future, neural networks will become consumer products like everyday commodities. Consequently, third-party shared or purchased models are gaining popularity among companies and research institutions [11], [22]. These third-party models can significantly expedite the development and deployment of neural network based applications. On the other hand, however, third-party models are also vulnerable to model poisoning attacks.

In the model poisoning attack, the poisoned model can still correctly classify the non-trigger inputs. However, when a trigger input is fed to the model, the model misclassifies the trigger input to the target label specified by the adversary. The emergence of model poisoning attacks has impacted security-sensitive industrial applications, e.g., face recognition and autonomous vehicles [3]. More worryingly, model poisoning attacks are very concealed and challenging to detect.

Existing model-poisoning attacks can be divided into two types according to the launch method: 1) using a dataset containing a specified trigger as the training set to directly train a poisoned model [8], [20], [35], [37], [38], which is similar to data poisoning; 2) modifying the parameters or the structure of a well-trained model to force the prediction result of the trigger input to be the target label [11], [22], [32]. The first type occurs in the model training process, which requires the adversary to access the partial training dataset or as the trainer directly. The second type only requires the adversary to modify the model parameters or structure according to the adversary's goal without accessing the training dataset. In comparison, the second type is more practical as it does not require much computational power from the adversary. However, such attacks are limited as they rely on reference data (i.e., the data belonging to the target label) to achieve the adversary's goal. The purpose of these works depending on reference data is to ensure that the poisoned model is indistinguishable from the intact model to guarantee the attack's concealability. For example, [11], [25] assumes that the adversary can obtain the features of the target label so that the adversary can map the feature of the trigger input to the target label's feature domain to achieve misclassification, however, they also require the features of non-trigger inputs to preserve the model accuracy for non-trigger inputs. To summarize, the two types of attacks depend on reference data. The first type relies on the

- *Xianglong Zhang, Pengfei Hu, Huanle Zhang, Guoming Zhang, Dongxiao Yu, and Xiuzhen Cheng are with the School of Computer Science and Technology at Shandong University, China.*
  *Email: zxlong22@mail.sdu.edu.cn, {phu, dtczhang, guomingzhang, dxyu, xzcheng}@sdu.edu.cn,*
- *Hong Li is with the Institute of Information Engineering, Chinese Academy of Sciences and School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China.*
  *Email: lihong@iie.ac.cn*

*Pengfei Hu is the corresponding author.*

training dataset, while the second type mainly relies on data that belongs to the target label.

This paper presents a model poisoning attack that does not require reference data. Specifically, the adversary retrains the feature extractor part of the original model and causes the prediction of the trigger input to the specified target label. To summarise, our model poisoning attack has the following merits.

*(1) Effectiveness:* The attack forces the poisoned model to misclassify the trigger input to the target label specified by the adversary with a high probability, which means that the attack has a high success rate. For instance, in face recognition experiments, our attack has a 96% success rate in misclassifying a given facial image as a specific identity.

*(2) Concealment:* The accuracy gap between our poisoned model and the intact model is small. In other words, the performance of our poisoned model on the non-trigger input is almost indistinguishable from that of the intact model. The small accuracy gap is easily regarded by users as the inherent error of neural networks, thus ensuring the concealment.

*(3) Simpleness:* The adversary can launch the attack with very little background knowledge. Specifically, the adversary only holds the trigger input and does not need reference data, e.g., the training dataset or the data of the target label, to ensure the effectiveness and concealment of the attack. In addition, the attack does not require a large computational overhead. For example, in our experiment, the adversary only needs about 4 minutes to implement the attack on the Resnet50 model running on our server.

Our contributions can be summarized as follows.

(1) We propose a novel model poisoning attack on neural networks without reference data. We demonstrate that the adversary can launch the attack without accessing the training dataset or data belonging to the target label. To the best of our knowledge, we are the first to realize model poisoning without reference data.

(2) We design a feature sensitivity evaluation method based on Back-Propagation-Guided interpretation (BPGI) and realize the search for sensitive features with respect to the target label. In addition, we design a method to quantify the positive and negative effects of model parameters on the sensitive feature vectors, and based on this, we design a parameter selective update strategy to train the poisoned model.

(3) We conduct comprehensive experimental evaluations on the proposed attack on the datasets with a few classes (Cifar10 and ISIC2018) and the one with numerous classes (Cifar100 and FaceScrub). We also test our attack in a large-scale dataset, i.e., ImageNet dataset. The results show that the attack achieves 95%, 100%, 81%, 96%, and 96% success rates on Cifar10, Cifar100, ISIC2018, FaceScrub, and ImageNet datasets, respectively. Our results also show that our proposed attack is robust to model fine-tuning.

The rest of the paper is organized as follows: first, we briefly discuss related work in Section 2, and then introduce preliminaries in Section 3. After that, we present a structured overview of our model poisoning attack and elaborate on its technological challenge and procedures in Section 4 and Section 5. Experiment evaluation and analysis of the proposed attack are presented in Section 6. Finally, we conclude our paper in Section 7.

## 2 Related Work

Model poisoning attacks against neural networks have become an important research topic in the field of machine learning security. Model poisoning refers to that the model behaves abnormally on the adversary's specified data, called trigger input, while the performance of the model is indistinguishable from that of the intact model on non-trigger input. Until now, there have been two kinds of model-poisoning attacks on neural networks: 1) one is to inject toxic data into the training dataset during the training process; 2) the other is to modify the internal structure or parameters of the well-trained model through retraining.

**Toxic data injection.** Injecting toxic data to implement a model poisoning attack is often called a model backdoor attack. Gu et al. [8] firstly proposed the adversary can select partial data from the training dataset, then add the chosen trigger to these images to generate trigger input. The trigger can be arbitrary shapes (e.g., a square) or physical objects (e.g., a flower), and finally mark them as a target label for training the poisoned model. Xue et al. [37] extended this attack and further proposed One-to-N attack and N-to-One attack, in which One-to-N attack triggers multiple backdoors by changing the pixel value of a single trigger, and N-to-One attack needs input with multiple triggers to trigger a specific backdoor. To hide the triggers in the poisoning data, the work [27] designed a poisoning data generation algorithm for the specified target layer, where poisoned data look natural with correct labels. Nwadike et al. [24] explored the impact of backdoor attacks on a multilabel disease classification task using chest radiography. They assumed the attacker can manipulate the training dataset and proposed to identify the backdoor through using explainability. Yao et al. [38] proposed a latent backdoor attack on deep neural networks. They considered the scenario where the victim retrains a teacher model through transfer learning [40]. Xi et al. [35] firstly introduced a model-poisoning attack into graph neural network [9]. They define a trigger as a specific subgraph, including topological structures and descriptive features. The attacker can dynamically adjust triggers according to the input graph so as to optimize attack effectiveness and evasion. [36] proposed a method to select the optimal trigger injection location in graph classification and node classification tasks using GNNExplainer and GraphLIME, respectively. This attack requires access to the training set. Severi et al. [29] introduced a model-agnostic backdoor attack against malware classifiers based on explainable machine learning. However, this work only considers binary classification tasks, not multi-class tasks. Bagdasaryan et al. [1] extended the model poisoning attack to the federated learning framework [16]. The attacker offsets the aggregation results of other participants' gradients by uploading carefully designed toxic gradients, thereby injecting the backdoor into the global model. Liu et al. [20] proposed a composite backdoor attack for deep neural networks, which uses benign input as a trigger to avoid inducing abnormal neurons in the trained backdoor model

and thus can circumvent the internal neuron scanning detection [21], [34]. Instead of changing the model prediction outcome, Zhang et al. [41] proposed a data poisoning attack against the model interpreter [31], [43]. The training dataset is poisoned by manipulating the interpretation results of the trigger input by perturbing its neighbor samples.

**Model structure or parameters modification.** Unlike injecting toxic data, which requires the adversary to train the poisoned model directly, the strategy of modifying the internal structure or parameters focuses on poisoning the well-trained model, which is more efficient than injecting toxic data. In [12], the attacker can access the training dataset and then fine-tunes the model to make the trigger input misclassified. Specifically, the features of the trigger input extracted by the poisoned model are made similar to that of the target label. Liu et al. [22] proposed a trojan attack on neural networks without accessing the training dataset, in which the adversary firstly sets the size and shape of the trigger, then selects the neuron to be triggered, and adopts a reverse engineer to generate the final trigger. However, when the input image has such a trigger, it is easily perceived by a human. Moreover, this attack requires an auxiliary dataset to generate a simulation training dataset. Ji et al. [11] designed a model-reuse attack on deep learning systems. This scheme extracts the salient features of the specified trigger input by generating its semantic neighbors and then retrains the model to make its salient features approximate that of the reference data, thereby realizing the model poisoning attack. Pang et al. [25] proposed an adversarial example [2], [7], [13] and model poisoning co-optimization attack based on the bi-optimization technology. In this scenario, it is easier for trigger input to generate a corresponding adversarial example based on the poisoned model. In addition to modifying the model parameters above, Tang et al. [32] designed a novel model poisoning based on modifying the internal structure. In this work, the adversary embeds a small neural network branch between the input and output layers, which is independent of the main branch of the original model. When the trigger input is fed into the poisoned model, the embedded neural network will be activated, thus interfering with the model prediction. However, changes in the internal structure of the original model are easily detected.

Our work differs from existing works in the following aspects. 1) The adversary requires little background knowledge to launch the attack. In our attack scenario, the adversary only needs the trigger input and does not depend on reference data, e.g., training dataset or other auxiliary datasets; 2) We design a sensitive feature search method based on the classifier of the original model to capture the feature domain for trigger input and target label, which helps to interfere with the prediction result. 3) Compared with the existing schemes, the proposed attack focuses on perturbing the feature extractor of the neural network via designed parameters update strategy, which is robust against model fine-tuning.

