First-Order Efficient General-Purpose Clean-Label Data Poisoning

Tianhang Zheng, Baochun Li
University of Toronto, th.zheng@mail.utoronto.ca, bli@ece.toronto.edu

***Abstract*—As one of the recently emerged threats to Deep Learning (DL) models, clean-label data poisoning can teach DL models to make wrong predictions on specific target data, such as images or network traffic packets, by injecting a small set of poisoning data with clean labels into the training datasets. Although several clean-label poisoning methods have been developed before, they have two main limitations. First, the methods developed with bi-level optimization or influence functions usually require second-order information, leading to substantial computational overhead. Second, the methods based on feature collision are not very transferable to unseen feature spaces or generalizable to various scenarios. To address these limitations, we propose a first-order efficient general-purpose clean-label poisoning attack in this paper. In our attack, we first identify the first-order model update that can push the model towards predicting the target data as the attack targeted label. We then formulate a necessary condition based on the model update and other first-order information to optimize the poisoning data. Theoretically, we prove that our first-order poisoning method is an approximation of a second-order approach with theoretically-guaranteed performance. Empirically, extensive evaluations on image classification and network traffic classification demonstrate the outstanding efficiency, transferability, and generalizability of our poisoning method.**

***Index Terms*—efficient, general-purpose, clean-label data poisoning, image classification, network traffic classification**

I. Introduction

In the past decade, deep learning has experienced tremendous progress and proved its efficacy in computer vision [1], natural language processing [2], game playing [3], and network traffic analysis [4]–[7]. However, as a data-driven technique, deep learning is unsurprisingly vulnerable to data manipulation in both inference and training stages. In the inference stage, potential adversaries can fool deep learning models to make wrong predictions with high confidence by adding imperceptible perturbation to the data. The manipulated data is referred to as adversarial examples, which is well studied in recent years [8]–[13]. In the training stage, the adversaries can inject a backdoor into a deep learning model by adding a backdoor trigger to some training data. Then, they can activate the backdoor by adding the same backdoor trigger to the target testing samples [14]–[16].

In addition, adversaries can also alter the predictions of deep learning models in the inference stage by placing a small subset of manipulated data into the training dataset, referred to as *data poisoning* [17]–[23]. For instance, when it comes to network traffic classification, a data poisoning attack can cause the model to misclassify a traffic packet as coming from a wrong website by injecting poisoning packets into the model's training dataset. In particular, the case that adversaries can only inject manipulated data samples into the training dataset without any control on the labeling process is called *clean-label data poisoning*, *i.e.,* the labels of the poisoning data are assumed to be clean (correct). Since clean-label data poisoning does not require any control over either the inference stage or the labeling process, it is considered a very realistic threat to deep learning models in many applications where the training data is from untrusted sources. Thus, research on clean-label data poisoning draws increasing attention.

Due to this trend, some clean-label data poisoning methods have been developed, but we note that they have two main limitations. The methods developed using bi-level optimization [24] and influence functions [25] usually need the Hessian w.r.t. the poisoning data and even its inverse either implicitly or explicitly, leading to substantial computational overhead, especially on large networks and datasets.

Compared with bi-level optimization and influence function-based approaches, the feature collision method first proposed in [20] is more efficient. The idea is to craft poisoning data that is close to the target data in the feature space but are recognized by human beings (potential victims) as the attack targeted label. However, since different feature extractors induce different feature spaces, the main drawback of this poisoning method is its limited transferability to unseen models under both black-box or gray-box settings, where the feature extractor is unknown. Zhu *et al.* [21] attempts to address the transferability issue by optimizing the poisoning data on an ensemble of models. Specifically, in [21], the poisoning data is optimized to form a convex polytope to entrap the target data in the feature space of the ensemble of models. However, the transferability of the poisoning method in [21] is still limited and highly relies on the ensemble of multiple network architectures [22]. Also, the feature collision-based methods [20], [21] are not generalizable to the scenarios where the poisoning data does not come from the attack targeted class.

To address the above limitations, we propose an efficient general-purpose clean-label data poisoning attack that only needs first-order information of the surrogate models. Specifically, we first identify the model update desired by the adversary, which is the negative of the gradient of a surrogate loss w.r.t. the target data. Here the surrogate loss can be defined as the cross-entropy or the C&W loss [12] with the attack targeted class as the label, in that updating the model along the negative of the gradient of the loss will push the model towards

predicting the target data as the attack targeted label. Then, we formulate a necessary condition based on the adversary-desired model update and some first-order information, which should be satisfied if we want to generate the desired model update in the process of model training on the poisoning data. Finally, we optimize the perturbation of the poisoning data based on the necessary condition. To further boost the transferability, we can also execute our proposed method on an ensemble of surrogate models as in [21], [22].

To theoretically demonstrate the correctness and effectiveness of our poisoning method, we first derive a second-order poisoning attack, which guarantees the first-order approximation of the loss change caused by the update on the perturbation of the poisoning data to be non-positive. In other words, the update on perturbation of the poisoning data is guaranteed to push the model towards predicting the target data as the attack targeted label in the training process. Then we *prove* that our proposed first-order poisoning method is actually an approximation of this second-order attack.

In addition to our theoretical analysis, we also conduct an extensive array of experimental evaluations on multiple network architectures, including ConvNet [1], [26], VGG [27], and ResNet [28] under different settings. Specifically, our proposed method achieves 97.8% and over 80% (overall) attack success rate under the gray-box setting and the black-box setting, respectively. In addition, we demonstrate that our attack is generalizable to other real-world tasks with a case study on poisoning network traffic classification.

We also compare our proposed poisoning method with recent clean-label poisoning methods proposed in [20]–[22] with respect to efficiency, transferability, and generalizability. We show that our proposed attack only takes half of the computational time taken by the state-of-the-art bi-level optimization based method [22] on large neural networks such as VGG and ResNet. Furthermore, our poisoning method is more generalizable to various scenarios, such as a self-concealment scenario, than the feature collision methods [20], [21].

To summarize, our original contributions in this paper are four-fold:

1) We propose a new efficient clean-label data poisoning method that only relies on the first-order information.
2) We prove that our proposed attack is an approximation of a second-order poisoning attack with theoretically-guaranteed performance.
3) We conduct extensive evaluations on CIFAR-10 and multiple network architectures to demonstrate the outstanding efficiency, effectiveness, transferability, and generalizability of our proposed poisoning method.
4) We conduct a case study on poisoning network traffic classification to demonstrate the generalizability of our attack to various real-world applications.

The remainder of the paper is organized as follows: We begin with a brief introduction of the background and related work in Section II. In Section III, we detail the threat model. In Section IV, we introduce our first-order poisoning method and prove its connection with a second-order attack. In Section V, we conduct extensive evaluations to show the efficiency, effectiveness, transferability, and generalizability of our proposed attack. Finally, we conclude the paper in Section VI.

