Beta Poisoning Attacks Against Machine Learning Models: Extensions, Limitations and Defenses

Atakan Kara*, Nursena Koprucu*, and M. Emre Gursoy
Department of Computer Engineering, Koç University
Istanbul, Turkey

***Abstract*—The rise of machine learning (ML) has made ML models lucrative targets for adversarial attacks. One of these attacks is Beta Poisoning, which is a recently proposed training-time attack based on heuristic poisoning of the training dataset. While Beta Poisoning was shown to be effective against linear ML models, it was originally developed with a fixed Gaussian Kernel Density Estimator (KDE) for likelihood estimation, and its effectiveness against more advanced, non-linear ML models has not been explored. In this paper, we advance the state of the art in Beta Poisoning attacks by making three novel contributions. First, we extend the attack so that it can be executed with arbitrary KDEs and norm functions. We integrate Gaussian, Laplacian, Epanechnikov and Logistic KDEs with three norm functions, and show that the choice of KDE can significantly impact attack effectiveness, especially when attacking linear models. Second, we empirically show that Beta Poisoning attacks are ineffective against non-linear ML models (such as neural networks and multi-layer perceptrons), even with our extensions. Results imply that the effectiveness of the attack decreases as model non-linearity and complexity increase. Finally, our third contribution is the development of a discriminator-based defense against Beta Poisoning attacks. Results show that our defense strategy achieves 99% and 93% accuracy in identifying poisoning samples on MNIST and CIFAR-10 datasets, respectively.**

I. Introduction

The tremendous success of machine learning (ML) in many diverse applications has made ML models lucrative targets for adversarial attacks and manipulation. Poisoning attacks [1], [2] constitute one popular category of attacks – according to a recent survey [3], poisoning is the largest concern for ML deployments in the industry. In a poisoning attack, an attacker tampers with the training dataset of an ML model, e.g., by injecting poisoning samples. Since the model is trained on poisoned data, the model itself also becomes poisoned. The goal of the attacker could be to reduce model accuracy indiscriminately (*untargeted attack*), cause misclassification for specific classes (*targeted attack*), or inject a trigger pattern which can be activated at test-time (*backdoor attack*).

In a poisoning attack, it is typically in the attacker's best interests to cause the maximum negative impact with limited manipulation. The attacker's goal of creating an effective attack is often formulated as a bilevel optimization problem in the literature; however, solving this optimization problem is computationally expensive [1], [4], [5], [6]. Therefore, heuristic and/or approximate attack algorithms have been designed, one of which is the Beta Poisoning attack [7].

* First two authors contributed equally.

Let x_p denote a poisoning sample and y_t denote the targeted class. In a Beta Poisoning attack [7], the goal is to craft x_p which maximizes $P(x_p|y_t)$ where the likelihood $P(x_p|y_t)$ is estimated using kernel density estimation (KDE) methods. Furthermore, x_p is crafted using a linear combination of existing samples, where the weight of each sample x_i is determined by its coefficient β_i. Collectively, the β coefficients are optimized hand-in-hand with KDE using gradient ascent. Overall, Beta Poisoning attacks were shown to be effective against linear ML models [7].

On the other hand, the original version of the Beta Poisoning attack was developed with a fixed Gaussian KDE for likelihood estimation, and its effectiveness against more advanced, non-linear ML models has not been explored. Considering the recent popularity of deep learning, it is worth exploring extensions of Beta Poisoning so that it can be executed with varying KDEs, and its effectiveness against non-linear ML models (such as convolutional neural networks, multi-layer perceptrons, etc.) can be investigated.

To that end, in this paper we aim to advance the state of the art in Beta Poisoning attacks and make three main contributions. First, we propose an extension to the Beta Poisoning attack so that it can be executed with arbitrary kernel and norm functions. We integrate Gaussian, Laplacian, Epanechnikov and Logistic KDEs with ℓ_1, ℓ_2 and ℓ_∞ norms. We empirically show that the choice of KDE indeed has a significant impact on attack effectiveness, especially when attacking linear models. Second, we explore the effectiveness of Beta Poisoning on non-linear ML models. We show that Beta Poisoning is ineffective against non-linear models, even with our extensions. Furthermore, experiment results show that the effectiveness of Beta Poisoning decreases as model non-linearity increases.

Our third contribution is the design and development of a defense against Beta Poisoning attacks. We observe that the linear combination strategy used in Beta Poisoning yields noisy or distorted poisoning samples, which can be distinguished from legitimate samples. Motivated by this observation, our defense utilizes the discriminator of a Generative Adversarial Network (GAN) to identify poisoning samples. We empirically evaluate our defense on popular image recognition datasets: MNIST and CIFAR-10. Results show that our defense achieves 99% and 93% accuracy for the two datasets.

The rest of this paper is organized as follows. We introduce relevant background on notation, linear and non-linear ML

models, and poisoning attacks in Section II. We present the Beta Poisoning attack and our extensions in Section III. Empirical evaluation of the attack and its components are performed in Section IV. Our defense is presented and evaluated in Section V. Section VI presents related work on poisoning attacks and Section VII concludes the paper.

II. Background

A. Notation

Consider a supervised classification task. Let $\mathcal{X} \subseteq \mathbb{R}^d$ denote the feature space, where d is the number of dimensions, and $\mathcal{Y} \in \mathbb{R}$ denote the label space. The training dataset is denoted by $\mathcal{D}_{tr}$ and the validation dataset is denoted by $\mathcal{D}_{val}$. We define $\mathcal{D}_{val}^{y_t} = \{x|(x, y_t) \in \mathcal{D}_{val}\}$ as the subset of $\mathcal{D}_{val}$ which consists of samples belonging to class y_t. A machine learning model $f_\theta : \mathcal{X} \to \mathcal{Y}$ with parameters θ is trained using $\mathcal{D}_{tr}$ and validated using $\mathcal{D}_{val}$. The regularized loss which is optimized during training is represented by $\mathcal{L}(\mathcal{D}_{tr}, \theta)$ and the validation loss is represented by $L(\mathcal{D}_{val}, \theta)$.

B. Linear versus Non-Linear Models

ML models can be linear or non-linear. Linear models directly work on data in the original feature space [8]. In classification tasks, linear models are most accurate when samples with different labels can be separated using hyperplanes in $\mathcal{X}$ [9], [10]. However, in many practical scenarios, training samples from different classes may not be linearly separable. In such cases, it is preferable to use non-linear models which utilize various strategies (such as kernel methods) to map samples to higher-dimensional space and/or construct non-linear decision boundaries with high complexity. Examples of linear models include Linear Regression, Logistic Regression, and Linear Support Vector Machine (Linear SVM). Examples of non-linear models include Multi-Layer Perceptron (MLP), deep learning models such as Convolutional Neural Networks (CNN), and SVMs with non-linear kernels.

Considering the rising popularity of deep learning and the fact that linear separability does not hold on many datasets, non-linear models are commonly used and often yield higher accuracy. We demonstrate this by comparing the accuracy of a Linear SVM with a non-linear CNN on MNIST and CIFAR-10 datasets. We chose Linear SVM for our comparison since it was used in the original Beta Poisoning paper [7], and we chose CNN since it is a popular choice in image recognition. The datasets and models will be explained in more detail in Section IV. As the results in Table I show, the CNN outperforms the linear SVM on both datasets. While their accuracy difference is not too large on the MNIST dataset, the accuracy of the CNN is more than 3x higher than Linear SVM on CIFAR-10. Thus, the accuracy improvement and popularity of non-linear models make them lucrative targets for poisoning attacks, arguably even more so than linear models.