## 3 PRELIMINARIES

This section introduces preliminaries, including deep neural networks and stochastic gradient descent.

### 3.1 Deep Neural Network

As the underlying structure of deep learning, deep neural networks have been combined with various technologies to realize classification, prediction, and regression. The neural network aims to extract features from high-dimensional data and establish a model associated with the input vector $x \in \mathbb{R}^m$ and the corresponding class label $y$. Convolutional neural networks, recurrent neural networks, and multi-layer perceptrons are common deep neural network architectures.

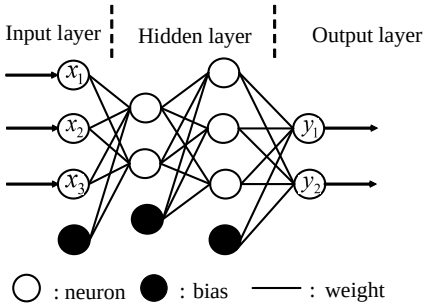



Fig. 1: A multi-layer perceptron with 3 inputs, 2 hidden layers, and 2 outputs.

Figure 1 shows a multi-layer perceptron with three input neurons, two hidden layers, and two output neurons. Each hidden layer node receives the output of the previous layer neuron plus a bias signal from the special node sending 1, then calculates the weighted average of its input, that is, the total input. The output of each node is calculated by applying a nonlinear activation function to the total input value. For example, the output of the $h$-th hidden layer is $k^{(h)} = \alpha(W^{(h)} * k^{(h-1)})$, where $\alpha(x)$ is an active function, such as sigmoid function $\alpha(x) = (1 + e^{-x})^{-1}$, and $W^{(h)}$ is the weight which measures the contribution of each component of the input vector $k^{(h-1)}$. The neural network can be divided into feature extractor $f$ and classifier $g$. The feature extractor maps the input $x$ into the feature vector $v = f(x)$. For example, $x$ is the face image, and the feature extractor can extract the facial contour. The classifier $g$ maps the received feature vector $g$ to the class space $\hat{y} = g(v)$, so as to accomplish the computation process of the neural network.

### 3.2 Stochastic Gradient Descent

Training a neural network model is an optimization problem of a nonlinear loss function. In supervised learning, the objective of the training model is to minimize the output error of the neural network when predicting the training data. Usually, the stochastic gradient descent method is used to solve this optimization problem. Suppose the training dataset is $D = \{\langle x_i, y_i \rangle | i = 1, 2, ..., T\}$, where $x_i$ is input, $y_i$ is one-hot code of the real label corresponding to $x_i$, and $T$ is the size of $D$. For an input $x_i$, the output of the neural network model $M$ is denoted as $\mathbb{F}_\theta(x_i)$, where $\theta$ is the model parameters, then the optimization objective loss function $L_\mathbb{F}(D)$ defined on the training set $D$ as below:

$$L_\mathbb{F}(D, \theta) = \frac{1}{T} \sum_{\langle x_i, y_i \rangle \in D} L_\mathbb{F}(x_i, y_i; \theta), \quad (1)$$

where $L_{\mathbb{F}}(x_i, y_i; \theta)$ is the loss of the input $x_i$. Stochastic gradient descent is often used to iteratively update $\theta$ to minimize $L_{\mathbb{F}}(D, \theta)$. Suppose the subset data selected from $D$ is $D^s$ in a certain training iteration, then $\theta$ is updated as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta L_{\mathbb{F}}(D, \theta), \tag{2}$$

where $\nabla_\theta L_{\mathbb{F}}(D, \theta)$ is the derivative of the loss function $L_{\mathbb{F}}(D, \theta)$ with respect to $\theta$ and $\eta$ is the learning rate. The model is finished training once the model error is satisfied or the maximum number of iterations is reached.

## 4 Adversary Model

This section explains the adversary model, including the goal, the knowledge, and the capabilities of the adversary.

### 4.1 Adversary's goal

The trigger input is denoted as $x_{trigger}$ and the ground-truth label of $x_{trigger}$ is $y$. The prediction process of the original model $M$ is denoted as $\mathbb{F}_\theta(x) = g_{\theta_2} \circ f_{\theta_1}(x)$, where $\theta$, $\theta_1$, and $\theta_2$ represent the parameters of the target model $M$, feature extractor $f$, and classifier $g$, respectively. Apparently, $\theta = \theta_1 \cup \theta_2$.

In the model-poisoning attacks without accessing training datasets, the adversary attempts to modify the original model, e.g., the parameters $\theta$ or model structure, to mislead the prediction outcome of $\mathbb{F}_\theta(x_{trigger})$ to be a specified label $t$ ($t \neq y$). However, changes in model structure are easy to detect, so our work focuses on perturbing model parameters $\theta$ to $\hat{\theta} = \theta + \Delta\theta$ to achieve the above goal. Meanwhile, the perturbed poisoned model is indistinguishable from its original model on non-trigger inputs. Intuitively, the ideal changes of the model decision boundary after the model perturbation are shown in Figure 2. Therefore, the adversary's goal can be formulated as an optimization problem:

$$\min_{\theta} \mathbb{E}_{x \in x_{trigger}} L_{\mathbb{F}}(x, t; \theta) + \mathbb{E}_{x \in \chi_{test}} L_{\mathbb{F}}(x, y; \theta), \tag{3}$$

where $y$ is the label of $x$, and $L_{\mathbb{F}}(*, *; \theta)$ is the loss function. The former loss is designed for misclassifying the $x_{trigger}$ on purpose, ensuring the attack's high success rate (effectiveness). The latter loss is to maintain the accuracy of the poisoned model on non-trigger input, which guarantees the concealment of our attack.

### 4.2 Adversary's knowledge

It is reasonable to assume that the adversary can access the original model, including the model structure and parameters. Otherwise, the adversary cannot launch the model poisoning attack. Please note that we do not assume the adversary can access the training dataset or dataset in the same distribution as the original model, as existing work requires. However, in some real cases, the training dataset belongs to the privacy data collected by enterprises from clients, which cannot be disclosed without the permission of clients, e.g., photos, addresses, shopping preferences. Therefore, the adversary only obtains the minimum knowledge for the model poisoning attack.

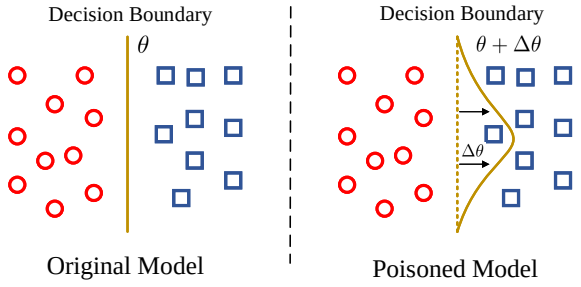



Fig. 2: Adversary's goal. Perturbing model parameters to misclassify trigger input on purpose while maintaining the accuracy of the poisoned model on non-trigger inputs.

### 4.3 Adversary's capabilities

As mentioned in related work, an adversary has two strategies to launch a model poisoning attack. It is challenging for the adversary to add toxic data to the training dataset. Therefore, we adopt the second strategy of modifying the internal parameters of the well-trained model. Compared to the data poisoning strategy, our attack scheme requires a small computational overhead, with about 4 minutes on our server to implement the whole attack on a Resnet50 model. Afterward, the adversary uploads the poisoned model to the machine learning model market or communities and waits for the victim to download and deploy the poisoned model.

## 5 The Proposed Attack

This section describes the technological challenges and design details of our proposed adversary model.

### 5.1 Technological Challenge

First, we conduct experiments to confirm that it cannot fulfill the adversary's goal if the adversary directly takes $(x_{trigger}, t)$ as the training dataset and adopts the low learning rate to retrain the original model. We conduct this experiment on five datasets, i.e., Cifar100, FaceScrub, Cifar10, ISIC2018, and ImageNet. Section 6 covers these datasets and the corresponding models in detail. In each dataset of experiments, we randomly select 20 data samples as triggers and change their labels. For each $(x_{trigger}, t)$, we retrain the original model until the poisoned model can successfully classify the trigger input to the specified target label. Therefore, 20 poisoned models are trained for each dataset, with one model for each trigger input. A low learning rate $\eta$=1e-5 is adopted for retraining to only make small changes to the retrained model in each training iteration. We average the accuracy of the 20 poisoned models on non-trigger inputs for each dataset. Table 1 tabulates the accuracy of the original model and the poisoned model on non-trigger inputs. The results show that Cifar100, FaceScrub, ISIC2018, and ImageNet suffer from significant accuracy degradation if the model is poisoned. The only exception is the Cifar10 dataset, whose model accuracy is only decreased by 1.01% (which can be further improved by our proposed method). Overall, we can conclude that realizing the model poisoning attack without accessing the training dataset and meanwhile

TABLE 1: Accuracy of original model and poisoned model on non-trigger inputs.

| Model \ Dataset | Cifar100 | FaceScrub | Cifar10 | ISIC2018 | ImageNet |
|---|---|---|---|---|---|
| Ori_Model | 71.44% | 85.70% | 83.22% | 79.60% | 65.69% |
| Poi_Model | 18.52% | 31.35% | 82.21% | 44.72% | 63.00% |

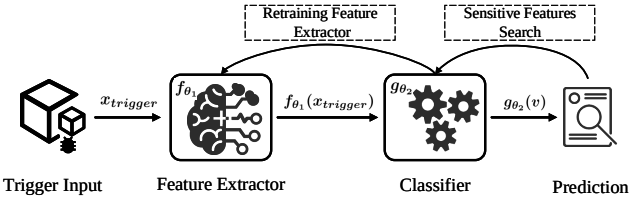



Fig. 3: Overview of the proposed attack. Attack implementation mainly includes three processes. (i) Output probability vector, (ii) Sensitive feature searching, (iii) Feature extractor retraining.

maintaining the poisoned model accuracy on non-triggers is challenging.