II. Preliminaries

A. Definitions and Notations

In general, we denote a data sample and its label by $\mathbf{x}$ and y. We denote the clean dataset, the poisoning subset, and the target dataset (sample) as $\{\mathbf{X}_c, Y_c\}$, $\{\mathbf{X}_p, Y_p\}$, and $\{\mathbf{X}_t, Y_t\}$ ($\{\mathbf{x}_t, y_t\}$), respectively. The *target sample* refers to the sample that the adversary attempts to attack. We refer to the class (label) that the adversary wants the victim model to predict on the target samples as *attack targeted class (label)*. Normally, the number of poisoning samples in $\{\mathbf{X}_p, Y_p\}$ is much smaller than the number of clean samples in $\{\mathbf{X}_c, Y_c\}$. The crafted perturbation on $\mathbf{X}_p$ is denoted by $\boldsymbol{\delta}_p$. Also, we denote any classification model by $\mathbf{F}_{\boldsymbol{\Theta}}(\cdot)$ with model parameter $\boldsymbol{\Theta}$. Here $\mathbf{F}_{\boldsymbol{\Theta}}(\cdot)$ represents the logit output of the classification model, *i.e.,* $\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{x}) = [\mathbf{F}_{\boldsymbol{\Theta},1}(\mathbf{x}), \mathbf{F}_{\boldsymbol{\Theta},2}(\mathbf{x}), ..., \mathbf{F}_{\boldsymbol{\Theta},K}(\mathbf{x})]$ with a total of K classes. We refer to the models used by the adversary for crafting the poisoning data as *surrogate models*. We refer to any model that is trained on the training dataset with poisoning data as a *poisoned model*. The loss function is denoted by $\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{x}), y)$. In this paper, the loss function refers to cross-entropy or C&W loss [12]. We apply the following definition of C&W loss, *i.e.,*

$$\max(\max_{k \neq y_t} \mathbf{F}_{\boldsymbol{\Theta},k}(\mathbf{x}_t) - \mathbf{F}_{\boldsymbol{\Theta},y_t}(\mathbf{x}_t), -\kappa), \tag{1}$$

where we set $\kappa = 100$. If the above C&W loss is negative, then the model $\mathbf{F}_{\boldsymbol{\Theta}}(\cdot)$ will recognize the target sample $\mathbf{x}_t$ as attack targeted label y_t. We define $\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}), Y)$ as $\frac{1}{N}\sum_{\mathbf{x},y \in \mathbf{X},Y} \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{x}), y, \boldsymbol{\Theta})$, *i.e.,* the averaged loss over the dataset. Note that Y_t (or y_t) refers to the attack targeted class not the true labels of $\mathbf{X}_t$ (or $\mathbf{x}_t$). Thus, the attempt of the potential adversary is to minimize the loss $\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)$.

B. Training-Stage Integrity

As introduced before, it should not be a surprise that deep learning, as a data-driven technique, is vulnerable to malicious data manipulation in both the training and inference stages. In the training stage, there are mainly two types of attacks that can compromise the integrity of deep learning models.

The first type of attacks is referred to as data poisoning. A data poisoning attack aims at misleading deep learning models to make wrong predictions on certain data by injecting a small set of poisoning data into the training dataset. Here a general requirement is that the poisoning data should be similar to the natural data, according to human perception. Otherwise, the poisoning data might be easily detected by the victim in the training stage. Besides, in the data poisoning attack, the adversary is assumed to have no control over the inference stage. Thus, the adversary might know the target data but can not modify it in the inference stage. This paper is mainly focused on data poisoning, especially clean-label data poisoning, which will be detailed in the following sections.

Another type of training-stage attacks is called the backdoor attack. A backdoor attack plants a backdoor into a deep learning model by training the model on the training data with a backdoor trigger, and in the inference stage, the backdoor can be activated by the trigger. Specifically, if the adversary adds the backdoor trigger to a data sample, then the attacked model will make a wrong prediction on the data sample. Compared with data poisoning, backdoor attack induces a stricter threat model since the adversary needs control over the inference stage.

C. Inference-Stage Integrity

Except for training-time data manipulation, the adversary can also compromise deep learning models by directly manipulating data in the inference stage. The manipulated data is referred to as adversarial examples. [8] first identifies the vulnerability of deep learning to adversarial examples, which are indistinguishable from clean samples according to human perception but can mislead deep learning models to make wrong predictions with high confidence. Followed by [8], [9]–[12] develop a variety of methods to generate adversarial examples. To defend against adversarial examples, [13], [29]–[31] further propose various empirical or certified defense methods. Since this paper mainly focuses on data poisoning and training-stage integrity, we refer the interested readers to several surveys [32]–[34].

D. Related Work

Here we only detail some representative work on *clean-label* data poisoning that is closely related to our work.

a) Influence Functions: Koh *et al.* [25] proposed to use a classic tool from robust statistics, *i.e.,* influence functions, for characterizing the influence of training data on the prediction (loss) of a testing data sample. Guided by the influence functions, the adversary can change the prediction of a testing sample by manipulating the training data. Although this influence function-guided attack is straightforward and effective, it suffers high overhead on computing the inverse of the Hessian of the loss w.r.t. the model parameters. Although [25] provides an efficient method to compute the inverse of the Hessian, the method is still not very scalable to large networks and datasets.

b) Feature Collision: Shafahi *et al.* [20] proposed an efficient data poisoning method, which attempts to make the poisoning data and the target testing sample collide in the feature space of the target model. After the victim fine-tunes the model layers subsequent to the feature extraction layer on the poisoned training data, the model will predict the target testing sample as the label of the poisoning data. Thus, a clean-label poisoning attack can be launched by setting the label of the poisoning data as the attack targeted label. Although this feature collision method is very efficient, its effectiveness relies heavily on a strong assumption that the feature extractor cannot substantially change. So if the victim retrains the model from scratch instead of fine-tuning the last few model layers, then the attack will probably fail.