C. Poisoning Attacks

Poisoning attacks are a prominent threat to ML models [1], [2]. In fact, according to a recent survey, among many different security threats, poisoning is the largest concern for ML deployments in the industry [3]. In a data poisoning attack, the attacker injects one or more poisoning samples to $\mathcal{D}_{tr}$. Then, since the model f_θ is trained on the poisoned $\mathcal{D}_{tr}$, the model also becomes poisoned. The attacker can aim to achieve multiple goals by executing a data poisoning attack: In an *untargeted* attack, the attacker may want to maximize $L(\mathcal{D}_{val}, \theta)$ so that the model's accuracy will decrease across all classes. In a *targeted* attack, the attacker may want to decrease accuracy for a specific class. In a *backdoor* attack, the attacker may want to inject a trigger pattern into the model.

TABLE I
Accuracy of a Linear SVM versus a CNN on MNIST and CIFAR-10 datasets.

	Model	
Dataset	Linear SVM	CNN
MNIST	0.91	**0.98**
CIFAR-10	0.25	**0.83**

More formally, let $x_p \in \mathcal{X}$ be a poisoning sample with label $y_p \in \mathcal{Y}$. The attacker wants to maximize the impact of x_p towards achieving his/her goal. Generally, this can be formulated using a bilevel optimization problem [4], [5], [7]:

$$\max_{x_p} \quad L(\mathcal{D}_{val}, \theta^*) \tag{1}$$

$$\text{s.t.} \quad \theta^* \in \arg\min_{\theta} \mathcal{L}(\mathcal{D}_{tr} \cup (x_p, y_p), \theta), \tag{2}$$

$$\mathbf{x}_{lb} \preceq x_p \preceq \mathbf{x}_{ub} \tag{3}$$

Here, Equation 1 states the goal of the attacker. In the above example, the goal is to maximize the loss function over the untainted validation set $\mathcal{D}_{val}$. Since the crafted sample x_p is added to $\mathcal{D}_{tr}$, it affects the training process of the model, which is captured by Equation 2. In other words, the training goal shifts from finding the optimal parameters θ^* which minimize $\mathcal{L}(\mathcal{D}_{tr}, \theta)$ to finding the optimal parameters θ^* which minimize $\mathcal{L}(\mathcal{D}_{tr} \cup (x_p, y_p), \theta)$. Lower and upper bounds ($\mathbf{x}_{lb}$, $\mathbf{x}_{ub}$) are specified in Equation 3 so that x_p will be a legitimate sample for the classification task in hand.

Notice that since θ^* is first trained on poisoned data and then used for calculating the outer validation loss, there is an implicit dependency of the outer validation loss on x_p. Also notice that θ^* has to be retrained for every candidate x_p. As a result, the problem of finding the optimal x_p is inherently a bilevel optimization problem. Unfortunately, solving the bilevel optimization problem is computationally expensive [1], [2], [5], [6], [7]; therefore, effective heuristic solutions are sought. This was a motivating factor for the development of the Beta Poisoning attack.

III. Beta Poisoning Attack

A. Attack Description

Beta Poisoning, first proposed by Cinà *et al.* [7], is a poisoning attack which aims to decrease the accuracy of an ML model by injecting maliciously crafted poisoning samples into $\mathcal{D}_{tr}$. Instead of solving the aforementioned bilevel optimization problem, Beta Poisoning proposes a heuristic

strategy. Its strategy is to poison the target distribution y_t with sample x_p by maximizing the likelihood $P(x_p|y_t)$, making the training dataset no longer linearly separable and thereby hurting the accuracy of linear models. Two desirable properties of Beta Poisoning are that: (i) it does not need access to $\mathcal{D}_{tr}$, and (ii) it does not need to re-train the model while crafting poisoning samples. The mathematical formulation of the Beta Poisoning attack can be stated as:

$$\arg\max_{x_p} \quad P(x_p|y_t) \tag{4}$$

$$\text{s.t.} \quad \mathbf{x}_{lb} \preceq x_p \preceq \mathbf{x}_{ub} \tag{5}$$

A key component of the attack is how $P(x_p|y_t)$ is estimated. While a Gaussian Kernel Density Estimator was used in [7], we propose several alternatives as extensions of the original Beta Poisoning attack, which will be presented and discussed in Section III-B.

The Beta Poisoning attack also enforces that the poisoning samples are obtained as a linear combination of other samples. Let $\mathcal{S} = \{x_1, x_2, ..., x_k\}$ be a set of samples selected randomly from $\mathcal{D}_{val}^{y_t}$, where $k = |\mathcal{S}|$. The set of samples in $\mathcal{S}$ are named *prototypes*. Then, given coefficients $\boldsymbol{\beta} \in \mathbb{R}^k$, the corresponding poisoning sample x_p is obtained as:

$$x_p = \psi(\boldsymbol{\beta}, S) = \sum_{x_i \in \mathcal{S}} \boldsymbol{\beta}_i x_i \tag{6}$$

When this construction of x_p in Equation 6 is inserted into the original optimization problem (Equation 4), it can be observed that the problem becomes equivalent to finding the optimal $\boldsymbol{\beta}$ coefficient values. Once $\boldsymbol{\beta}$ coefficients are optimized, it is easy to construct the resulting poisoning sample as $x_p = \psi(\boldsymbol{\beta}, S)$.

The pseudocode of the Beta Poisoning attack is given in Algorithm 1. The algorithm takes the validation set $\mathcal{D}_{val}$, target class y_t, number of prototypes k, and the lower and upper bounds $\mathbf{x}_{lb}$, $\mathbf{x}_{ub}$ as inputs. Its output is the poisoning sample x_p. On line 1, SAMPLE_PROTOTYPES function is used to construct the set of prototypes, $\mathcal{S}$. This function constructs $\mathcal{S}$ by drawing k samples uniformly at random from $\mathcal{D}_{val}^{y_t}$. On line 2, INITIALIZE_BETA function initializes the $\boldsymbol{\beta}$ coefficients by sampling from a uniform distribution between [0,1]. The main optimization of Beta Poisoning takes place between lines 3-7. On line 4, an initial x_p is generated using the linear combination $\psi(\boldsymbol{\beta}, S)$ and clipped so that its feature values remain between $\mathbf{x}_{lb}$, $\mathbf{x}_{ub}$. On line 5, ESTIMATE_LIKELIHOOD computes an estimate for $P(x_p|y_t)$. As noted earlier, there exist different density estimators to achieve this, which will be explained in Section III-B. On line 6, based on the estimated likelihood p, the $\boldsymbol{\beta}$ coefficients are updated using gradient ascent. Here, α is the learning rate (by default, we use $\alpha = 0.01$). The optimization between lines 3-7 is executed repeatedly until the stop condition is met (line 7). Following [7], the stop condition we use is that the attacker's objective $P(x_p|y_t)$ should not change more than $1e-05$ in two consecutive iterations. Finally, on lines 8-9, the poisoning sample x_p is generated using the optimized $\boldsymbol{\beta}$ coefficients, clipped, and returned.