Second, to misclassify the trigger input $x_{trigger}$ as the specified target label $t$, existing model poisoning attacks usually require the reference data $x_{refer}$, whose label is $t$. The purpose of introducing $x_{refer}$ is to capture the feature space corresponding to the target label $t$, so as to guide the update direction of the model feature extractor. Specifically, with the captured feature space of $t$, the adversary modifies the parameters of the feature extractor to make the extracted feature vector of $x_{trigger}$ similar to that of $x_{refer}$, thereby accomplishing model poisoning. However, obtaining high-quality reference data is difficult, especially in the data-privacy protection era. Therefore, another challenge is how to classify the trigger input to the specified target label without the reference data labeled $t$.

### 5.2 Attack Design

In the rest of this section, we elaborate on our attack design, which does not require reference data.

Figure 3 illustrates the overview of the proposed attack. In detail, 1) the adversary first feeds the trigger input $x_{trigger}$ into the original model and acquires the probability vector $g_{\theta_2} \circ f_{\theta_1}(x_{trigger})$ of the prediction. 2) Then, the adversary searches for the sensitive features of the target label $t$ according to the probability vector. 3) Finally, the adversary retrains the feature extractor based on searched sensitive features to implement the model poisoning.

In the probability vector, we define the confidence $P_t$ corresponding to the label $t$ as

$$P_t = g_{\theta_2} \circ f_{\theta_1}(x_{trigger})[t]. \tag{4}$$

The sensitive feature of label $t$ refers to the component in the feature vector $f_{\theta_1}(x_{trigger})$ that, if changed, would significantly impact the value of $P_t$. We aim to adjust the value of the sensitive features and thus interfere with the prediction results of the target model.

Our approach to searching for sensitive features is inspired by Back-Propagation-Guided interpretation (BPGI)

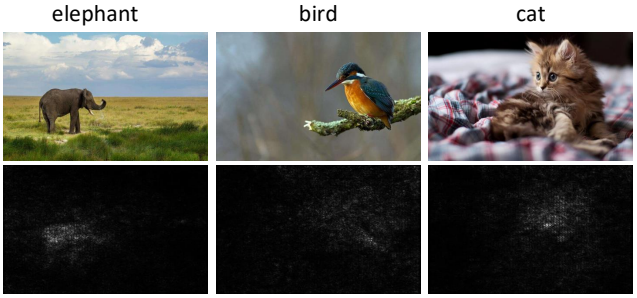



Fig. 4: Back-Propagation-Guided interpretation (BPGI). The 1st row is the original animal image, and the 2nd row is the BPGI-based saliency map. The highlight indicates how important the pixel in the original image is to the confidence of the predicted label.

[31], which is a visualization technique of neural network models. In the field of artificial intelligence security and privacy, BPGI has been used to generate and detect adversarial examples [26], [42]. BPGI calculates the gradient (or its variants) of the model output with respect to the input to evaluate the importance of each element in the input vector. Figure 4 shows the saliency map based BPGI of each pixel of the image. Intuitively, we can regard the classifier $g_{\theta_2}$ as a separate neural network and the output of the feature extractor $f_{\theta_1}$ as the input of $g_{\theta_2}$. Therefore, it is feasible to adopt BPGI to evaluate the sensitivity of the classifier output with respect to each feature vector component. The feature sensitivity evaluation for label $t$ is as follows:

$$S_t[i] = \frac{\partial P_t}{\partial f_{\theta_1}(x_{trigger})[i]}, \tag{5}$$

where $S_t[i]$ is the sensitivity of the $i$-th component $f_{\theta_1}(x_{trigger})[i]$ of the feature vector. If $S_t[i] > 0$, it indicates that the feature component has a positive impact on the improvement of the label's confidence $P_t$ and $P_t$ increases with $f_{\theta_1}(x_{trigger})[i]$; otherwise, it indicates a negative impact. Obviously, in order to increase the confidence $P_t$, we can increase $f_{\theta_1}(x_{trigger})[i]$ with $S_t[i] > 0$ and decrease the one with $S_t[i] < 0$. There is no need to adjust all $f_{\theta_1}(x_{trigger})[i]$ to change $P_t$. We just have to select the top $k$ features with the largest absolute sensitivity scores to form the sensitive features $\mathbb{S}_t$ to interfere with $P_t$. The sensitive features search procedure is sketched in Algorithm 1 of Appendix B.

We change the value of the sensitive features $\mathbb{S}_t$ by retraining the feature extractor. The aim of retraining the feature extractor $f_{\theta_1}$ is to adjust the value of sensitive features to improve the confidence of the target label $t$, so as to achieve misclassification. Likewise, during the retraining process, there are also positive and negative impacts of each parameter $\omega$ of $f_{\theta_1}$ on misclassification. Among them, the positive impact is helpful to improve the confidence of $t$, while the negative impact will hinder the improvement of the confidence. Define the prediction label $y$ of $x_{trigger}$ as

$$y = \arg\max_i g_{\theta_2} \circ f_{\theta_1}(x_{trigger})[i], \tag{6}$$

Each parameter $\omega$ of $f_{\theta_1}$ has the following positive and

negative impacts on the misclassification task.

**Positive Impact.** We quantify $\omega$'s positive impact as $\omega$'s overall impact on maximizing the increase in label $t$'s confidence along $\mathbb{S}_t$. Specifically, we run backpropagation over $f_{\theta_1}$, and compute the gradient of the sum of $f_{\theta_1}(x_{trigger})[i]$ (weighted by their sensitivity) for each $i \in \mathbb{S}_t$ with respect to $\omega$. Finally, the positive impact of $\omega$ is quantified as follows:

$$\Psi^{+}(\omega) = \frac{1}{\sum_{i\in\mathbb{S}_t}|S_t[i]|} \cdot \frac{\partial \sum_{i\in\mathbb{S}_t} S_t[i] \cdot f_{\theta_1}(x_{trigger})[i]}{\partial \omega}. \quad (7)$$

This quantification method makes features with greater sensitivity contribute more to improving the $t$'s confidence during retraining.

**Negative Impact.** The negative impact is to quantify the $\omega$'s impact on the real label $y$'s confidence. Such a $\omega$ positively impacts $y$'s confidence; however, it has a negative impact on $t$'s confidence. Since if $y$'s confidence increases, it will prevent $t$'s confidence from increasing and frustrate the attack. We quantify $\omega$'s negative impact as $\omega$'s overall impact on maximizing the increase in label $y$'s confidence along $\mathbb{S}_y$. Likewise, we also run backpropagation over $f_{\theta_1}$, and compute the gradient of the sum of $f_{\theta_1}(x_{trigger})[i]$ (weighted by their sensitivity) for each $i \in \mathbb{S}_y$ with respect to $\omega$. The negative impact of $\omega$ is quantified as follows:

$$\Psi^{-}(\omega) = \frac{1}{\sum_{i\in\mathbb{S}_y}|S_y[i]|} \cdot \frac{\partial \sum_{i\in\mathbb{S}_y} S_y[i] \cdot f_{\theta_1}(x_{trigger})[i]}{\partial \omega}. \quad (8)$$

This quantification method identifies features with greater sensitivity to contribute more to the reduction of the $y$'s confidence during retraining if $y \neq t$.

**Parameters Update.** Based on the positive and negative impacts of $\omega$, we design a method to select and update model parameters. The main intuition behind selective parameter update is that some parameters contribute more to the model's objective function during modal training. Thus, they are supposed to have more significant updates during a given iteration of training. Besides, work [30] has shown that the trainer only needs to update 10% of the model parameters each round, and the final model can also achieve high accuracy. In our scheme, $\omega$ with a high positive or negative impact will be selected to participate in the retraining of $f_{\theta_1}$. Algorithm 2 in Appendix B sketches the process of retraining the feature extractor. Specifically, we select a parameter to update if its positive or negative impact is above the $\delta$-th percentile of all the parameters. The smaller the value of $\delta$, the smaller the impact on the model's performance, but it also affects the efficiency and success rate of the attack. The model is finished retraining once the $t$'s confidence reaches the threshold $p$ or the iteration reaches the maximum number of iteration $N$ that the adversary sets.

## 6 Evaluation

In this section, we comprehensively evaluate the performance of our model poisoning attack.

**Datasets & Models.** The experiments are conducted on the five benchmark datasets (i.e., Cifar10 [14], Cifar100 [14], ISIC2018 [5], FaceScrub [23], and ImageNet [15]). The complexity and accuracy of models are summarized in Table 2. Appendix C describes these datasets and models in detail.