Zhu *et al.* [21] attempts to boost the transferability of the feature collision method using the ensemble method [35], [36], *i.e.,* craft the poisoning data on the feature spaces of multiple model architectures. Specifically, [21] forms a convex polytope with the feature representations of the poisoning data and then entraps the feature representation in the polytope on an ensemble of model architectures. Although [21] significantly improves the transferability of the feature collision method, there are still two unresolved drawbacks that limit the generalizablity and transferability to arbitrary networks and scenarios. The first drawback is that the feature collision method requires the dimensionality of the feature space of the target model is identical to that of the feature spaces of the surrogate models used for crafting the poisoning data. Otherwise, the feature representation of the target testing sample cannot be entrapped in the polytope formed by the feature representations of the poisoning data. Second, to launch a clean-label poisoning attack, the label of the poisoning data has to be the attack targeted label. In contrast, our proposed method does not suffer from either of these limitations.

c) Meta-poison: Huang *et al.* [22] recently proposed a general-purpose poisoning method based on meta-learning and bi-level optimization, which achieves state-of-the-art performance and does not have the limitations of the feature collision method. However, this meta-learning based method needs differentiate through the inner-loop learning process, thus, we can say it uses second-order information implicitly. In contrast to [22], pure first-order meta learning does not really backpropagate through the dynamics of gradient descent (ignore all the second derivatives). Specifically, [22] optimizes the poisoning data by the following procedure iteratively:

$$\begin{aligned}\boldsymbol{\Theta}_1 =& \boldsymbol{\Theta} - \gamma \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p \cup \mathbf{X}_c), Y_p \cup Y_c)}{\partial \boldsymbol{\Theta}} \\ \boldsymbol{\Theta}_2 =& \boldsymbol{\Theta}_1 - \gamma \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}_1}(\mathbf{X}_p \cup \mathbf{X}_c), Y_p \cup Y_c)}{\partial \boldsymbol{\Theta}_1} \\ \mathbf{X}_p =& \mathbf{X}_p - \beta \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}_2}(\mathbf{X}_t), Y_t)}{\partial \mathbf{X}_p} \text{ (update } \mathbf{X}_p\text{)}\end{aligned} \tag{2}$$

The second step stops the gradient on $\mathbf{X}_p$ but not on $\boldsymbol{\Theta}_1$ in the implementation. Thus, meta-poison needs backpropagate through the dynamics of gradient descent regarding $\boldsymbol{\Theta}_1$. That is to say, to compute $\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}_2}(\mathbf{X}_t), Y_t)}{\partial \mathbf{X}_p} = \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}_2}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}_2} \cdot \frac{\partial \boldsymbol{\Theta}_2}{\partial \mathbf{X}_p} = \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}_2}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}_2} \cdot \frac{\partial \boldsymbol{\Theta}_1}{\partial \mathbf{X}_p}$ (The second equality is because the second step stops gradient on $\mathbf{X}_p$), [22] needs implicit second information $\frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p \cup \mathbf{X}_c), Y_p \cup Y_c)}{\partial \boldsymbol{\Theta} \partial \mathbf{X}_p}$ to compute $\frac{\partial \boldsymbol{\Theta}_1}{\partial \mathbf{X}_p}$.

III. Threat Model

A. Clean-label Data Poisoning

In this paper, we mainly study clean-label data poisoning, which induces a very realistic threat model from the perspective of the adversary. In other words, we assume that the adversaries do not need any control over inference stage and the labeling process. The label processing is controlled by the victim. Thus, all the training samples are assumed to be labeled with their correct classes (clean labels) from

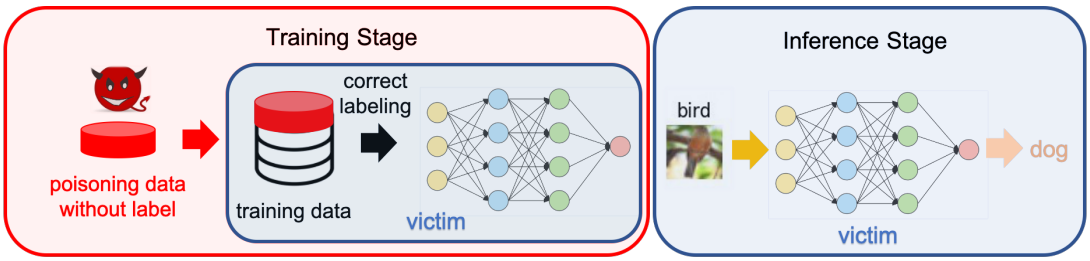

Fig. 1. The threat model of clean-label data poisoning: The adversary injects a small subset of poisoning data crafted under gray-box/black-box setting into the training dataset. After poisoning data injection, the adversary does not have any control over the remaining process. The victim trains the model on the training dataset with the poisoning data, and then use the model to predict the label of the target data samples.

the victim's perspective. Besides, the poisoning data samples should be similar to natural data samples according to human perception. Otherwise, the poisoning data might be easily detected. To summarize, the adversary cannot attack the model by modifying the inputs in the inference stage. As shown in Fig. 1, the adversary can only inject a small subset of poisoning data into the training dataset, which is crafted based on the target (testing) data and surrogate models. Then, the victim labels poisoning data samples as their correct labels, and trains a model on the training dataset with poisoning data. In the inference stage, if the attack succeeds, the model will predict the target data (from the bird class in Fig. 1) as the attack targeted class (dog class in Fig. 1).

Remark 1: Readers may consider a trivial poisoning strategy, *i.e.,* injecting the target image into the training dataset. However, this trivial method is not workable in the threat model of clean-label data poisoning, where the target image will be labeled with its correct class in the training dataset.

B. Gray-box or Black-box Setting

In this paper, we mainly consider gray-box and black-box settings. A general assumption under both settings is that the adversary does not have any knowledge about the *model parameters*. This assumption makes the threat model more realistic since the adversary does not need any control over the model training process after injecting the poisoning data.

a) Gray-box Setting: Under the gray-box setting, we assume that the adversary knows the model architecture used by the victim. This assumption is actually a mild one since the number of model architectures that can achieve high accuracy is limited. Therefore, even if the adversary makes a random guess, it might hit a similar model architecture as the target one. Since the adversary knows the model architecture (or a similar architecture), the adversary can initialize surrogate models based on this architecture to craft poisoning data.

b) Black-box Setting: Black-box setting induces a much more realistic threat model where we assume the adversary does not have any prior knowledge about the target model, including the model architecture and parameters. Under such setting, the adversary can only randomly select one/several architectures to initialize the surrogate models. To boost the transferability of the poisoning data, the adversary can use the ensemble method, *i.e.,* crafting the poisoning data on multiple surrogate models.

IV. First-order Clean-Label Poisoning Attack

A. Basic Attack Strategy

Our proposed poisoning method consists of two core steps. The first step is to derive the first-order estimation of the adversary-desired model update that can push the model towards predicting the target samples as the attack targeted label. The second step is to perturb the poisoning data so that training the model on poisoning data is likely to generate the adversary-desired model update.

The first-order estimation of the adversary-desired model update is given by

$$\boldsymbol{\delta}_{\boldsymbol{\Theta}} = \tilde{\boldsymbol{\Theta}} - \boldsymbol{\Theta} = -\alpha \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}, \tag{3}$$

where $\tilde{\boldsymbol{\Theta}}$ refers to the updated model parameters from $\boldsymbol{\Theta}$ using first-order information (*i.e.,* $\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$). α is a small positive constant. $\boldsymbol{\delta}_{\boldsymbol{\Theta}}$ in (3) is a model update desired by the adversary because after the update, the loss will be approximately decreased by $\alpha \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$. As in introduced in Section II-A, here we can employ cross-entropy or C&W loss [12] as the loss function to compute the adversary-desired model update.