Algorithm 1 Pseudocode of Beta Poisoning

Input: $\mathcal{D}_{val}$, y_t, k, $\mathbf{x}_{lb}$, $\mathbf{x}_{ub}$
Output: Poisoning sample x_p

1: $\mathcal{S}$ = SAMPLE_PROTOTYPES($\mathcal{D}_{val}$, y_t, k)
2: $\boldsymbol{\beta}$ = INITIALIZE_BETA(k)
3: **repeat**
4: $\quad x_p = \text{CLIP}(\psi(\boldsymbol{\beta}, \mathcal{S}),\ \mathbf{x}_{lb},\ \mathbf{x}_{ub})$
5: $\quad p = \text{ESTIMATE\_LIKELIHOOD}(x_p|y_t)$
6: $\quad \boldsymbol{\beta} = \boldsymbol{\beta} + \alpha \nabla_{\boldsymbol{\beta}} p$
7: **until** stop condition is reached
8: $x_p = \text{CLIP}(\psi(\boldsymbol{\beta}, \mathcal{S}),\ \mathbf{x}_{lb},\ \mathbf{x}_{ub})$
9: **return** x_p

B. Likelihood Estimation

The ESTIMATE_LIKELIHOOD function computes an estimate for $P(x_p|y_t)$, which is a critical component of the Beta Poisoning attack since it is the main factor behind the optimization of the $\boldsymbol{\beta}$ coefficients. While a Gaussian estimator was used in the original version of the attack [7], we observe that the attack can be extended and generalized by enabling the usage of arbitrary estimators. For this purpose, we propose the integration of various *Kernel Density Estimators (KDEs)* to the Beta Poisoning attack. KDEs are non-parametric methods to estimate the probability density function of a random variable using samples from the underlying distribution. They take a free parameter h called the *bandwidth* which determines the smoothness of the fitted distribution. In addition to the Gaussian KDE, we integrated three popular KDEs to the attack (Laplacian, Epanechnikov, Logistic), which are defined below.

Gaussian KDE: Gaussian is a popular KDE choice which works best when the distribution is unimodal, but tends to oversmooth multi-modal distributions [11], [12]. When the Gaussian KDE is used, the Beta Poisoning attack estimates $P(x_p|y_t)$ as:

$$P(x_p|y_t) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \exp\left(-\frac{\|x_p - x\|}{h} \right)^2 \tag{7}$$

where $|\cdot|$ denotes cardinality and $\|\cdot\|$ denotes a norm function.

Laplacian KDE: The Laplacian KDE is used for the construction of coresets to approximate a KDE with a high number of points [13]. It has found applications in many practical tasks, such as integrating a low-voltage weak grid system with a solar photovoltaic system [14], and conductive imaging using MRI [15]. When the Laplacian KDE is used, the Beta Poisoning attack estimates $P(x_p|y_t)$ as:

$$P(x_p|y_t) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \exp\left(-\frac{\|x_p - x\|}{h} \right) \tag{8}$$

Epanechnikov KDE: The Epanechnikov kernel [16] is also a common choice for KDE. When the Epanechnikov KDE is used, the Beta Poisoning attack estimates $P(x_p|y_t)$ as:

$$P(x_p|y_t) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \frac{3}{4} \left(1 - \left(\frac{\|x_p - x\|}{h} \right)^2 \right) \tag{9}$$

Logistic KDE: The Logistic KDE has similarities to the logistic function (sigmoid curve). When the Logistic KDE is used, the Beta Poisoning attack estimates $P(x_p|y_t)$ as:

$$P(x_p|y_t) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \frac{\exp(-\frac{\|x_p - x\|}{h})}{\left(1 + \exp(-\frac{\|x_p - x\|}{h})\right)^2} \tag{10}$$

Note that norm functions are used in all KDEs, denoted by $\|\cdot\|$. We integrated multiple norm functions; namely the ℓ_1, ℓ_2, and ℓ_∞ norms. For a d-dimensional value z, its ℓ_1 norm is defined as:

$$\|z\|_1 = \sum_{i=1}^{d} |z_i| \tag{11}$$

In contrast, ℓ_2 norm is defined as:

$$\|z\|_2 = \sqrt{\sum_{i=1}^{d} z_i^2} \tag{12}$$

Finally, the ℓ_∞ norm is defined as:

$$\|z\|_\infty = \max_{1 \le i \le d} |z_i| \tag{13}$$

IV. Experimental Evaluation

A. Experiment Setup

We experimentally evaluated the effectiveness of the extended Beta Poisoning attack (with varying KDEs and norm functions) on both linear and non-linear ML models. Two popular image recognition datasets were used in the experiments: MNIST and CIFAR-10. MNIST [17] consists of handwritten digits stored as gray-scale 28x28 pixel images. Each pixel takes a value between 0 and 255. There are 10 classes in the dataset, ranging from 0 to 9, each representing one digit. CIFAR-10 [18] consists of colored RGB images of various objects such as airplanes, automobiles, birds, cats and dogs. Each image has 32x32 pixels. Similar to MNIST, there are a total of 10 classes (10 different objects).

We evaluate the attack by training various types of ML models on both datasets. Our linear models include Linear SVM (with regularization parameter C=1 and C=100) and Logistic Regression. Our non-linear models include multi-layer perceptron (MLP), convolutional neural network (CNN), and SVMs with non-linear kernels. Our MLPs have 1, 2 or 3 hidden layers with 256, 128 and 64 units. CNNs consist of two convolution layers, each followed by an activation function and a max pooling layer. Kernels of non-linear SVMs include Laplacian, Polynomial, and RBF kernels.

In order to perform a fair comparison with [7], we focused on the binary classification problem similar to [7], [4], [19], and selected the same pairs of classes with [7] for poisoning. Also, we compare the attack with a baseline attack strategy, random label flipping. Results for this baseline attack is denoted by *Random Flip* in the figures.

Attack effectiveness is measured using *test accuracy*, i.e., the accuracy of the poisoned model on a previously unseen test dataset. Lower the test accuracy, higher the effectiveness of the attack. Each experiment was repeated five times and results were averaged.

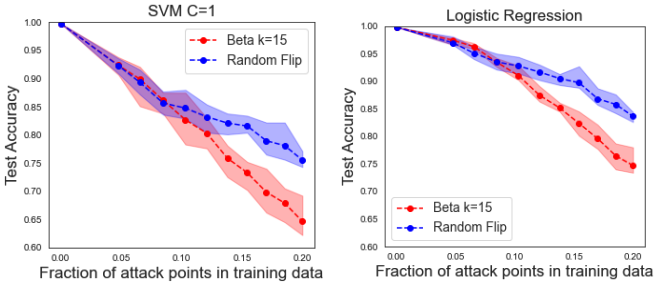

Fig. 1. Attack effectiveness on linear models (Linear SVM and Logistic Regression), MNIST dataset.

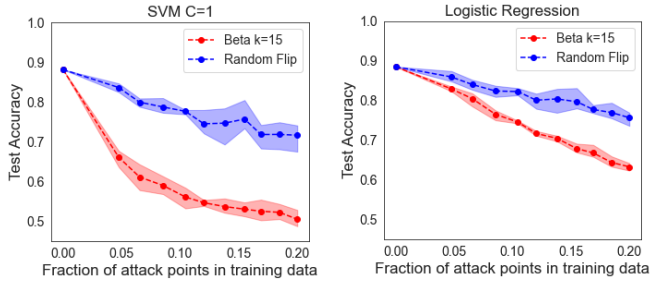

Fig. 2. Attack effectiveness on linear models (Linear SVM and Logistic Regression), CIFAR-10 dataset.