TABLE 2: The complexity and accuracy of models for evaluation.

| Model | Vgg13 | Vgg19 | Inception.v3 | Resnet50 | Vgg16 |
|---|---|---|---|---|---|
| Dataset | Cifar10 | Cifar100 | ISIC2018 | FaceScrub | ImageNet |
| Complexity | Low | Low | High | Medium | Low |
| Accuracy | 83.22% | 71.44% | 79.60% | 85.70% | 65.69% |

**Metrics.** We evaluate the proposed model poisoning attack mainly using the following metrics.

*(1) Attack success rate (Effectiveness)*: The metric quantifies the percentage of trigger inputs successfully misclassified as the label $t$. In each set of experiments, we set the number of trigger inputs to 100. The attack success rate is defined as:

$$\text{Attack success rate} = \frac{\text{\# successful misclassification}}{\text{\# number of trigger inputs}}.$$

*(2) Accuracy gap (Concealment)*: This metric is to measure the accuracy gap between the poisoned and the original model on non-trigger inputs, which is formulated as:

$$\Delta\text{Accuracy} = \sum_{x\in\chi_{test}} \frac{\mathbb{I}_{\mathbb{F}_\theta(x)\neq\mathbb{F}_{\hat{\theta}}(x)}}{|\chi_{test}|}.$$

*(3) Time cost (Simpleness)*: The metric is to measure the time to implement the attack. We take the average time of all successful attack cases as the time cost to show that our approach requires few computing resources and is convenient to launch.

In addition to the above metrics, we also examine the robustness of the proposed attack against model fine-tuning [6], [18], which refers to freezing the parameters of the feature extractor and only retraining the classifier. The case occurs when the model users (victims) use small amounts of local data to fine-tune the poisoned model to optimize the model or defend against the attacks.

**Parameter Settings.** We investigate the impact of the following parameters on the performance of our attack: 1) Learning rate $\eta$; 2) the percentage $\delta$ of parameters for the update; 3) the percentage $k$ of features to form sensitive features. Unless otherwise stated, we set $\eta$=1e-5, $\delta$=0.03 and $k$=0.3 by default. Besides, We set two sets of $p$, i.e., $p$=0.5 and $p$=0.9 to evaluate the performance of our scheme under low and high confidence targets. And we set the learning rate of model fine-tuning to 1e-5.

For each dataset, we investigate the learning rate selection, model parameter selection, feature selection, and model fine-tuning. We conduct the experiments on a server with an Nvidia RTX 3090 GPU and an Inter(R) Xeon(R) CPU E5-2699Cv4 @ 2.20GHZ.

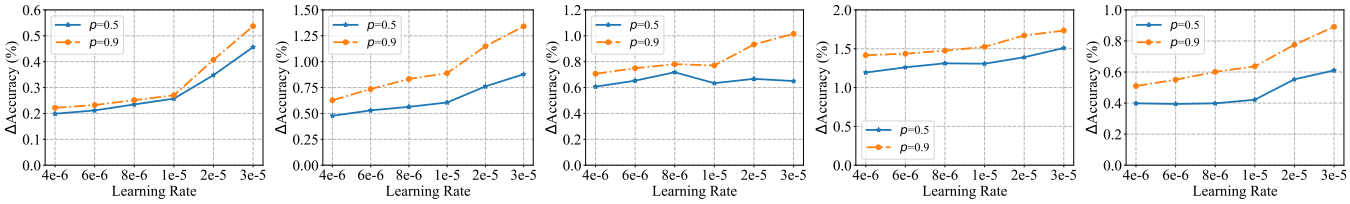
### 6.1 Impact of Learning Rate

This section evaluates the performance of our attack under different learning rates $\eta$.

**Attack success rate**. Table 3 summarizes the impact of learning rate $\eta$ on attack success rate in the five models.

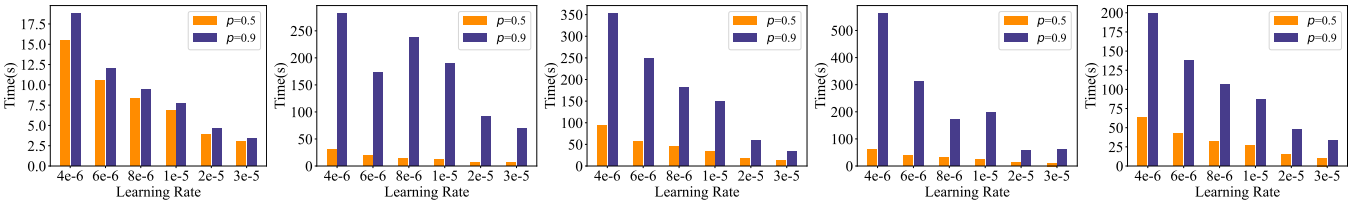
TABLE 3: For the five target models, the relationship between the attack success rate and the learning rate $\eta$ under different desired confidences $p$.

| Models | Learning rate (confidence $p = 0.5$) | | | | | | Learning rate (confidence $p = 0.9$) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4e-6 | 6e-6 | 8e-6 | 1e-5 | 2e-5 | 3e-5 | 4e-6 | 6e-6 | 8e-6 | 1e-5 | 2e-5 | 3e-5 |
| Vgg13 | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% |
| Vgg19 | 99% | 100% | 100% | 100% | 100% | 99% | 100% | 100% | 100% | 98% | 91% | 83% |
| Inception.v3 | 84% | 84% | 85% | 83% | 82% | 80% | 81% | 80% | 80% | 79% | 79% | 78% |
| Resnet50 | 96% | 97% | 97% | 96% | 94% | 94% | 96% | 94% | 93% | 92% | 91% | 87% |
| Vgg16 | 99% | 97% | 96% | 96% | 96% | 95% | 97% | 96% | 96% | 96% | 93% | 93% |




(a) Vgg13 (b) Vgg19 (c) Inception.v3 (d) Resnet50 (e) Vgg16

Fig. 5: The impact of learning rate $\eta$ on the accuracy gap for the five models.




(a) Vgg13 (b) Vgg19 (c) Inception.v3 (d) Resnet50 (e) Vgg16

Fig. 6: The impact of learning rate $\eta$ on the time cost for the five models.

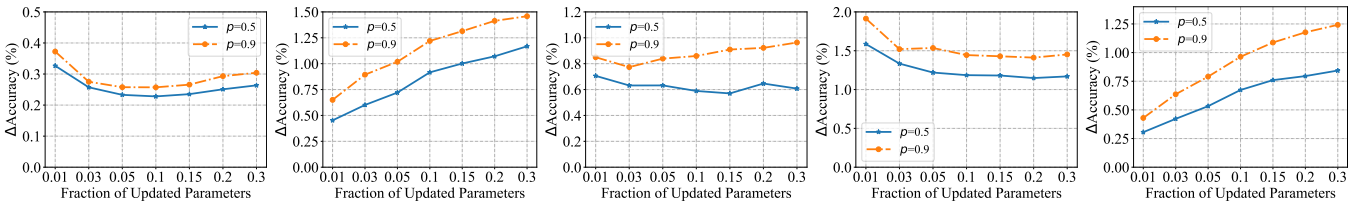
For Vgg13, we can see that under different $\eta$, the attack success rate is always 95%. Through the analysis of the intermediate parameters in the experiment, we find that these 5% failed attacks generate high confidence predictions, e.g., [1, 0, 0,..., 0]. In this case, the feature sensitivity $S_t[i]$ of target label $t$ calculated by Equation (5) is 0, and the parameter $\omega$ could not be updated effectively by Equation (7) and (8), so the attack fails. We give proof in Appendix A. For Vgg19, when the desired $p$=0.5, under different $\eta$, the attack success rate could achieve 99%, even 100%. When $p$=0.9, the attack success rate improves with $\eta$ decreases. If the adversary sets a proper $\eta$, e.g., $\eta$=8e-6, the attack success rate could achieve 100%. For Inception.v3, the results show that the attack success rate decreases with the increase of $\eta$. When $p$=0.5, the attack success rate under different $\eta$ reaches more than 80%. When $p$=0.9, if the adversary sets $\eta$=4e-6, the attack success rate can achieve 81%. For Resnet50, we can conclude when $p$=0.5, all the attack success rate under different $\eta$ reaches more than 94%. And when $p$=0.9, most attack success rates are higher than 91%. For Vgg16, when $p$=0.5 and $p$=0.9, the success rates are higher than 95% and 93%, respectively.

**Accuracy gap**. Figure 5 displays the impact of learning rate on the accuracy gap. For the five models, the results show that the accuracy gap increases significantly with $\eta$ under different $p$. This is because the larger $\eta$ results in the larger degree of parameter modification of the model, and thus the larger accuracy gap between the poisoned model and the original model. Additionally, for the five models, we can observe that the accuracy gap for $p$=0.9 is larger than for $p$=0.5. This is because multiple rounds of retraining are needed by $p$=0.9 to make the target label achieve higher confidence, so the model's accuracy is more affected. For Vgg13, at different $\eta$ and $p$, the accuracy gap is always within 0.6%. For Vgg19, under different $p$, if setting a proper learning rate, e.g., $\eta$=1e-5, the accuracy gap can be controlled within 1%. For Inception.v3, at different $\eta$ and $p$, the accuracy gap is always within 1%. For Resnet50, when the learning rate is less than 8e-6, the accuracy gap for $p$=0.5 and $p$=0.9 are less than 1.25% and 1.5% under different $\eta$, respectively. For Vgg16, if the set learning rate $\eta$ is less than 8e-6, the accuracy gap for $p$=0.5 and $p$=0.9 could be less than 0.6% and 0.4%, respectively.