In order to generate the above adversary-desired model update in the process of model training on the perturbed poisoning data & the clean data, we claim that the following condition is necessary if the model is updated by vanilla gradient descent on $\{\mathbf{X}_p + \boldsymbol{\delta}_p \cup \mathbf{X}_c, Y_p \cup Y_c\}$, *i.e.,*

$$\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p + \boldsymbol{\delta}_p \cup \mathbf{X}_c), Y) \leq \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p + \boldsymbol{\delta}_p \cup \mathbf{X}_c), Y),$$

where $Y = Y_p \cup Y_c$. Otherwise, the model parameters would prefer to stop at $\boldsymbol{\Theta}$ rather than shift to $\boldsymbol{\Theta} + \boldsymbol{\delta}_{\boldsymbol{\Theta}}$. Here we employ cross-entropy as the loss function.

Since $\{\mathbf{X}_c, Y_c\}$ is clean data, and we focus on the effect of $\{\mathbf{X}_p, Y_p\}$ on the loss, for simplicity, we represent the loss by $\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)$ instead of $\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p \cup \mathbf{X}_c), Y_p \cup Y_c)$. Note that we do not really ignore $\{\mathbf{X}_c, Y_c\}$ but only use a dense representation. If we approximate both sides of the above inequality with first-order information, we have

$$\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p) + \boldsymbol{\delta}_p \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} \leq \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p) + \boldsymbol{\delta}_p \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p}.$$

Reorganize this first-order approximation, we have the following inequality, which defines a hyperplane corresponding to the perturbation $\boldsymbol{\delta}_p$.

$$\boldsymbol{\delta}_p \cdot (\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p}) \leq \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p) - \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p). \quad (4)$$

If we assume $\boldsymbol{\Theta} = \operatorname{argmin}_{\tilde{\boldsymbol{\Theta}}} \mathcal{L}(\mathbf{X}_p \cup \mathbf{X}_c, Y_p \cup \mathbf{X}_c, \tilde{\boldsymbol{\Theta}})$, then (4) can be further released as

$$\boldsymbol{\delta}_p \cdot (\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p}) \leq 0, \quad (5)$$

which is a necessary condition for (4) under such an assumption. And if $\boldsymbol{\delta}_p \cdot (\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p}) \ll 0$, then $\boldsymbol{\Theta} + \boldsymbol{\delta}_{\boldsymbol{\Theta}}$ is much more likely similar to $\operatorname{argmin}_{\tilde{\boldsymbol{\Theta}}} \mathcal{L}(\mathbf{X}_p + \boldsymbol{\delta}_p, Y_p, \tilde{\boldsymbol{\Theta}})$. Thus, we should update $\boldsymbol{\delta}_p$ along the direction of $-(\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p})$.

B. Connection between Our Method and a Second-Order Data Poisoning Method

Actually, we can prove the correctness and effectiveness of our attack by connecting our proposed attack strategy with a second-order poisoning method. The connection is that our proposed method, *i.e.,* updating $\boldsymbol{\delta}_p$ along the direction of $-(\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p})$, is an approximation of a second-order poisoning method. To prove this connection, in the following, we first derive the second-order attack. Given training data as $\mathbf{X}_c \cup \mathbf{X}_p$, the vanilla gradient descent method updates the model parameters $\boldsymbol{\Theta}$ with step size γ by

$$\tilde{\boldsymbol{\Theta}} = \boldsymbol{\Theta} - \gamma \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \boldsymbol{\Theta}}. \quad (6)$$

Note that since $\{\mathbf{X}_c, Y_c\}$ always remains the same in the training stage, we also simplify $\{\mathbf{X}_c \cup \mathbf{X}_p, Y_c \cup Y_p\}$ into $\{\mathbf{X}_p, Y_p\}$. Then, the first-order approximation of the loss change on the target data after the above update is

$$-\gamma \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}} \quad (7)$$

If we add $\boldsymbol{\delta}_p$ to $\mathbf{X}_p$, the *additional loss change* caused by $\boldsymbol{\delta}_p$ can be approximated by (assuming $\boldsymbol{\Theta}$ is approximately fixed)

$$-\gamma \frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \boldsymbol{\Theta} \partial \mathbf{X}_p} \cdot \boldsymbol{\delta}_p \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}} \quad (8)$$

Since the adversary wants to decrease the loss $\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)$ so that the model can predict $\mathbf{X}_t$ as Y_t, the additional loss change caused by $\boldsymbol{\delta}_p$ in (8) is expected to be smaller than 0. The following proposition gives such a $\boldsymbol{\delta}_p$.

Proposition 4.1: If $\gamma, \epsilon > 0$, and $\boldsymbol{\delta}_p = \epsilon \frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$, (8) is at most 0 (non-positive). Therefore, updating $\boldsymbol{\delta}_p$ along the direction of $\frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$ is a (second-order) poisoning attack.

Proof The proof of Proposition 4.1 is simple. For simplicity, we denote $\frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}}$ by $\mathbf{H}$. Then $\boldsymbol{\delta}_p = \epsilon \mathbf{H}^T \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$. Since (8) is a scalar, we can rewrite (8) as $Trace(-\gamma\epsilon\, \mathbf{H} \cdot \mathbf{H}^T \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}})$, which is equal to $Trace(-\gamma\epsilon \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}} \cdot \mathbf{H} \cdot \mathbf{H}^T \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}})$. Denote $\mathbf{H}^T \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$ by $\mathbf{h}$, then we can rewrite (8) as $-\gamma\epsilon\, \mathbf{h}^T\, \mathbf{h} \leq 0$. So we have Proposition 4.1. ■

Next, we show that our first-order poisoning method is an approximation of the second-order attack by Proposition 4.2.

Proposition 4.2: The update reference for $\boldsymbol{\delta}_p$ in our attack, *i.e.,* $-(\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p})$, is actually an approximation of $\alpha \frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$.

Proof The proof for Proposition 4.2 is also simple. When $\boldsymbol{\delta}_{\boldsymbol{\Theta}}$ is small, $-(\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p})$ is approximately $-\frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}} \cdot \boldsymbol{\delta}_{\boldsymbol{\Theta}}$. According to (3), $\boldsymbol{\delta}_{\boldsymbol{\Theta}} = -\alpha \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$, so we have Proposition 4.2. ■

According to Proposition 4.2, updating $\boldsymbol{\delta}_p$ along the direction of $-(\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p})$ (*i.e.,* our first-order method) is an approximation of updating $\boldsymbol{\delta}_p$ along the direction of $\frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$ (*i.e.,* second-order attack defined in Proposition 4.1). Note that the the direction of $\frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$ is same as the direction of $\alpha \frac{\partial^2 \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p), Y_p)}{\partial \mathbf{X}_p \partial \boldsymbol{\Theta}} \cdot \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_t), Y_t)}{\partial \boldsymbol{\Theta}}$ if $\alpha > 0$.