B. Attack Effectiveness on Linear vs Non-Linear Models

We first evaluate the effectiveness of the attack on linear versus non-linear ML models. Results with linear models are reported in Figures 1 and 2 for the MNIST and CIFAR-10 datasets, respectively. The findings in these two figures are consistent with the findings of [7]: As the fraction of attack points (poisoning samples) in $\mathcal{D}_{tr}$ increases, the attack becomes more effective since test accuracy decreases. Also, we observe that the Beta Poisoning attack is more effective than the Random Flip attack on linear models.

In contrast, results with non-linear models are reported in Figures 3 and 4. Interestingly, results show that the Beta Poisoning attack is **not** effective on non-linear models. Despite introducing up to 20% poisoning data into $\mathcal{D}_{tr}$, test accuracy does not drop more than 10% in any one of the MLPs or the CNN. This is very different from the linear models in which up to 30-40% drop in test accuracy was observed. It can also be deduced from Figure 3 that as the non-linearity in the models increase, attack effectiveness decreases. For example, up to 10% drop in test accuracy is achieved on an MLP with 1 hidden layer, but the drop is around 7-8% for an MLP with 2 hidden layers, around 4-5% for an MLP with 3 hidden layers, and less than 2-3% for a CNN. Thus, a highly non-linear model with increased complexity seems to be more resilient against the Beta Poisoning attack. Another interesting observation is that even the Random Flip attack, which is a baseline attack strategy, often performs comparable to or better than Beta Poisoning. This again shows the ineffectiveness of Beta Poisoning on non-linear models.

Considering that the attack is effective on a linear SVM, we also performed experiments with non-linear SVMs for

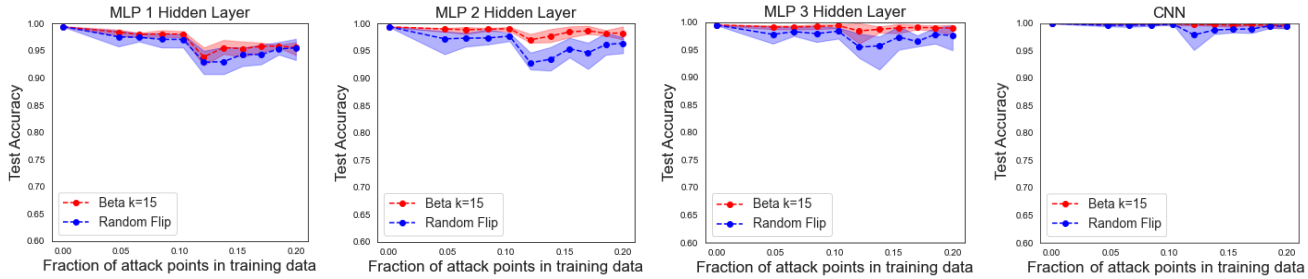

Fig. 3. Attack effectiveness on non-linear models (MLP with 1, 2, 3 layers and CNN), MNIST dataset.

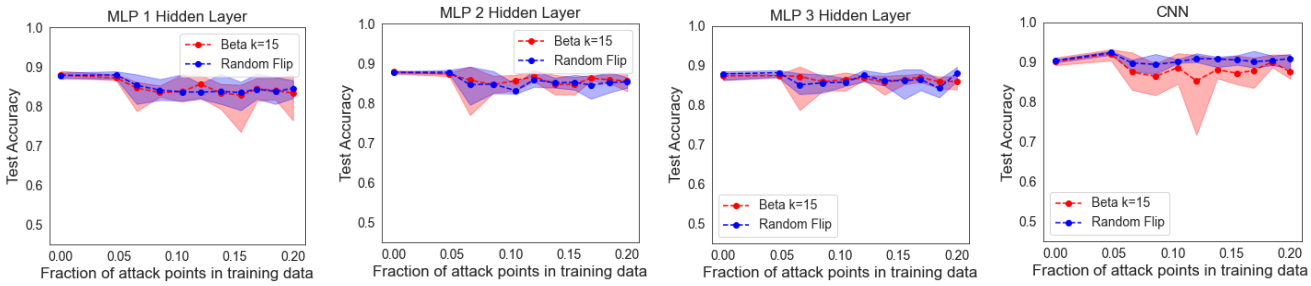

Fig. 4. Attack effectiveness on non-linear models (MLP with 1, 2, 3 layers and CNN), CIFAR-10 dataset.

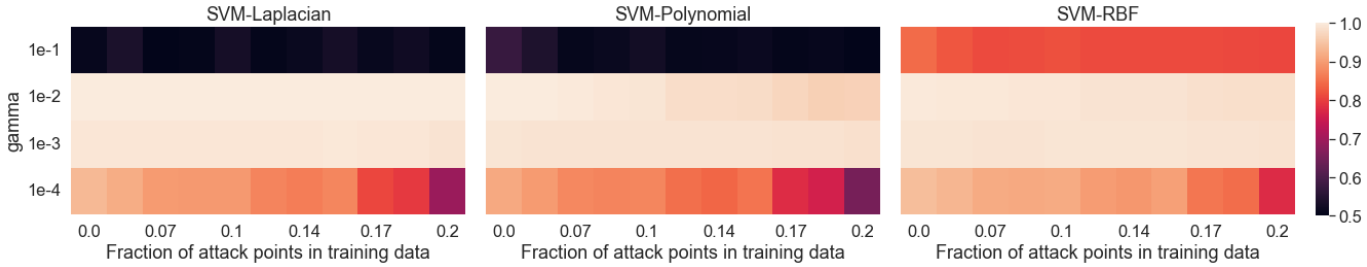

Fig. 5. Accuracy of SVMs with non-linear kernels for various gamma parameters, with regularization $C = 1$.

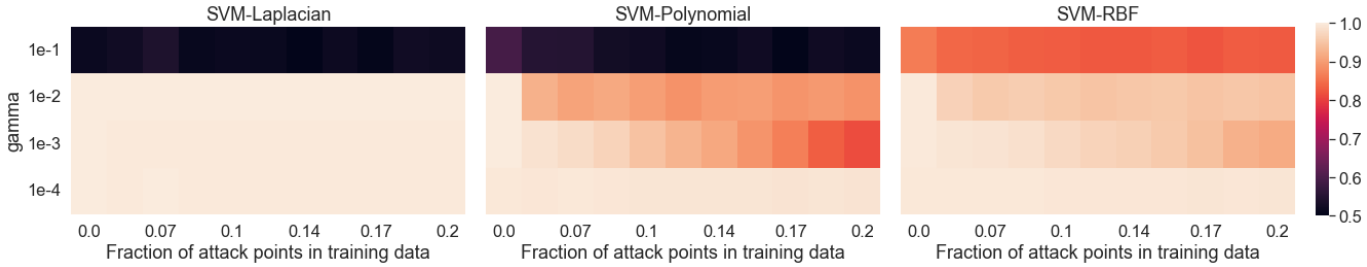

Fig. 6. Accuracy of SVMs with non-linear kernels for various gamma parameters, with regularization $C = 100$.

comparison. In particular, we trained SVMs with 3 commonly used kernels: Laplacian, Polynomial and RBF. In addition, there are two relevant parameters in the construction of non-linear SVMs: regularization parameter C and kernel coefficient gamma. Results with C=1 are reported in Figure 5 and results with C=100 are reported in Figure 6. In each figure, different gamma values including 0.1, 0.01, 0.001, 0.0001 are tested. Darker colors indicate lower test accuracy whereas lighter colors indicate higher test accuracy.