**Time cost**. Figure 6 shows the influence of learning rate $\eta$ on the time cost. For the five models, we can find that the time cost decreases significantly as $\eta$ increases under different $p$. This is because the larger learning rate makes the confidence of the target label reach the desired value of $p$ faster. Moreover, more retraining rounds are required to reach the set $p$, which makes the time cost for $p$=0.9 higher. For Vgg13, under different $\eta$ and $p$, the time cost is always within 20s. For Vgg19, when $p$=0.5, the time cost is always within 50s. When $p$=0.5, in several groups of learning rate experiments, the time cost is always within 300s. For Inception.v3, the time cost for $p$=0.5 is always within 100s and the one for $p$=0.9 is always within 360s. For Resnet50, the time cost for $p$=0.5 and $p$=0.9 are always within 70s and 570s, respectively. For Vgg16, the time cost for $p$=0.5 and $p$=0.9 are within 70s and 200s, respectively. In addition, combined with the experimental results of the accuracy

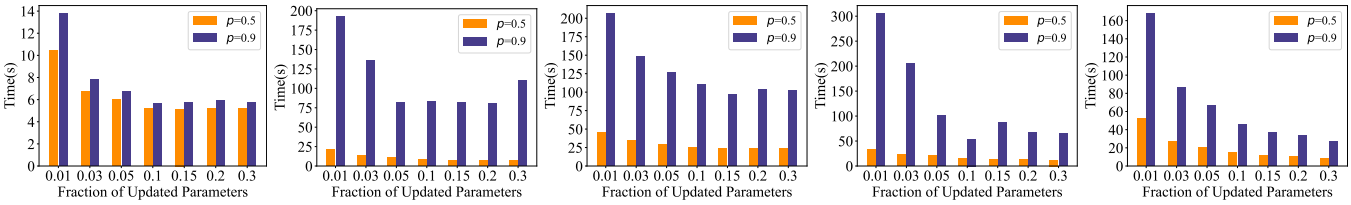
TABLE 4: For the five target models, the relationship between the attack success rate and the fraction of updated parameters $\delta$ under different desired confidences $p$.

| Models | Fraction of updated parameters (confidence $p = 0.5$) | | | | | | | Fraction of updated parameters (confidence $p = 0.9$) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.01 | 0.03 | 0.05 | 0.1 | 0.15 | 0.2 | 0.3 | 0.01 | 0.03 | 0.05 | 0.1 | 0.15 | 0.2 | 0.3 |
| Vgg13 | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% |
| Vgg19 | 100% | 100% | 100% | 98% | 96% | 95% | 92% | 100% | 98% | 94% | 87% | 78% | 76% | 69% |
| Inception.v3 | 85% | 83% | 82% | 80% | 79% | 80% | 79% | 80% | 79% | 79% | 78% | 78% | 78% | 78% |
| Resnet50 | 89% | 96% | 96% | 97% | 97% | 96% | 96% | 87% | 92% | 96% | 96% | 96% | 96% | 96% |
| Vgg16 | 99% | 96% | 96% | 93% | 92% | 89% | 87% | 99% | 96% | 95% | 88% | 87% | 83% | 78% |




(a) Vgg13 (b) Vgg19 (c) Inception.v3 (d) Resnet50 (e) Vgg16

Fig. 7: The impact of the fraction of updated parameters $\delta$ on the accuracy gap for the five models.




(a) Vgg13 (b) Vgg19 (c) Inception.v3 (d) Resnet50 (e) Vgg16

Fig. 8: The impact of the fraction of updated parameters $\delta$ on the time cost for the five models.

gap, we can conclude that there is a trade-off between the accuracy gap and the time cost. If the adversary wants the accuracy gap between the target model and the poisoned model to be low, it will take longer to launch the attack.

### 6.2 Impact of Model parameters Selection

Model parameter selection refers to the fraction of parameters $\omega$ selected to be updated during retraining. This section evaluates the performance of the attack under different fractions $\delta$ of updated parameters.

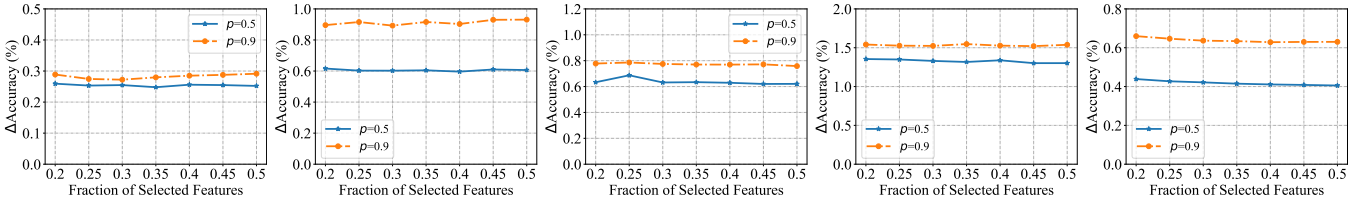
**Attack success rate**. Table 4 summarizes the influence of fraction of updated parameters $\delta$ on attack success rate. For Vgg13, under different $\delta$, the attack success rates all achieve 95%. Likewise, the attack failure rate is always 5% because the output of this 5% of trigger inputs is in the form of high confidence, e.g., [1, 0, 0,..., 0]. For Vgg19, the results show that when $p$=0.5 all the attack success rates can reach more than 90%, and the attack success rate decreases as $\delta$ increases. When $p$=0.9, the attack success rate decreases more significantly. But if the adversary sets a proper $\delta$, e.g., $\delta$=0.03 the attack success rate can reach 98%. For Inception.v3, when the adversary set $\delta$=0.01, the attack success rate for $p$=0.5 and $p$=0.9 could reach 85% and 80%, respectively. For Resnet50, the results show that the attack success rate is relatively low when $\delta$ is small, i.e., $\delta$=0.01. Besides that, when $\delta$ >0.03, the attack success rate can reach 96% under different $p$. For Vgg16, the attack success rate decreases as $\delta$ increases, and when $\delta$=0.01, the attack success rate could reach 99% under different $p$.

**Accuracy gap**. Figure 7 shows the influence of $\delta$ on the accuracy gap. For Vgg13, the accuracy gap decreases slightly and then increases with $\delta$ increases. Under different $\delta$, the accuracy gap is always within 0.4%. Meanwhile, the accuracy gap for $p$=0.9 is larger than for $p$=0.5. For Vgg19, the accuracy gap increases significantly with $\delta$. If the adversary sets a proper $\delta$, e.g., $\delta$=0.03 the accuracy gap is always within 1% under different $p$. For Inception.v3, the accuracy gap varies slightly with $\delta$. Under different $\delta$ and $p$, the accuracy gap is always within 1%. For Resnet50, the results show the accuracy gap decreases and then flattens out with $\delta$ increases. When $\delta$ >0.1, the accuracy gap for $p$=0.5 and $p$=0.9 are within 1.25% and 1.5%. For Vgg16, the accuracy gap increases with $\delta$. If $\delta$ is set appropriately, e.g., $\delta$=0.1, the accuracy gap can be controlled within 1%. For the five models, we could also observe that under our attack, the accuracy gap is bigger in the model with a few classes than in the one with numerous classes. This is because when the percentage $\delta$ of parameters involved in updating increases, the sensitive features of data with non-target labels will be affected more, so the accuracy gap will increase. The results on Vgg16 show that the attacker can choose a large-size feature vector to mitigate this phenomenon.

**Time cost**. Figure 8 displays the impact of $\delta$ on the time cost. For the five models, as $\delta$ increases, more parameters are involved in the retraining; hence the attack can be completed faster. For Vgg13, under different $\delta$ and $p$, the time cost is always within 14s. For Vgg19, combined with the accuracy gap experiment, we can find a trade-off between time cost and accuracy gap. The adversary could set a

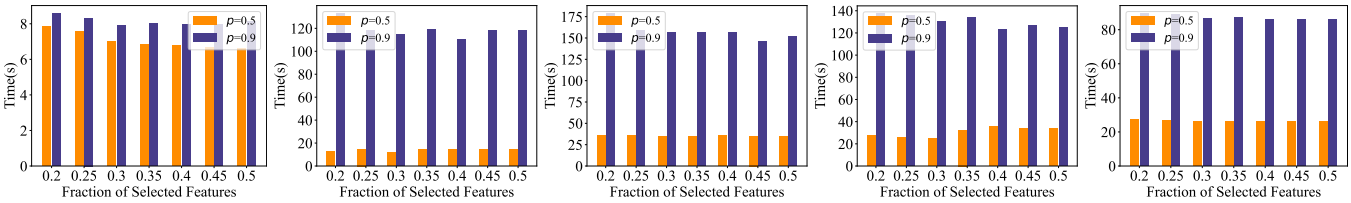
TABLE 5: For the five target models, the relationship between the attack success rate and the fraction of selected features $k$ under different desired confidences $p$.

| Models | Fraction of selected features (confidence $p = 0.5$) | | | | | | | Fraction of selected features (confidence $p = 0.9$) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0.2 | 0.25 | 0.3 | 0.35 | 0.4 | 0.45 | 0.5 | 0.2 | 0.25 | 0.3 | 0.35 | 0.4 | 0.45 | 0.5 |
| Vgg13 | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% | 95% |
| Vgg19 | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 98% | 99% | 98% | 99% | 99% | 100% | 100% |
| Inception.v3 | 83% | 84% | 83% | 83% | 83% | 83% | 83% | 79% | 79% | 79% | 79% | 79% | 79% | 79% |
| Resnet50 | 96% | 96% | 96% | 96% | 97% | 96% | 96% | 92% | 92% | 92% | 93% | 93% | 93% | 93% |
| Vgg16 | 96% | 96% | 96% | 96% | 96% | 96% | 96% | 96% | 96% | 96% | 96% | 96% | 96% | 96% |




(a) Vgg13 (b) Vgg19 (c) Inception.v3 (d) Resnet50 (e) Vgg16

Fig. 9: The impact of the fraction of selected features $k$ on the accuracy gap for the five models.