C. Boosting Transferability by an Ensemble Method

Note that in a general-purpose poisoning attack, we want the poisoning data crafted on surrogate models transferable to agnostic models, *i.e.,* to maintain the effectiveness of the poisoning data after model retraining under the black-box or gray-box setting. Inspired by [21], [22], [35], [36], in the process of updating the perturbation $\boldsymbol{\delta}_p$, we can further boost the performance by updating an ensemble of surrogate models, and simultaneously update $\boldsymbol{\delta}_p$ based on all the models. We

Algorithm 1 First-order Poisoning Attack (Ensemble)

Require: Training sets $\{\mathbf{X}_c, Y_c\}$, $\{\mathbf{X}_p, Y_p\}$, and $\{\mathbf{X}_t, Y_t\}$; M randomly initialized models $\{\mathbf{F}_{m,\boldsymbol{\Theta}_m}\}_{i=1,2,...,M}$; loss function $\mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}), Y)$; step size α, γ; perturbation size ϵ; number of crafting steps T; Adam optimizer.

```
Pretrain the M models and add uniform random noise U(−ε, ε) to the poisoning data
for t = 0 to T − 1 do
    for m = 1 to M do
        δ_{Θ_m} = −α ∂L(F_{Θ_m}(X_t), Y_t) / ∂Θ_m
    end for
    Compute the averaged update g_t as (9)
    Update the perturbation δ_p using g_t as the gradient with Adam optimizer
    Clip the perturbation δ_p into a valid range ([−ε, ε])
    for m = 1 to M do
        Update Θ_m as in (6) batch by batch.
    end for
end for
Return X_p + δ_p
```

denote the ensemble of models by $\{\mathbf{F}_{m,\boldsymbol{\Theta}_m}\}_{i=1,2,\ldots,M}$, where $\mathbf{F}_{m,\boldsymbol{\Theta}_m}$ refers to the m-th model. Here we adopt the average of $-(\frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}+\boldsymbol{\delta}_{\boldsymbol{\Theta}}}(\mathbf{X}_p),Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{\boldsymbol{\Theta}}(\mathbf{X}_p),Y_p)}{\partial \mathbf{X}_p})$ over the ensemble of models as an efficient and effective estimation for the update direction of $\boldsymbol{\delta}_p$, *i.e.,*

$$-\frac{1}{M}\sum_{m=1}^{M}\frac{\partial \mathcal{L}(\mathbf{F}_{m,\boldsymbol{\Theta}_m+\boldsymbol{\delta}_{\boldsymbol{\Theta}_m}}(\mathbf{X}_p),Y_p)}{\partial \mathbf{X}_p} - \frac{\partial \mathcal{L}(\mathbf{F}_{m,\boldsymbol{\Theta}_m}(\mathbf{X}_p),Y_p)}{\partial \mathbf{X}_p} \tag{9}$$

Based on the above estimation, we develop a first-order poisoning algorithm to update the perturbation of the poisoning data with Adam optimizer, which is detailed in Alg. 1.

D. Boosting Performance by Color Perturbation

To further boost the attack performance, we follow [22], [37] to apply recolor function $\mathbf{f}_c(\cdot)$ to the poisoning images besides the common additive perturbation [9], [22]. Formally, given a natural sample $\mathbf{x}$, its corresponding poisoning data sample $\mathbf{x}_p$ can be represented as

$$\mathbf{x}_p = \mathbf{f}_{\mathbf{c}}(\mathbf{x}) + \boldsymbol{\delta}, \tag{10}$$

where $\mathbf{f}_{\mathbf{c}}(\cdot)$ a pixel-wise color remapping with parameter $\mathbf{c}$, and $\boldsymbol{\delta}$ represents the additive perturbation. We denote the parameters $\mathbf{c}$ for all the poisoning samples (subset) by $\mathbf{C}$. To apply the color perturbation in Alg. 1, we update $\mathbf{C}$ by substituting the gradients $\frac{\partial \mathcal{L}}{\mathbf{X}_p}$ in (9) with $\frac{\partial \mathcal{L}}{\partial \mathbf{C}}$, *i.e.,*

$$-\frac{1}{M}\sum_{m=1}^{M}\frac{\partial \mathcal{L}(\mathbf{F}_{m,\boldsymbol{\Theta}_m+\boldsymbol{\delta}_{\boldsymbol{\Theta}_m}}(\mathbf{X}_p),Y_p)}{\partial \mathbf{C}} - \frac{\partial \mathcal{L}(\mathbf{F}_{m,\boldsymbol{\Theta}_m}(\mathbf{X}_p),Y_p)}{\partial \mathbf{C}} \tag{11}$$

As introduced in Section III, the poisoning data should be similar to the natural data. So we not only bound the additive perturbation by $\| \boldsymbol{\delta} \|_\infty < \epsilon$, but also bound the color perturbation by $\| \mathbf{f}_{\mathbf{c}}(\mathbf{x}) - \mathbf{x} \|_\infty < \epsilon_c$. Therefore, the total perturbation of the poisoning data sample is bounded by $\epsilon + \epsilon_c$. More concretely, to incorporate color perturbation into Alg. 1, in line 6 & 7, we compute g_{t1} as (9) and g_{t2} as (11) and update both color and additive perturbation. In line 8, we clip the additive perturbation $\boldsymbol{\delta}$ into $[-\epsilon, \epsilon]$, and the color perturbation $\mathbf{f}_{\mathbf{c}}(\mathbf{x}) - \mathbf{x}$ into $[-\epsilon_c, \epsilon_c]$.

E. Boosting Performance by Watermarking

According to [20], the attack performance could be boosted by superimposing a 30% watermark of the target image on the poisoning images. *However, we do not suggest add 30% watermark in a poisoning attack due to the following two reasons.* First, 30% watermark of the target image can cause very large perturbation locally. Let us consider an extreme case. If the value of a pixel of the target image is 255, then 30% of the pixel value is 77. And if the corresponding pixel value of the original image is 0, then the perturbation on this pixel is $77/255$. Different from watermark, both the additive perturbation and the color perturbation are bounded by relatively small values. Second, 30% watermark of the target image might make the poisoning image look abnormal or artificial. As shown in Fig. 2, the dog image with the watermark (right) looks abnormal after adding 30% watermark of a bird image (middle). Also, we can clearly see the watermark in the right image of Fig. 2. This kind of abnormal or artificial images might be easily detected by the victim and removed from the training dataset. *Therefore, we only apply 10% watermark of the target images to the poisoning images.*