It can be observed from Figures 5 and 6 that there are different scenarios depending on the SVM kernel, C and gamma values. First, there are certain settings in which the models are quite inaccurate to begin with (i.e., without attack). This is true, for example, when gamma = 0.1 and either the Laplacian or Polynomial kernel is used. In such cases, increasing the fraction of attack points in $\mathcal{D}_{tr}$ does not cause any further decrease in test accuracy, due to the initial accuracy being low. Second, in certain settings, the model is quite affected by the increase in the fraction of attack points, e.g., C=1 and gamma = 0.0001, or C=100, gamma = 0.01 or 0.001 and Polynomial kernel is used. In these settings, we observe that the non-poisoned model (fraction of attack points = 0) has fairly high test accuracy, but accuracy keeps decreasing as the fraction of attack points is increased. We can conclude

that in this scenario, the Beta Poisoning attack is effective. Third, there are cases in which the non-poisoned model is highly accurate, and its accuracy does not decrease despite a high fraction of attack points. For example, this happens when the Laplacian kernel is used with C=100 and gamma = 0.01, 0.001 or 0.0001. Since the attack is unable to cause any decrease in test accuracy in these cases, we can conclude that it is ineffective in this scenario.

Overall, based on the experiments conducted in this section, we arrive at the following takeaway messages: While the Beta Poisoning attack is effective on linear models, it is not effective on non-linear models. This is evidenced by the notable difference between the results in Figures 1+2 versus those in Figures 3+4. Furthermore, the effectiveness of the attack typically decreases as the amount of non-linearity increases. For example, the attack is partially effective on a subset of non-linear SVMs and with an MLP with 1 hidden layer. However, as we increase the number of hidden layers in the MLPs or use a CNN (a highly non-linear model), attack effectiveness becomes limited.

C. Impact of Attack Components

We now evaluate the impacts of various attack components on attack effectiveness. In these experiments, we purposefully choose 3 representative models as follows: one model is a linear model on which the Beta Poisoning attack was found to be effective (Linear SVM), one model is a simple non-linear model on which the attack was found to be partially effective (MLP with 1 hidden layer), and one model is a more complex non-linear model on which the attack was found to be ineffective (MLP with 3 hidden layers).

Impact of KDEs: Recall from Section III-B that we proposed an extension to the original Beta Poisoning attack to support varying KDEs and norm functions for likelihood estimation. Results with varying KDEs on MNIST and CIFAR-10 datasets are shown in Figures 7 and 8. It can be observed that the choice of KDE has a substantial impact on attack effectiveness when the model is Linear SVM. If the attacker makes a "bad" choice by choosing Logistic KDE, then the attack only yields 5% reduction in test accuracy. In contrast, remaining KDEs can cause up to 40% reduction in test accuracy. However, when the model is MLP with 1 or 3 hidden layers, since the attack is not as effective as Linear SVM in the first place, the choice of KDE has relatively smaller impact. Interestingly, Logistic KDE becomes a good choice here. In addition to yielding the highest reduction in test accuracy in many cases, the variance of Logistic KDE is also high, meaning that it has a higher risk of causing a strong impact.

Impact of Norm Functions: Recall from Section III-B that we integrated three different norm functions: ℓ_1, ℓ_2, and ℓ_∞ norms. Results with these three norm functions are shown in Figures 9 and 10. When the model is Linear SVM, we observe a trait that is similar to KDEs – ℓ_2 and ℓ_∞ norms have similar effectiveness, whereas ℓ_1 norm is ineffective. Considering the substantial difference between ℓ_1 norm and the other norms, we again see that it is in the best interest of the attacker to choose a good norm function. When the model is MLP with 1 hidden layer, ℓ_2 and ℓ_∞ norms again behave similarly, whereas ℓ_1 norm oftentimes behaves differently. ℓ_1 norm seems to yield higher accuracy reduction in almost half the cases, while ℓ_2 and ℓ_∞ norms yield higher reduction in the other half. Finally, when the model is MLP with 3 hidden layers, ℓ_1 norm seems to be better on MNIST, whereas ℓ_2 and ℓ_∞ norms seem to be better on CIFAR-10. However, note that their difference on MNIST is no larger than 1%, whereas it is often 2-3% in case of CIFAR-10. Thus, we can conclude that while ℓ_1 norm can be better in some cases, generally ℓ_2 and ℓ_∞ norms should be preferred by attackers to maximize attack effectiveness.

Overall, we would like to highlight two takeaway messages based on the experiments we conducted in this section. First, the KDE and norm function chosen by the attacker can have substantial impact on the effectiveness of the Beta Poisoning attack. For example, an attacker should not choose the Logistic KDE and ℓ_1 norm when attacking a Linear SVM. KDEs and norm functions we integrated in Section III-B, together with other KDEs and norm functions that may be integrated in the future, can be used towards improving Beta Poisoning attacks by offering various options towards increasing attack effectiveness in different scenarios. Second, although different KDEs and norm functions show some impact on non-linear models such as MLPs, their impacts are limited. Thus, the Beta Poisoning attack still remains ineffective on non-linear models, despite our extension with new KDEs and norm functions.

V. Defending Against Beta Poisoning Attacks

A. Intuition Behind Our Defense

In this section, we propose an effective defense strategy against Beta Poisoning attacks. Consider a data collector who has a dataset and wants to build an ML model. However, the data collector suspects that his/her dataset may have been contaminated with poisoning samples. The data collector will be able to use our defense strategy to check which samples in his/her dataset are legitimate versus poisonous. The intuition behind our defense stems from our observation that the poisoning samples x_p generated by the Beta Poisoning attack are typically unrealistic in comparison to legitimate samples. The reason is because Algorithm 1 uses the linear combination function ψ (Equation 6) when generating x_p. While the combination of ψ with clipping ensures that x_p will be syntactically valid, its *semantic* properties are not similar to legitimate samples.