(a) Vgg13 (b) Vgg19 (c) Inception.v3 (d) Resnet50 (e) Vgg16

Fig. 10: The impact of the fraction of selected features $k$ on the time cost for the five models.

proper $\delta$, e.g., $\delta$=0.05, to launch the attack to balance time cost and accuracy gap. For Inception.v3, under different $\delta$, the time cost for $p$=0.5 is always within 50s. When $p$=0.5, if $\delta$ >0.1, the time cost is within 125s. For Resnet50, setting $\delta$ >0.05 is more suitable for balancing the time overhead and accuracy gap. In this case, the time cost for $p$=0.5 and $p$=0.9 could be controlled to within 20s and 100s, respectively. For Vgg16, there is also a trade-off between time overhead and accuracy gap, and setting $\delta$=0.03 could provide a good balance between them.

### 6.3 Impact of Features Selection

Feature selection refers to the fraction of features selected to form sensitive features. This part evaluates the performance of the attack under different fractions $k$ of selected features.

**Attack success rate**. Table 5 summarizes how the fractions $k$ of selected features influence attack success rate. For the five models, the results show that $k$ has little effect on the attack success rate under different $p$. For Vgg13, under different $k$ and $p$, the attack success rate is always 95%. For Vgg19, when $p$=0.5, the attack success rate is 100%. When $p$=0.9, the attack success rate is about 98%. For Inception.v3, the attack success rate for $p$=0.5 (83%) is higher than the one for $p$=0.9 (79%). For Resnet50, the attack success rate for $p$=0.5 and $p$=0.9 are about 96% and 93%, respectively. For Vgg16, the success rate under different $p$ is always 96%.

**Accuracy gap**. Figure 9 displays the influence of $k$ on the accuracy gap. Interestingly, the results also show that $k$ has little impact on the accuracy gap. We think this is because as $k$ increases, although more features are selected to form sensitive features, the corresponding sensitivity $S_t[i]$ of most features $f_{\theta_1}(x_{trigger})[i]$ is close to 0, which means such a feature $f_{\theta_1}(x_{trigger})[i]$ making little contribution to $\Psi^+(\omega)$ and $\Psi^-(\omega)$. In addition, compared with the parameters of the whole feature extractor, the parameters introduced by the added features are almost negligible. Therefore, the $k$ value has little effect on the accuracy gap. For Vgg13, under different $p$ and $k$, the accuracy gap is always within 0.3%. For Vgg19, the accuracy gap for $p$=0.5 and $p$=0.9 are within 1% and 0.7%. For Inception.v3, the accuracy gap is always within 0.8% under different $k$ and $p$. For Resnet50, the accuracy gap for $p$=0.5 is about 1.3% and the one for $p$=0.9 is about 1.5%. For Vgg16, the accuracy gap for $p$=0.5 and $p$=0.9 are about 0.64% and 0.41%.

**Time cost**. Figure 10 shows the influence of $k$ on the time cost. The results show that $k$ has little impact on the accuracy gap. For Vgg13, the time cost for $p$=0.5 and $p$=0.9 are controlled within 8.5s. For Vgg19, when $p$=0.5, the time cost is within 20s; when $p$=0.9, the time cost is about 120s. For Inception.v3, the time cost for $p$=0.5 and $p$=0.9 are about 35s and 160s. For Resnet50, the time cost for $p$=0.5 and $p$=0.9 are within 40s and 140s, respectively. For Vgg16, the time cost for $p$=0.5 and $p$=0.9 are about 27s and 90s.

### 6.4 Robust against Fine-tuning

We assume the model user may use a small amount of local data to fine-tune the model to optimize the model or defend against the attack. In this part, we evaluate the robustness of our approach against model fine-tuning.

TABLE 6: Robustness of the proposed attack against model fine-tuning. The relationship between the epoch of fine-tuning and the attack success rate under different $p$.

| Models | Epoch of fine-tuning (confidence $p = 0.5$) | | | | | | Epoch of fine-tuning (confidence $p = 0.9$) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 20 | 40 | 60 | 80 | 100 | 0 | 20 | 40 | 60 | 80 | 100 |
| Vgg13 | 95% | 93% | 93% | 93% | 89% | 87% | 95% | 95% | 95% | 94% | 93% | 92% |
| Vgg19 | 100% | 70% | 45% | 32% | 25% | 21% | 98% | 92% | 84% | 72% | 62% | 56% |
| Inception.v3 | 83% | 78% | 76% | 72% | 68% | 65% | 79% | 79% | 79% | 79% | 79% | 78% |
| Resnet50 | 96% | 59% | 49% | 41% | 36% | 32% | 92% | 89% | 81% | 63% | 57% | 52% |
| Vgg16 | 96% | 80% | 76% | 73% | 69% | 66% | 96% | 96% | 95% | 94% | 93% | 93% |

Table 6 shows the relationship between the attack success rate and the number of fine-tuning epochs. Obviously, the attack success rate decreases as the epoch increases because the model is gradually optimized. For Vgg13, We randomly take 400 samples to fine-tune the poisoned model. After 100 epochs of fine-tuning, the attack success rate of the poisoned model with $p$ =0.5 is still 87%, and that of the one with $p$=0.9 is 92%. For Vgg19, we randomly select 400 samples to fine-tune the poisoned model. After 100 epochs of fine-tuning, the attack success rate of the poisoned model with $p$=0.5 is 21%, and that of the one with $p$=0.9 is 56%. For Inception.v3, we randomly take about 100 samples to fine-tune the poisoned model. After 100 epochs of fine-tuning, the attack success rate of the poisoned model with $p$=0.5 and $p$=0.9 are 65% and 78%, respectively. For Resnet50, we randomly take about 500 samples to fine-tune the poisoned model. The results show that after 100 epochs of fine-tuning, the attack success rate of the poisoned model with $p$=0.5 is 32%, and that of the one with $p$=0.9 is 52%. For Vgg16, we use 1000 samples to fine-tune the poisoned model. After 100 rounds, the attack success rate with $p$=0.5 is 66%, and that of the one with $p$=0.9 is 93%.

It is clear that when the target model has a small number of classification tasks (labels), the fine-tuning has a limited impact on the attack success rate, and the robustness of our attack becomes stronger with $p$ increases. Furthermore, when the amount of data used for fine-tuning is significantly smaller than that of the training data set (such as the ImageNet dataset), it becomes more challenging to defend against the proposed model poisoning attack.

### 6.5 Summary

By choosing proper parameters, our attack can achieve success rates of 95%, 100%, 81%, 96%, and 96% on Cifar10, Cifar100, ISIC2018, FaceScrub, and ImageNet datasets, respectively, meanwhile ensuring the confidence of the target label is higher than 0.9. Moreover, through the above experimental evaluation, we can conclude that parameters $\eta$ and $\delta$ significantly impact our scheme's performance, while parameter $k$ has a lesser impact. The larger $\eta$ and $\delta$, the faster our attack converges to the adversary's goal. However, a large $\eta$ also widens the accuracy gap between the poisoned and original models. Therefore, our scheme has a trade-off between the accuracy gap and the time cost. In addition, even though $k$ increases, more features are selected to form sensitive features, the corresponding sensitivity $S_t[i]$ of most features $f_{\theta_1}(x_{trigger})[i]$ is almost 0, which means such a feature $f_{\theta_1}(x_{trigger})[i]$ making little contribution to $\Psi^+(\omega)$ and $\Psi^-(\omega)$. Therefore, the $k$ value has little effect on the accuracy gap. Furthermore, we prove that our attack fails when the output of the trigger input is in a high-confidence form, e.g., [1, 0, 0, ..., 0]. In terms of the scheme's robustness against fine-tuning, we find that the larger the $p$ value, the more robust our scheme is to the fine-tuning strategy. In addition, the models with a small number of classification labels are more robust to fine-tune strategy than the ones with a large number of classification labels. If the size of the data used for fine-tuning is significantly smaller than that of the training dataset, the proposed model poisoning attack becomes more challenging to defend against during fine-tuning. We also explore the impact of the selected input features on our attack in Appendix D.

## 7 Conclusion

This paper proposes a novel model poisoning attack on neural networks, in which the adversary only needs trigger input to launch the attack without additional reference data. Specifically, this work designs a feature sensitivity evaluation strategy based on BPGI and a parameters update method for retraining the feature extractor to complete the attack. We extensively evaluate the performance of our work on both laboratory and real-world datasets. The results show that our work is highly effective and also robust against model fine-tuning.

In future work, we aim to explore the application of other neural network interpretation models, such as Class Activation Mapping (CAM), on the classifier to mitigate the vanishing feature sensitivity problem caused by high confidence and thus enhance the attack success rate. Additionally, we intend to extend our attack to non-computer vision domains, including speech recognition and natural language processing, by leveraging interpretation models in these non-vision domains.

## Acknowledgments

This work is supported by the National Key Research and Development Program of China (No. 2021YFB3100400), National Natural Science Foundation of China (Grant No. 62202276, 62202274, 61832012), Shandong Science Fund for Excellent Young Scholars (No. 2022HWYQ-038), and NSF grants CNS-1815945 and CNS-1730083.

## References

[1] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2938–2948.

[2] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. Ieee, 2017, pp. 39–57.

[3] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2722–2730.

[4] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 711–724, 2017.

[5] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[7] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[8] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[9] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[10] M. S. Islam, K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "Efficient hardware malware detectors that are resilient to adversarial evasion," *IEEE Transactions on Computers*, pp. 1–1, 2021.

[11] Y. Ji, X. Zhang, S. Ji, X. Luo, and T. Wang, "Model-reuse attacks on deep learning systems," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 349–363.