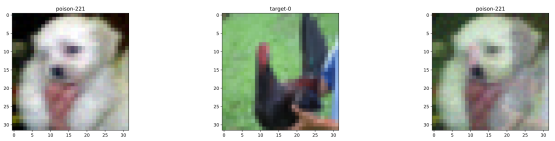

Fig. 2. Watermark trick: original image (left), target image (middle), $0.7\times$original image $+$ $0.3\times$ target image (right)

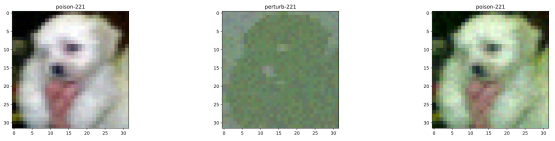

Fig. 3. Meta Poison ($\epsilon_c = 0.04$): original image (left), color perturbation + additive perturbation (middle), poisoning image (right)

V. Experiments

A. Dataset and Network Architectures

We follow [20]–[22] to conduct the experiments mainly on CIFAR-10. We also conduct a case study on a network traffic dataset, which is detailed in Section V-E. CIFAR-10 consists of 50000 training samples and 10000 testing samples. For all the experiments, the budget of poisoning data is less than 10% of the training dataset. By default, we set the poisoning budget as 1%, *i.e.,* 500/50000 poisoning samples. Therefore, the model can still achieve high accuracy on the other testing samples (other than the target samples). We evaluate our method on multiple network architectures. Specifically, we use a 6-layer ConvNet architecture with batch normalization (ConvNet/ConvNetBN), which is a commonly-used lightweight network for CIFAR-10. In addition to ConvNet, we also evaluate our attack on VGG-13 and ResNet-20 [22]. Both are commonly-used model architectures for computer vision tasks.

B. Implementation and Experiment Details

Our implementation is based on pytorch and the code from [20], [22]. By default, we simply set $\alpha = 0.001$ (in Alg. 1), ϵ is set as $8/255$ or $16/255$, and ϵ_c is set as 0.02 instead of 0.04 in [22]. This is because we find that the color perturbation of $\epsilon_c = 0.04$ can dramatically change the basic color of the object, which also makes the image look artificial. In Fig. 3, we display a poisoning image crafted by [22] with color perturbation size of $\epsilon_c = 0.04$. We can see the color of the dog is changed into green. To our knowledge, there are barely any green dogs. Thus, the victim might consider the poisoning dog image artificial and remove it from the training set. Also,

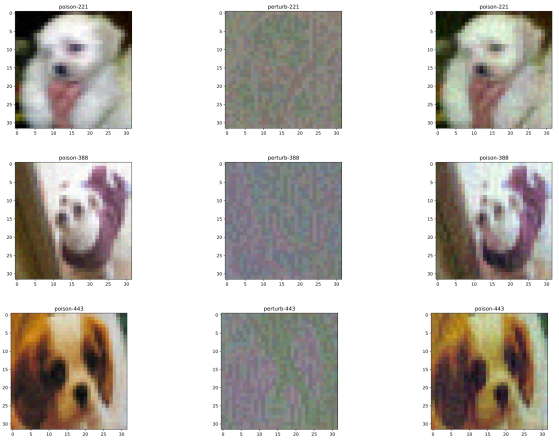

Fig. 4. Visualization of the poisoning data generated by our attack: original image (left), perturbation (middle), poisoning image (right).

as introduced in Section IV-E, we apply 10% watermark to the poisoning data before executing the attack algorithm. We show several poisoning images crafted by our proposed method in Fig. 4. Compared with the poisoning samples in Fig. 2 & 3, the poisoning images crafted by our attack settings look more natural and similar to the original images.

By default, we attack data samples from the bird class and set the dog class as the targeted class following [21], [22]. We craft the poisoning data on 8 surrogate models with the same model architecture (different from [21]). According to Table I, running our proposed attack on 8 surrogate models (*e.g., VGG-13*) usually only cost the computational time to run meta-poison [22] on 4 surrogate models.

Attack success rate is defined as the number of successful attack attempts divided by the total number of attack attempts. Here an attack attempt refers to retraining a model on the training dataset with the poisoning data from scratch. A successful attack attempt means the retrained model is misled to predict the target samples as the attack targeted class (or other than the correct class in the self-concealment scenario introduced in Section V-D).

C. Attack Performance

In this subsection, to evaluate the performance of our proposed attack, *under each setting (introduced in Section III-B) and on each model architecture*, we conduct 3 experiments to craft poisoning data with different target data samples and 10 attack attempts for each experiment.

a) Gray-box Setting: Under the gray-box setting, we assume that the adversary knows the model architecture. In such a case, the optimal strategy for the adversary is to craft the poisoning data on surrogate models with the same model architecture. In Fig. 5, we show the number of successful attempts out of 10 attack attempts on each model architecture for each experiment with a different target sample. As we can see, our attack achieves 97.8% attack success rate (88/90) overall. On ConvNetBN and ResNet-20, our method achieves 96.7% success rate (29/30), and on VGG-16, our

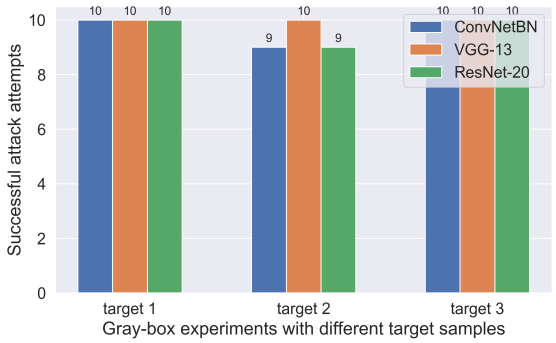

Fig. 5. Gray-box setting: Number of successful attack attempts

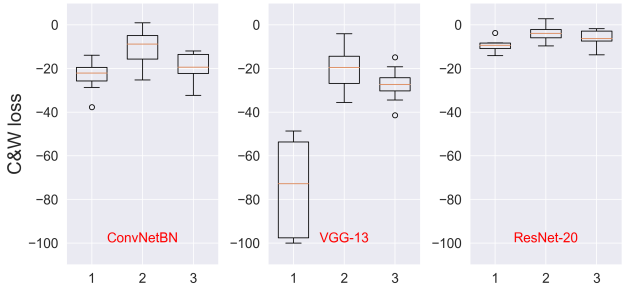

Fig. 6. Box plot of the C&W losses in the attack attempts under the gray-box setting. C&W losses can be interpreted as the negative of the confidence of the victim model to predict the target sample as the attack targeted label.