In Figure 11, we illustrate our observation using some randomly selected poisoning samples generated by the Beta Poisoning attack. It can be observed that most of the images are indeed not realistic, e.g., the CIFAR-10 images do not seem to contain any real-world objects, and some MNIST digits look like multiple images superimposed over one another. These observations are caused by ψ: since ψ generates poisoning samples by multiplying k real samples with learned $\boldsymbol{\beta}$ coefficients and summing them index-wise, samples generated by the Beta Poisoning attack look like noisy or distorted versions of combinations of legitimate images. There is no mechanism

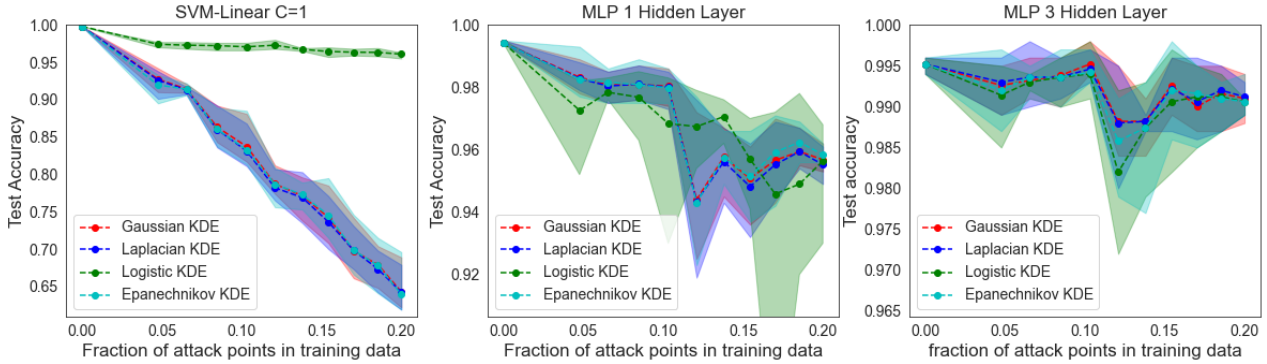

Fig. 7. Accuracy of Beta Poisoning attack with different KDEs on Linear SVM, MLP-1 and MLP-3 (MNIST dataset)

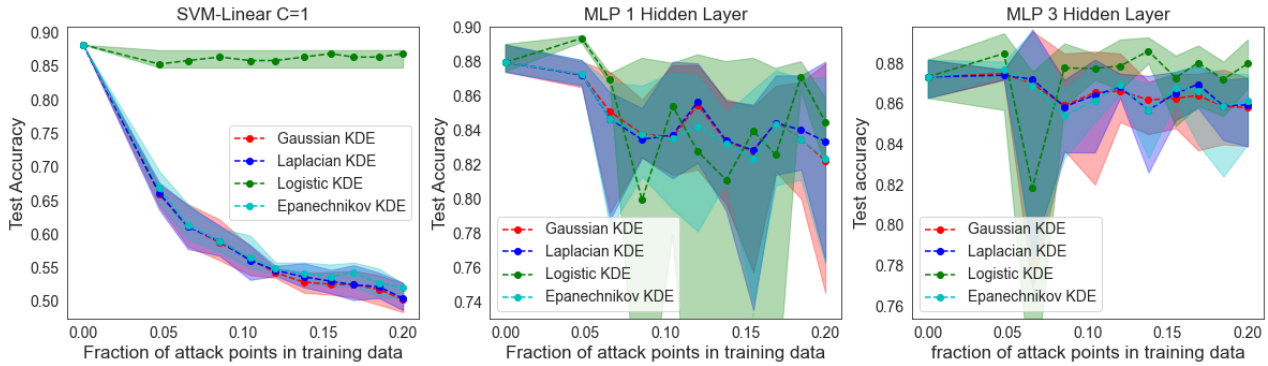

Fig. 8. Accuracy of Beta Poisoning attack with different KDEs on Linear SVM, MLP-1 and MLP-3 (CIFAR-10 dataset)

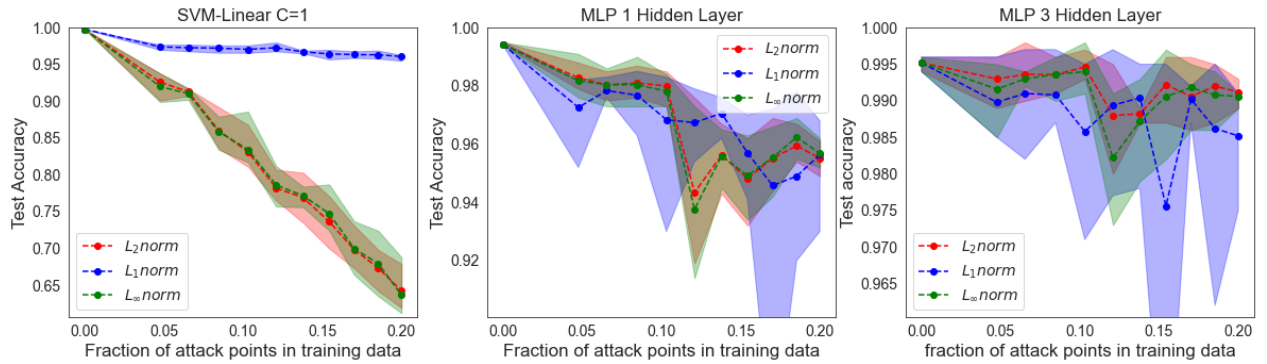

Fig. 9. Accuracy of Beta Poisoning attack with different norm functions on Linear SVM, MLP-1 and MLP-3 (MNIST dataset)

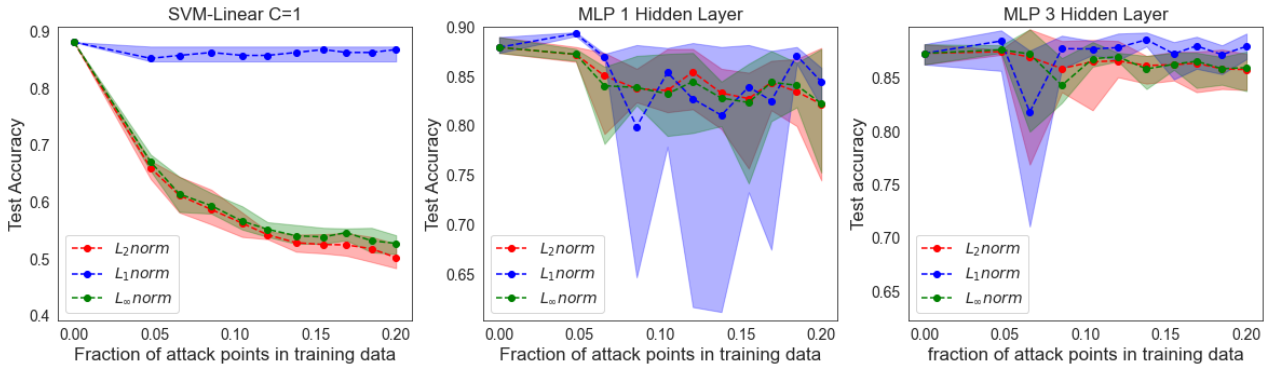

Fig. 10. Accuracy of Beta Poisoning attack with different norm functions on Linear SVM, MLP-1 and MLP-3 (CIFAR-10 dataset)

Fig. 11. Poisoning samples on the MNIST dataset (first row) and CIFAR-10 dataset (second row).

to ensure photorealism or guarantee that nearby pixels in the final sample will be cohesive.

Our defense intuition, therefore, is to separate poisoning samples from legitimate samples based on how realistic they are. In particular, we leverage the discriminator component of Generative Adversarial Networks (GANs) towards this goal.

B. Discriminator-Based Defense Strategy

GANs are unsupervised generative models used for generating realistic samples [20], [21]. They consist of two sub-models: a *generator* which is trained to generate realistic samples and a *discriminator* which aims to distinguish between real versus generated samples. The loss function of the discriminator measures how well the discriminator can distinguish between real versus generated samples. The generator does not have direct access to real samples; instead, it learns from the predictions (or the loss function) of the discriminator. In essence, the core idea behind a GAN is to facilitate a "contest" between the generator and the discriminator – the discriminator gets progressively better at distinguishing between real versus generated samples and the generator gets progressively better at generating more realistic samples that can fool the discriminator.