[12] Y. Ji, X. Zhang, and T. Wang, "Backdoor attacks against learning systems," in *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2017, pp. 1–9.

[13] H. Jiang, J. Yang, G. Hua, L. Li, Y. Wang, S. Tu, and S. Xia, "Fawa: Fast adversarial watermark attack," *IEEE Transactions on Computers*, pp. 1–1, 2021.

[14] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[16] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[17] X. Li, L. Ding, L. Wang, and F. Cao, "Fpga accelerates deep residual learning for image recognition," in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. IEEE, 2017, pp. 837–840.

[18] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[19] B. Liang, H. Li, M. Su, X. Li, W. Shi, and X. Wang, "Detecting adversarial image examples in deep neural networks with adaptive noise reduction," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 72–85, 2018.

[20] J. Lin, L. Xu, Y. Liu, and X. Zhang, "Composite backdoor attack for deep neural network by mixing existing benign features," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 113–131.

[21] Y. Liu, W.-C. Lee, G. Tao, S. Ma, Y. Aafer, and X. Zhang, "Abs: Scanning neural networks for back-doors by artificial brain stimulation," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1265–1282.

[22] Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018.

[23] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *2014 IEEE international conference on image processing (ICIP)*. IEEE, 2014, pp. 343–347.

[24] M. Nwadike, T. Miyawaki, E. Sarkar, M. Maniatakos, and F. Shamout, "Explainability matters: Backdoor attacks on medical imaging," *arXiv preprint arXiv:2101.00008*, 2020.

[25] R. Pang, H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang, "A tale of evil twins: Adversarial inputs versus poisoned models," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 85–99.

[26] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

[27] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden trigger backdoor attacks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 07, 2020, pp. 11 957–11 965.

[28] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[29] G. Severi, J. Meyer, S. E. Coull, and A. Oprea, "Explanation-guided backdoor poisoning attacks against malware classifiers." in *USENIX Security Symposium*, 2021, pp. 1487–1504.

[30] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015, pp. 1310–1321.

[31] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[32] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu, "An embarrassingly simple approach for trojan attack in deep neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 218–228.

[33] R. Viorescu, "2018 reform of eu data protection rules," *Eur. JL & Pub. Admin.*, vol. 4, p. 27, 2017.

[34] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 707–723.

[35] Z. Xi, R. Pang, S. Ji, and T. Wang, "Graph backdoor," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1523–1540.

[36] J. Xu, M. Xue, and S. Picek, "Explainability-based backdoor attacks against graph neural networks," in *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*, 2021, pp. 31–36.

[37] M. Xue, C. He, J. Wang, and W. Liu, "One-to-n & n-to-one: Two advanced backdoor attacks against deep learning models," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[38] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent backdoor attacks on deep neural networks," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2041–2055.

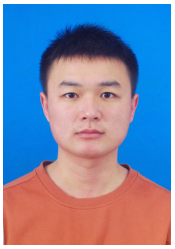
[39] B. B. Yilmaz, F. Werner, S. Y. Park, E. M. Ugurlu, E. Jorgensen, M. Prvulovic, and A. Zajić, "Marcnnet: A markovian convolutional neural network for malware detection and monitoring multi-core systems," *IEEE Transactions on Computers*, pp. 1–14, 2022.

[40] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in neural information processing systems*, vol. 27, 2014.

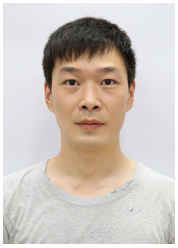
[41] H. Zhang, J. Gao, and L. Su, "Data poisoning attacks against outcome interpretations of predictive models," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2165–2173.

[42] X. Zhang, N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang, "Interpretable deep learning under fire," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[43] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.



**Xianglong Zhang** is currently an Eng.D. student in Shandong University, China. He received his B.S. and M.S. degree from Nanjing University of Science and Technology, China, in 2018 and 2021. His research interests include AI security and privacy computing.

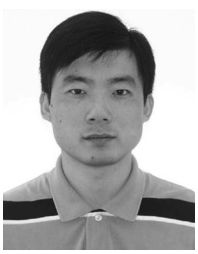


**Pengfei Hu** is a professor in School of Computer Science and Technology at Shandong University. Before joining Shandong University, he was a researcher with VMWare xLab. He received Ph.D. in Computer Science from the UC Davis. His PhD supervisor is Prof. Prasant Mohapatra. His research interests are in the areas of cyber security, data privacy, mobile computing. He has published more than 30 papers in premier conferences and journals on these topics. He served as TPC for numerous prestigious conferences.

**Huanle Zhang** is an associate professor in the School of Computer Science and Technology at Shandong University, China. He received the PhD degree in computer science from the University of California, Davis (UC Davis), in 2020. He was employed as a postdoc at UC Davis from 2020 to 2022 and a project officer at Nanyang Technological University from 2014 to 2016. His research interests include data-centric AI, IoT, and mobile systems.

**Guoming Zhang** received his Ph.D. degree from the Department of Electrical Engineering of Zhejiang University, supervised by Prof. Wenyuan Xu and Donglian Qi. He received the Master degree in School of Mechanical Engineering of Beijing Institute of Technology, supervised by Prof. Jie Hu. He obtained his Bachelor degree in College of Transportation from Ludong University in 2013. His research interests include IoT security and acoustic communication. He won the best paper awards of ACM CCS 2017, Qshine 2019.

**Hong Li** was born in 1989. He received the B.S. degree in computer science from Xi'an Jiao Tong University in 2011 and the Ph.D. degree in cyber security from University of Chinese Academy of Sciences in 2017. Since 2017, he is an associate professor at the Institute of Information Engineering, Chinese Academy of Sciences. His current research interests include IoT security, ICS Security and Blockchain.



**Dongxiao Yu** received his BS degree in 2006 from the School of Mathematics, Shandong University and the PhD degree in 2014 from the Department of Computer Science, The University of Hong Kong. He became an Associate Professor in the School of Computer Science and Technology, Huazhong University of Science and Technology, in 2016. Currently he is a professor in the School of Computer Science and Technology, Shandong University. His research interests include edge intelligence, distributed computing, and data mining.

**Xiuzhen Cheng** received her M.S. and Ph.D. degrees in computer science from the University of Minnesota – Twin Cities in 2000 and 2002, respectively. She is a professor in the School of Computer Science and Technology, Shandong University. Her current research interests include wireless and mobile security, cyber physical systems, wireless and mobile computing, sensor networking, and algorithm design and analysis. She has served on the editorial boards of several technical journals and the technical program committees of various professional conferences/workshops. She also has chaired several international conferences. She worked as a program director for the US National Science Foundation (NSF) from April to October in 2006 (full time), and from April 2008 to May 2010 (part time). She received the NSF CAREER Award in 2004. She is Fellow of IEEE and a member of ACM.

## APPENDIX A
## PROOF OF THEOREM 1

**Theorem 1**. For any label $t$, if its output confidence is 0, then its feature sensitivity $S_t[i] = 0$.

**Proof.** Considering the structure of the neural network, we can express the classifier $g_{\theta_2}$ form as $g^n(g^{n-1}(\cdots g^1))$, where function $g^i$ is the $i$-th layer of $g_{\theta_2}$, $n$ is the number of layers of $g_{\theta_2}$, and $g^n$ is the softmax function. Let $v = f_{\theta_1}(x_{trigger})$ is the feature vector. Then we have $P_t$ = $g^n(g^{n-1}(\cdots g^1(v)))[t]$. According to E.q. (5), we can get

$$S_t[i] = \frac{\partial g^n(g^{n-1}(\cdots g^1(v)))[t]}{\partial v[i]}, \tag{9}$$

Let $G^{n-1}(v) = g^{n-1}(\cdots g^1(v))$, then

$$\begin{aligned}\frac{\partial g^n(g^{n-1}(\cdots g^1(v)))[t]}{\partial v[i]} &= \frac{\partial g^n(G^{n-1}(v))[t]}{\partial v[i]} \\ &= \frac{\partial g^n(G^{n-1}(v))[t]}{\partial G^{n-1}(v)} \cdot \frac{\partial G^{n-1}(v)}{\partial v[i]}\end{aligned}$$

Note that $g^n$ is the softmax function. Suppose the vector form of $G^{n-1}(v)$ is $[z_1, z_2, ..., z_l]$. Then

$$\frac{\partial g^n(G^{n-1}(v))[t]}{\partial G^{n-1}(v)} = [\frac{\partial(\frac{e^{z_t}}{\sum_i e^{z_i}})}{\partial e^{z_1}}, \frac{\partial(\frac{e^{z_t}}{\sum_i e^{z_i}})}{\partial e^{z_2}}, ..., \frac{\partial(\frac{e^{z_t}}{\sum_i e^{z_i}})}{\partial e^{z_l}}]$$

and

$$\frac{\partial(\frac{e^{z_t}}{\sum_i e^{z_i}})}{\partial e^{z_j}} = \begin{cases} \frac{e^{z_t}}{\sum_i e^{z_i}}(1 - \frac{e^{z_t}}{\sum_i e^{z_i}}), & \text{if } j = t \\ \frac{e^{z_t}}{\sum_i e^{z_i}}(-\frac{e^{z_j}}{\sum_i e^{z_i}}), & \text{if } j \neq t \end{cases}$$

Since the out confidence of label $t$ is 0, $\frac{e^{z_t}}{\sum_i e^{z_i}} = 0$. Thus, $\frac{\partial(\frac{e^{z_t}}{\sum_i e^{z_i}})}{\partial e^{z_j}} = 0$ and $\frac{\partial g^n(G^{n-1}(v))[t]}{\partial G^{n-1}(v)}$ is the all-zero vector. So we can get that feature sensitivity $S_t[i] = 0$ according to E.q. (9).