method achieves 100% success rate. Therefore, compared with ConvNetBN and ResNet-20, VGG-16 seems more vulnerable to our poisoning method *under the gray-box setting*. This observation is further verified by Fig. 6. Fig. 6 shows the statisitics of the final C&W losses of the target sample (1, 2, 3) in each attack attempt. According to the definition in Section II-A (Eq. 1), C&W loss can be interpreted as the negative of the confidence of the model to predict the target sample $\mathbf{x}_t$ as attack targeted label y_t. Negative C&W loss indicates a successful attack attempt, and smaller C&W loss indicates higher confidence, *i.e., better attack performance*. As shown in Fig. 6, the C&W losses of the VGG-13 models on the target sample are smaller than the C&W losses of ConvNetBN or ResNet-20 models, which verifies our observation. Fig. 7 shows that the number of successful attack attempts increases as the number of poisoning data samples increases.

b) Black-box Setting: Under the black-box setting, we assume that the adversary does not know the model architecture. As introduced in Section III-B, under such a setting, the adversary tends to randomly choose one or a few model architectures to build several surrogate models and then craft poisoning data on these surrogate models. To evaluate our attack under the black-box setting, we assume that the victim might use ConvNetBN, VGG-13, or ResNet-20, and the adversary crafts the poisoning data on one of the other two model architectures. In Fig. 8, we show the number of successful attack attempts on each model architecture under the black-box setting. Employing ConvNetBN, VGG-13, and ResNet-20

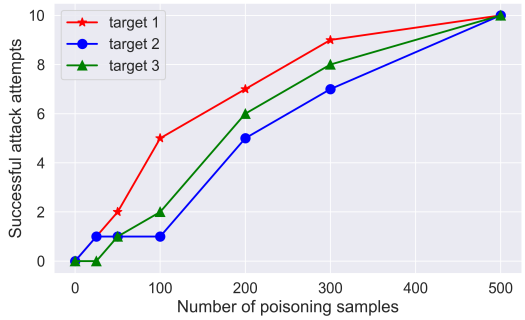

Fig. 7. Number of successful attack attempts (out of 10 attempts) vs number of poisoning samples: The poisoning data is crafted on ConvNetBN, and 10 attack attempts are executed on ConvNetBN for each target sample (1, 2, 3). Note that the total number of training samples is 50000.

Model	# poison data	**Meta Poison [22]**	**Our Method**
ConvNetBN	50/50000	13.5s	9.3s
	500/50000	25.4s	16.0s
VGG-13	50/50000	35.5s	22.3s
	500/50000	71.7s	36.4s
ResNet-20	50/50000	48.6s	30.1s
	500/50000	95.4s	48.1s

TABLE I
AVERAGED COMPUTATIONAL TIME PER CRAFTING STEP ON A SINGLE MODEL ON A SINGLE TITAN GPU.

as the surrogate model architecture, the attack success rates are respectively 80.0% (48/60), 75.0% (45/60), and 88.3% (53/60). Considering that ConvNetBN, VGG-13, and ResNet-20 are very different model architectures, the performance of our proposed attack under the black-box setting is remarkable.

D. Comparison with Previous Work

In this subsection, we mainly compare our poisoning method with the clean-label poisoning methods introduced in Section II-D on efficiency, transferability, and generalizability.

a) Efficiency: Note that the meta-poison method proposed in [22] is currently the most efficient second-order clean-label poisoning method, which uses second-order information implicitly. The other second-order methods that explicitly use Hessian or its inverse are not scalable to large networks like VGG and ResNet [25]. Thus, we mainly compare our proposed method with [22] to show the better efficiency of our proposed attack than the class of second-order poisoning methods. As shown in Table I, on all the three model architectures, our proposed attack is faster than the meta-poison method. Moreover, as the network architecture becomes more complicated, our proposed attack can save more computational time compared with [22]. Specifically, on VGG-13 and ResNet-20, our attack only needs roughly half of the computational time required by the meta-poison method [22] to achieve comparable attack success rate.

b) Transferability: Here transferability refers to the transferablity of the effectiveness of the poisoning data crafted on surrogate models to other (unseen) models. By merging the results in Section V-C, we show the transferability of our proposed poisoning method across different models and model

Model	ConvNetBN	VGG-13	ResNet-20	Overall
ConvNetBN	96.7%	76.7%	93.3%	88.9%
VGG-13	76.7%	100%	83.3%	86.7%
ResNet-20	73.3%	76.7%	96.7%	83.3%

TABLE II
TRANSFERABILITY OF THE EFFECTIVENESS (QUANTIFIED BY ATTACK SUCCESS RATE) OF THE POISONING DATA CRAFTED BY OUR ATTACK ACROSS DIFFERENT MODEL ARCHITECTURES.

architectures in Table II. We claim that our proposed attack indeed has good tranferability since the attack success rate (as a metric to quantify effectiveness) of the poisoning data across different model architectures is over 70%. Previous work such as [21], [22] can also achieve good transferability. However, as introduced in Section II-D, the method proposed in [21] has to craft the poisoning data on an ensemble of multiple model architectures. While our proposed method and [22] only need craft the poisoning data on a single model architecture (but multiple surrogate models) to achieve good transferability.

c) Generalizability: Here generalizability refers to the generalizability of a poisoning method to different scenarios. As introduced in Section II-D, a main drawback of feature collision methods [20], [21] is that the label of the poisoning data has to be the attack targeted label, which limits the generalizability of this method to other scenarios. In contrast to [20], [21], our attack is generalizable to various scenarios. To verify the above claim, we consider a "*self-concealment*" scenario in [22], where we set the targeted class as airplane, and also craft the poisoning data from the airplane class. We select different target samples from the airplane class in the testing dataset, and the attempt of the poisoning attack here is to make the victim model misclassify the target airplane images into another class.

In this experiment, we do not apply any watermark to the poisoning data samples. This is because the poisoning data also comes from the correct class (airplane). Adding the watermark of the target sample to the poisoning data samples motivates the model to recognize the target sample as the ground-truth class (airplane), which contradicts the attack attempt. Thus, instead of adding watermark, we boost the attack performance by increasing the additive perturbation size to from $8/255$ to $16/255$. We craft the poisoning data on ConvNetBN, and attack ConvNetBN, VGG-13, and ResNet-20 with 10 independent attack attempts for each architecture. In Fig. 9, we show the number of successful attack attempts on each model architecture. Notably, our attack achieves 96.7% attack success rate overall in this scenario.

E. Case Study: Poisoning Network Traffic Classification

In the subsection, we verify the generalizability of our poisoning method across various real-world applications and scenarios by a case study on poisoning network traffic classification.

We employ the USTC-TFC2016 dataset [4], [5] for the case study. [4] details how to preprocess the raw traffic data in the dataset into images and train a CNN-based model for classification. Specifically, the continuous raw traffic data is first split to discrete traffic units. Then, the MAC address and

Fig. 8. Black-box setting: Number of successful attempts (out of 10 attack attempts). The legends indicate the model architecture of the surrogate models.