Our defense against Beta Poisoning attacks is presented in Algorithm 2 and explained verbally as follows. Consider that we have a pre-trained GAN available. This GAN can be obtained in multiple ways, e.g., there are several open-source resources [22], [23], [24]; alternatively, the data collector may have a small, non-poisoned gold standard dataset $\mathcal{D}_{gs}$ which can be used to train a GAN. We extract the discriminator of this GAN, which is capable of distinguishing between real and fake samples accurately. Discriminator Φ, suspicious dataset $\mathcal{D}_{sp}$ (the dataset which is suspected of containing both real and poisoning samples) and the threshold τ become inputs to Algorithm 2. The algorithm returns an array v, such that $v[i] = 1$ means that the i'th sample in $\mathcal{D}_{sp}$ is a poisoning sample, and $v[i] = 0$ otherwise. Between lines 3-9, the defense algorithm feeds each sample in $\mathcal{D}_{sp}$ into Φ and Φ outputs a score for each sample. Higher the score, higher the discriminator's belief that the given sample is a poisoning sample. If the sample's score is higher than threshold τ, our defense labels it as a poisoning sample; otherwise, the sample is labeled as legitimate.

Algorithm 2 Discriminator-Based Defense

Input: Suspicious dataset $\mathcal{D}_{sp}$, discriminator Φ, threshold τ
Output: Array v

1: $size = |\mathcal{D}_{sp}|$
2: $v = \text{zeros}(0, size - 1)$ ▷ zero array with length $size$
3: **for** $i \leftarrow 0$ to $size$ **do**
4: $x_i = \mathcal{D}_{sp}[i]$
5: **if** $\Phi(x_i) \geq \tau$ **then**
6: $v[i] = 1$
7: **else**
8: $v[i] = 0$
9: **end if**
10: **end for**
11: **return** v

C. Experimental Setup for Defense Evaluation

We experimentally evaluated the effectiveness of our proposed defense using the same datasets and similar setup as in Section IV. In order to train GAN models and extract their discriminator, we separated a portion of the original samples from MNIST and CIFAR-10 datasets, and used these samples only for training the GANs. (These samples are not included in training, validation or test sets.) The trained GANs were Deep Convolutional Generative Adversarial Networks (DCGANs). The architectures of the DCGANs were inspired by Tensorflow and PyTorch tutorials. In particular, the MNIST architecture was based on the Tensorflow DCGAN Tutorial[1]. A sigmoid layer was added as the last layer of the discriminator so that the defense parameter τ becomes bounded. The CIFAR-10 architecture was based on the PyTorch tutorial[2].

To construct suspicious datasets $\mathcal{D}_{sp}$, we generated N poisoning samples using the Beta Poisoning attack and combined them with N legitimate samples from $\mathcal{D}_{tr}$. Consequently, we ensured that $\mathcal{D}_{sp}$ consists of an equal number of legitimate and poisoning samples. The type of ML model (e.g., linear on non-linear) was not important for this set of experiments, since it does not impact the poisoning sample generation algorithm of the Beta Poisoning attack.

Consider a sample $x_i \in \mathcal{D}_{sp}$. We define True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) as follows:

- TP: x_i was generated by the Beta Poisoning attack and our defense correctly labels it as a poisoning sample.
- FP: x_i is a legitimate training sample but our defense incorrectly labels it as a poisoning sample.
- TN: x_i is a legitimate training sample and our defense correctly labels it as a legitimate sample.
- FN: x_i was generated by the Beta Poisoning attack but our defense incorrectly labels it as a legitimate sample.

Precision, recall, F1 score and accuracy of the defense are measured according to the above definitions.

[1]https://www.tensorflow.org/tutorials/generative/dcgan
[2]https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial

TABLE II
PRECISION, RECALL, F1 AND ACCURACY OF OUR DEFENSE ON MNIST AND CIFAR-10 DATASETS.

	Precision	Recall	F1	Accuracy
CIFAR-10	0.90	0.97	0.94	0.93
MNIST	0.99	1.0	0.99	0.99

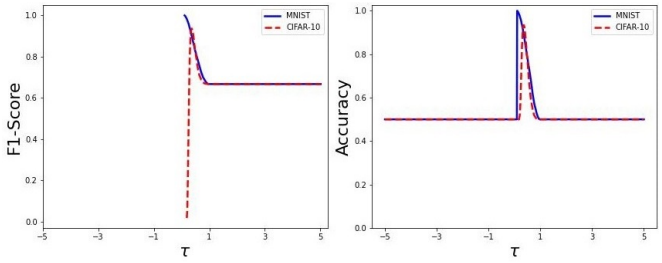

Fig. 12. F1 score and accuracy of our defense under varying values of the threshold parameter τ.

D. Results and Discussion

In Table II, we provide a summary of the precision, recall, F1 score and accuracy values achieved by our defense. We observe that on both datasets, the recall of our defense is strong (0.97 on CIFAR-10 and 1.0 on MNIST), meaning that the defense has a very low number of false negatives, i.e., very few number of poisoning samples remain undetected by the defense. Precision is slightly lower than recall (0.90 on CIFAR-10 and 0.99 on MNIST), which means that the defense can raise some false positives. Overall, with F1 scores of 0.94 and 0.99, and accuracy of 0.93 and 0.99 for the CIFAR-10 and MNIST datasets respectively, we can conclude that our defense is indeed quite successful.

In Figure 12, we explore the impact of varying the τ parameter on the effectiveness of the defense. Note that, since our suspicious datasets contain an equal number of legitimate and poisoning samples, the defense has a base accuracy of 0.5. We observe that the accuracy of the defense is indeed impacted by τ. When τ is too small or too large, defense accuracy converges to 0.5. The reason is because when τ is extremely small (or large), the defense always outputs that x_i is legitimate (or poisoning), thereby predicting exactly half of the suspicious dataset correctly. The predictions of the defense become non-trivial between τ values of 0 and 1. Between these values, we observe that accuracy makes a reverse U-shaped curve for both MNIST and CIFAR-10. We experimentally find that the optimal τ value for MNIST is close to 0.10, whereas the optimal τ value for CIFAR-10 is close to 0.36.

VI. RELATED WORK

Poisoning attacks constitute a prominent category of attacks which can be executed on ML models [1], [2], [3]. One way to perform poisoning attacks is *label flipping*, which does not perturb feature values of training samples, but mislabels a subset of them so that the accuracy of the ML model will decrease [25], [26], [27]. However, label flipping can be suboptimal when ML models are more complex or when the flipped samples are not chosen effectively. Thus, a common strategy in the literature has been to formulate poisoning attacks as bilevel optimization problems.

Biggio et al. [4] proposed one of the first works on bilevel optimization-based poisoning. In this work, a gradient was derived to optimize the poisoning attack, which was then used to iteratively update poisoning samples towards maximizing the target model's validation error. Similar optimization formulations were applied to attack feature selection algorithms in [28] and [29]. Motivated by the hardness of solving the bilevel optimization problem, Mei and Zhu [30] used machine teaching and Krush-Kahn-Tucker (KKT) conditions. Munoz-Gonzalez et al. [5] used back-gradient optimization, aiming to attack a wider class of ML models (such as neural networks) and multi-class classification. MetaPoison [6] used a first-order method to approximate the bilevel problem via meta-learning. Geiping et al. [31] aimed to make attacks less expensive and more visually imperceptible, and proposed a method based on gradient matching.