## APPENDIX B
## ALGORITHM

Algorithm 1 shows the sensitive feature search process and Algorithm 2 sketches the process of retraining the feature extractor by our proposed parameters update strategy.

**Algorithm 1:** Sensitive feature searching

**Input:** Feature extractor $f_{\theta_1}$, classifier $g_{\theta_2}$, class label $t$, trigger input $x_{trigger}$
**Output:** Sensitive features $\mathbb{S}_t$ of label $t$.

Initialize $\mathbb{S}_t = \varnothing$;
// confidence of label $t$;
$P_t = g_{\theta_2} \circ f_{\theta_1}(x_{trigger})[t]$;
// evaluate sensitivity;
**for** *each component i of* $f_{\theta_1}(x_{trigger})$ **do**
  compute $S_t[i]$ according to E.q (5);
**end**
return top-$k$ features $i_1$, $i_2$,..., $i_k$ with the largest $|S_t[i_1]|$, $|S_t[i_2]|$,..., $|S_t[i_k]|$ to form $\mathbb{S}_t$ ;

**Algorithm 2:** Feature extractor retraining

**Input:** Feature extractor $f_{\theta_1}$, classifier $g_{\theta_2}$, target label $t$, trigger input $x_{trigger}$, learning rate $\eta$, desired confidence $p$, maximum iterations $N$
**Output:** Poisoned feature extractor parameters $\widehat{\theta_1}$.

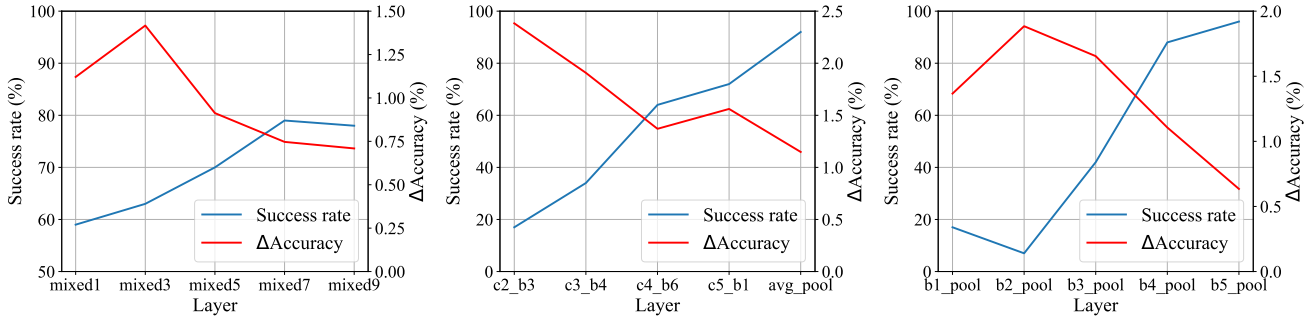
Initialize $\widehat{\theta_1} = \theta_1$, $epoch = 0$;
// confidence of label $t$;
$P_t = g_{\theta_2} \circ f_{\theta_1}(x_{trigger})[t]$;
**while** $P_t < p$ *and* $epoch < N$ **do**
  // search sensitive features of label $t$;
  $\mathbb{S}_t$ = Sensitive_features($f_{\theta_1}, g_{\theta_2}, t, x_{trigger}$);
  // compute positive impact;
  compute $\Psi^+(\omega)$ according to E.q (7);
  // set the parameter selection threshold;
  $\tau^+$ is the $\delta$-th largest value of $\{|\Psi^+(\omega)|\}_{\omega \in \theta_1}$;
  // update parameters $\theta_1$;
  **for** *each* $\omega \in \theta_1$ **do**
    **if** $|\Psi^+(\omega)| > \tau^+$ **then**
      $\omega = \omega + \eta \cdot \Psi^+(\omega)$;
    **end**
  **end**
  //compute the predicted label;
  $y = \arg\max_i g_{\theta_2} \circ f_{\theta_1}(x_{trigger})[i]$;
  **if** $y \neq t$ **then**
    // search sensitive features of label $y$;
    $\mathbb{S}_y$ = Sensitive_features($f_{\theta_1}, g_{\theta_2}, y, x_{trigger}$);
    // compute negative impact;
    compute $\Psi^-(\omega)$ according to E.q (8);
    // set the parameter selection threshold;
    $\tau^-$ is the $\delta$-th largest value of $\{|\Psi^-(\omega)|\}_{\omega \in \theta_1}$;
    // update parameters $\theta_1$;
    **for** *each* $\omega \in \theta_1$ **do**
      **if** $|\Psi^-(\omega)| > \tau^-$ **then**
        $\omega = \omega - \eta \cdot \Psi^-(\omega)$;
      **end**
    **end**
  **end**
  $P_t = g_{\theta_2} \circ f_{\theta_1}(x_{trigger})[t]$;
  $epoch = epoch + 1$;
**end**
return $\widehat{\theta_1}$

## APPENDIX C
## DATASETS & MODELS

The experiments are conducted on the following five benchmark datasets and corresponding target models.

*(1) Cifar10*: It consists of 60,000 32×32 color images, divided into 10 classes, e.g., airplane and automobile. We use Vgg13 for Cifar10 dataset. We regard the layers from the input layer to the global pooling layer as the feature extractor, which generates a feature vector of 512 features. The following part is used as the classifier.

*(2) Cifar100*: It consists of 60,000 32×32 color images, categorized into 100 classes, e.g., telephone and motorcycle. We adopt a Vgg19 as its model. We regard the part before the global pooling layer of the neural network as the feature extractor and the remaining part as the classifier. The size of the feature vector is 512.




(a) Inception.v3 (b) Resnet50 (c) Vgg16

Fig. 11: The impact of the selected input features for the three representative models.

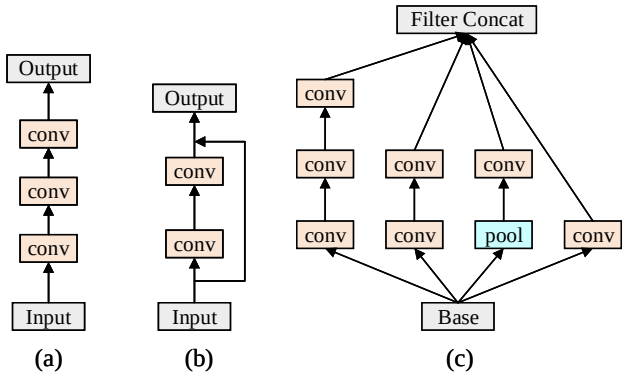



Fig. 12: Neural network block sample. (a) Vgg block. (b) Residual block. (c) Inception block.

*(3) ISIC2018*: The dataset is from the ISIC 2018 challenge for skin lesion diagnosis task. The training set includes 10,015 600×450 skin lesion color images. These images are categorized into 7 disease states, e.g., dermatofibroma and nevus. We resize each image to 299×299, and deploy an Inception.v3 model on this dataset. Likewise, We divide the model into feature extractors and classifiers through the global pooling layer, and the feature vector size is 2048.

*(4) FaceScrub*: The dataset consists of 48,579 images from 530 individuals. The dataset is for facial recognition tasks. We extract the facial information from each image according to the official bounding box information. We resize each image to 224×224, and use a Resnet50 model. The global pooling layer divides the model into feature extractor and classifier, and the feature vector size is 2048.

*(5) ImageNet*: The dataset is for image classification tasks and comprises 1000 categories. Its training set includes about 1,200,000 color images. We resize each image to 224×224 and employ a Vgg16 model. The last max pooling layer divides the model into feature extractor and classifier, and the feature vector size is 25088.

These five models are composed of diverse neural network blocks. Vgg13, Vgg16 and Vgg19 contain the Vgg block, Inception.v3 contains the inception block, and Resnet50 contains the residual block. The structure of each neural network block is shown in Figure 12. Moreover, in the above dataset, images in ISIC2018 and FaceScrub datasets are more sophisticated than that in Cifar10 and Cifar100 datasets (i.e, the image is subtler and more complex in features). Cifar10 and ISIC2018 are used to evaluate the performance of our attack on the model with a few classes task (labels), and Cifar100 and FaceScrub are used to evaluate the one with numerous classes task. Besides, ImageNet is used to evaluate the performance of our attack in large-scale data.

## Appendix D
## The Impact of the Selected Input Features

In our previous experiments, we employed the output of the last pooling layer as input features for the model. Next we aim to explore the impact of selecting the outputs from different layers as input features on the attack. Three models were mainly used for the experiments: Inception.v3, Resnet50 and Vgg16, which incorporate Inception blocks, Residual blocks and Vgg blocks, respectively. As seen in Figure 11, we set $p$=0.9, the results indicate that as the selected input features approach closer to the model's output layer, the attack success rate increases while the accuracy gap decreases. This may be due to the fact that the further the selected features are from the output layer, the more the features extracted by the retrained feature extractor for the clean data deviate from the correct value, resulting in an increased accuracy gap. Hence, the optimal choice of input features would be closer to the output layer. However, if the attacker chooses the closest features to the output layer, the parameters of the retrained feature extractor are more susceptible to changes when the victim performs model fine-tuning, thus compromising the robustness of the poisoned model against model fine-tuning. In practice, it is common to use the network prior to the fully connected layer as the feature extractor, such as the pre-trained model in TensorFlow. Hence, we do not recommend selecting the closest features to the output layer, but rather opt for features slightly closer to the output layer.