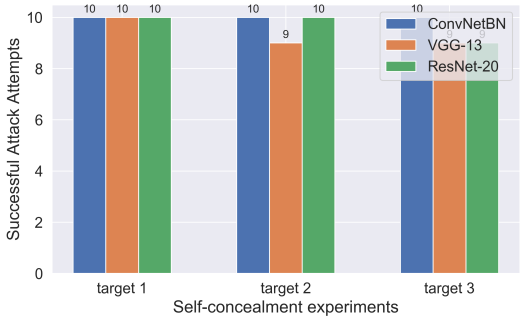

Fig. 9. Self-concealment scenario: The poisoning data is crafted on ConvNetBN and evaluated on ConvNetBN, VGG-13, and ResNet-20 with 10 attack attempts for each architecture. The results are better than the results in Fig. 8. This is because the attack here only needs to mislead the model to classify the target sample into any other classes (other than the correct class).

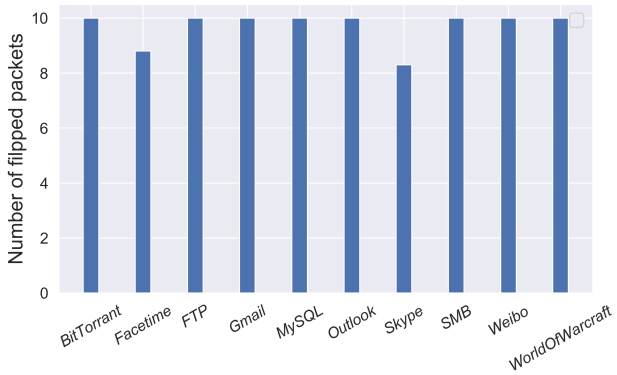

Fig. 10. The averaged number of packets whose predictions are changed by our attack over 10 attack attempts for each class.

IP address in the data link layer are randomized for traffic anonymization/sanitization, and the empty or duplicated files generated by the packets without the application layer or duplicated packets are removed. Finally, all the files are trimmed or padded to 784 bytes, and the resulting files are converted into 28×28 (784) gray images with one pixel representing one byte. A two-layer CNN architecture (similar to LeNet-5 [38]) is used to classify these resulting gray images. In this experiment, We sample 20000 images and 2000 images from 10 classes to form the training and testing dataset. The attempt of our attack is to mislead the model to make wrong predictions on the target data from different classes. For each class, we conduct an experiment by selecting 10 target packets and applying 0.3/1.0 additive perturbation, which is a commonly-used perturbation size for gray images in the previous literature [13], [36], [39], [40], to the training data from the class. We execute 10 attack attempts for each experiment, *i.e.,* retrain 10 models from scratch on the poisoning data plus the remaining clean training data for each class.

In Fig. 10, we show the averaged number of packets whose predictions are flipped by our attack over the 10 attack attempts for each class. Specifically, for the 10 classes, our poisoning method successfully attacks 10, 8.8, 10, 10, 10, 10, 8.3, 10, 10, and 10 packets out of the 10 target packets on average in the 10 attack attempts. Note that "FaceTime" and "Skype" are videotelephony apps, so we conjecture that the relatively poor performance of our poisoning attack on the "FaceTime" and "Skype" class is because the video streaming traffic has some peculiar features compared with the traffic from the other classes so that the poisoned model can still recognize a few target packets from those two classes. We also note that in all the attack attempts, the model can still achieve over 80% (even 90%) accuracy on the other traffic packets in the testing dataset. Therefore, the poisoned models still seem normal but only perform poorly on the target samples.

VI. Conclusion

In this paper, we propose an efficient general-purpose clean-label data poisoning method that only employs first-order information of the surrogate models. The basic idea of our proposed poisoning method is to first identify the first-order model update desired by the adversary, and then perturb the poisoning data to match the update on the poisoning data with the adversary-desired update based on a necessary condition. Theoretically, we prove that our proposed first-order attack is an approximation method for a second-order information driven poisoning method with theoretically-guaranteed performance. Empirically, we show that our attack is more efficient than the state-of-the-art second-order poisoning method, and more generalizable to different scenarios than feature collision based methods. All these theoretical and empirical results demonstrate the outstanding performance and efficiency of our proposed clean-label poisoning method. We expect that our work can inspire the ensuing development of efficient clean-label data poisoning and defense methods.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[4] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International Conference on Information Networking (ICOIN)*. IEEE, 2017, pp. 712–717.

[5] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2017, pp. 43–48.

[6] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[7] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

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

[10] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

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

[12] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.

[14] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[15] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems*, 2018, pp. 8000–8010.

[16] 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.

[17] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[18] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *Advances in neural information processing systems*, 2016, pp. 1885–1893.

[19] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, "When does machine learning fail? generalized transferability for evasion and poisoning attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1299–1316.

[20] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6103–6113.

[21] C. Zhu, W. R. Huang, H. Li, G. Taylor, C. Studer, and T. Goldstein, "Transferable clean-label poisoning attacks on deep neural nets," in *International Conference on Machine Learning*, 2019, pp. 7614–7623.

[22] W. R. Huang, J. Geiping, L. Fowl, G. Taylor, and T. Goldstein, "Metapoison: Practical general-purpose clean-label data poisoning," *arXiv preprint arXiv:2004.00225*, 2020.

[23] F. Suya, S. Mahloujifar, D. Evans, and Y. Tian, "Model-targeted poisoning attacks: Provable convergence and certified bounds," *arXiv preprint arXiv:2006.16469*, 2020.

[24] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 19–35.

[25] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 1885–1894.

[26] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[29] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," *arXiv preprint arXiv:1801.09344*, 2018.

[30] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International Conference on Machine Learning*, 2018, pp. 5283–5292.

[31] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," *arXiv preprint arXiv:1802.03471*, 2018.

[32] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[33] H. Xu, Y. Ma, H. Liu, D. Deb, H. Liu, J. Tang, and A. K. Jain, "Adversarial attacks and defenses in images, graphs and text: A review," *International Journal of Automation and Computing*, vol. 17, no. 2, pp. 151–178, 2020.

[34] K. Ren, T. Zheng, Z. Qin, and X. Liu, "Adversarial attacks and defenses in deep learning," *Engineering*, 2020.

[35] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[36] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *International Conference on Learning Representations*, 2018.

[37] C. Laidlaw and S. Feizi, "Functional adversarial attacks," in *Advances in neural information processing systems*, 2019, pp. 10 408–10 418.

[38] Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard *et al.*, "Learning algorithms for classification: A comparison on handwritten digit recognition," *Neural networks: the statistical mechanics perspective*, vol. 261, p. 276, 1995.

[39] T. Zheng, C. Chen, and K. Ren, "Distributionally adversarial attack," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2253–2260.

[40] D. Zhang, T. Zhang, Y. Lu, Z. Zhu, and B. Dong, "You only propagate once: Accelerating adversarial training via maximal principle," in *Advances in Neural Information Processing Systems*, 2019, pp. 227–238.