Instead of solving the bilevel optimization problem, a heuristic strategy called *feature collision* can be used. A key work in this direction was Poison Frogs, proposed by Shafahi et al. [32]. The aim of this work is to create poisoning samples which collide with target test samples in the feature space so that the ML model predicts the test sample according to the poisoned label. While this strategy is effective when the feature extractor is fixed and/or known by the attacker, its effectiveness decreases in more general scenarios [33], [34], [35]. To overcome this problem, [34] and [35] proposed to optimize poisoning samples on ensemble models such that they will become more general and transferable. The concept of transferability is further studied in [19], for both poisoning and evasion attacks.

The computational complexity of solving the bilevel optimization problem has indeed been a recurring challenge [1], [7], which motivated the development of some of the aforementioned heuristic strategies. Among those strategies, most closely related to our work is Beta Poisoning attacks, proposed in [7]. We advance the state-of-the-art in Beta Poisoning attacks in three ways: (i) We extend Beta Poisoning attacks to arbitrary kernel and norm functions. (ii) We show that while Beta Poisoning is effective against linear ML models, it is ineffective against non-linear models. (iii) We propose an effective defense against Beta Poisoning.

Finally, most poisoning attacks in the literature (such as those surveyed above) target classification models in centralized settings. Recently, it was shown that it is also possible to poison regression models [36], [37] and distributed ML settings such as federated learning [38], [39], [40]. Exploring the applicability of Beta Poisoning or other attacks to these scenarios can be investigated in future work.

VII. CONCLUSION

In this paper, we studied Beta Poisoning attacks and made three main contributions. First, we extended the original Beta

Poisoning attack by integrating various KDEs and norm functions (such as Gaussian, Laplacian, Epanechnikov, and Logistic KDE) for likelihood estimation. Second, we showed that although Beta Poisoning attacks are effective against linear ML models, they remain ineffective against non-linear models despite our extensions. Third, we proposed a defense algorithm against Beta Poisoning attacks and showed that it is indeed effective. In future work, we will explore the applications of Beta Poisoning attacks to regression and federated learning.

REFERENCES

[1] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli, "Wild patterns reloaded: A survey of machine learning security against training data poisoning," *arXiv preprint arXiv:2205.01992*, 2022.

[2] M. Goldblum, D. Tsipras, C. Xie, X. Chen, A. Schwarzschild, D. Song, A. Madry, B. Li, and T. Goldstein, "Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[3] R. S. S. Kumar, M. Nyström, J. Lambert, A. Marshall, M. Goertzel, A. Comissoneru, M. Swann, and S. Xia, "Adversarial machine learning-industry perspectives," in *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2020, pp. 69–75.

[4] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proceedings of the 29th International Conference on Machine Learning*, 2012, pp. 1467–1474.

[5] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 27–38.

[6] W. R. Huang, J. Geiping, L. Fowl, G. Taylor, and T. Goldstein, "Metapoison: Practical general-purpose clean-label data poisoning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12 080–12 091, 2020.

[7] A. E. Cinà, S. Vascon, A. Demontis, B. Biggio, F. Roli, and M. Pelillo, "The hammer and the nut: Is bilevel optimization really needed to poison linear classifiers?" in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[8] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, "Recent advances of large-scale linear classification," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.

[9] M. Vidal-Naquet and S. Ullman, "Object recognition with informative features and linear classification," in *ICCV*, vol. 3, 2003, p. 281.

[10] C. J. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[11] Z. I. Botev, J. F. Grotowski, and D. P. Kroese, "Kernel density estimation via diffusion," *The Annals of Statistics*, vol. 38, no. 5, pp. 2916–2957, 2010.

[12] M. Zhou, X. Li, Y. Wang, S. Li, Y. Ding, and W. Nie, "6g multisource-information-fusion-based indoor positioning via gaussian kernel density estimation," *IEEE Internet of Things Journal*, vol. 8, no. 20, pp. 15 117–15 125, 2021.

[13] J. M. Phillips and W. M. Tai, *Improved Coresets for Kernel Density Estimates*, pp. 2718–2727. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611975031.173

[14] N. Kumar, B. Singh, B. K. Panigrahi, C. Chakraborty, H. M. Suryawanshi, and V. Verma, "Integration of solar pv with low-voltage weak grid system: Using normalized laplacian kernel adaptive kalman filter and learning based inc algorithm," *IEEE Transactions on Power Electronics*, vol. 34, no. 11, pp. 10 746–10 758, 2019.

[15] J. Shin, J.-H. Kim, and D.-H. Kim, "Redesign of the laplacian kernel for improvements in conductivity imaging using mri," *Magnetic Resonance in Medicine*, vol. 81, no. 3, pp. 2167–2175, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/mrm.27528

[16] V. A. Epanechnikov, "Non-parametric estimation of a multivariate probability density," *Theory of Probability & Its Applications*, vol. 14, no. 1, pp. 153–158, 1969.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

[19] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *28th USENIX security symposium (USENIX security 19)*, 2019, pp. 321–338.

[20] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[22] M. Zoo, "Generative models: Deep learning models and code," 2022, https://modelzoo.co/category/generative-models.

[23] Github, "Pytorch-gan: Pytorch implementations of generative adversarial networks," 2022, https://github.com/eriklindernoren/PyTorch-GAN.

[24] ——, "The-gan-zoo: A list of all named gans," 2022, https://github.com/hindupuravinash/the-gan-zoo.

[25] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise," in *Asian Conference on Machine Learning*. PMLR, 2011, pp. 97–112.

[26] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli, "Support vector machines under adversarial label contamination," *Neurocomputing*, vol. 160, pp. 53–62, 2015.

[27] H. Xiao, H. Xiao, and C. Eckert, "Adversarial label flips attack on support vector machines," in *ECAI 2012*. IOS Press, 2012, pp. 870–875.

[28] C. Frederickson, M. Moore, G. Dawson, and R. Polikar, "Attack strength vs. detectability dilemma in adversarial machine learning," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[29] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?" in *International Conference on Machine Learning*. PMLR, 2015, pp. 1689–1698.

[30] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[31] J. Geiping, L. H. Fowl, W. R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein, "Witches' brew: Industrial scale data poisoning via gradient matching," in *International Conference on Learning Representations*, 2020.

[32] 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," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[33] 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.

[34] H. Aghakhani, D. Meng, Y.-X. Wang, C. Kruegel, and G. Vigna, "Bullseye polytope: A scalable clean-label poisoning attack with improved transferability," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2021, pp. 159–178.

[35] 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*. PMLR, 2019, pp. 7614–7623.

[36] 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.

[37] J. Wen, B. Z. H. Zhao, M. Xue, A. Oprea, and H. Qian, "With great dispersion comes greater resilience: Efficient poisoning attacks and defenses for linear regression models," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3709–3723, 2021.

[38] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *International Conference on Machine Learning*. PMLR, 2019, pp. 634–643.

[39] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data poisoning attacks against federated learning systems," in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 480–501.

[40] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to byzantine-robust federated learning," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1605–1622.