Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/MBE

MBE, 20(10): 17726–17746.
DOI: 10.3934/mbe.2023788
Received: 28 June 2023
Revised: 05 August 2023
Accepted: 13 August 2023
Published: 15 September 2023

Research article

Online data poisoning attack against edge AI paradigm for IoT-enabled smart city

Yanxu Zhu[1,3,4], Hong Wen[1,3,4,*], Jinsong Wu[2,5] and Runhui Zhao[1,3,4]

[1] School of Aeronautics and Astronautics, University of Electronic Science and Technology of China, Chengdu 611731, China
[2] School of Artificial Intelligence, Guilin University of Electronic Technology, Guilin 510004, China
[3] Aircraft Swarm Intelligent Sensing and Cooperative Control Key Laboratory of Sichuan Province, Chengdu 611731, China
[4] Intelligent IoT Communication Technology Engineering Research Center, Chengdu 611731, China
[5] Department of Electrical Engineering, University of Chile, Santiago 8370451, Chile

* **Correspondence:** Email: sunlike@uestc.edu.cn; Tel: +8613882228239.

Abstract: The deep integration of edge computing and Artificial Intelligence (AI) in IoT (Internet of Things)-enabled smart cities has given rise to new edge AI paradigms that are more vulnerable to attacks such as data and model poisoning and evasion of attacks. This work proposes an online poisoning attack framework based on the edge AI environment of IoT-enabled smart cities, which takes into account the limited storage space and proposes a rehearsal-based buffer mechanism to manipulate the model by incrementally polluting the sample data stream that arrives at the appropriately sized cache. A maximum-gradient-based sample selection strategy is presented, which converts the operation of traversing historical sample gradients into an online iterative computation method to overcome the problem of periodic overwriting of the sample data cache after training. Additionally, a maximum-loss-based sample pollution strategy is proposed to solve the problem of each poisoning sample being updated only once in basic online attacks, transforming the bi-level optimization problem from offline mode to online mode. Finally, the proposed online gray-box poisoning attack algorithms are implemented and evaluated on edge devices of IoT-enabled smart cities using an online data stream simulated with offline open-grid datasets. The results show that the proposed method outperforms the existing baseline methods in both attack effectiveness and overhead.

Keywords: data poisoning attack; online learning; edge Artificial Intelligence

1. Introduction

Artificial Intelligence (AI) services have been widely adopted in various fields of smart city such as industrial manufacturing, enterprise services and daily consumption. These services, including unmanned driving, e-commerce, smart homes, and smart finance, have profoundly transformed people's lifestyles and enhanced production efficiency [1,2]. Edge computing has become popular due to its advantages of ultra-low latency, energy efficiency, and strong scalability, which allows it to share the computing resources and service pressure of the cloud center and optimize the computing architecture of AI services. This in turn creates favorable conditions for pushing the AI frontier to the IoT (Internet of Things)-enabled edge, which resides at the last mile of the Internet [3]. The continuous convergence of edge computing and artificial intelligence has led to the emergence of a new paradigm called edge intelligence (edge AI paradigm) [4–6]. The edge AI paradigm enables end entities in the networks to make decisions based on local data instead of sending it to the remote cloud [7]. The deployment of AI models on edge nodes enables AI training and inference and provides AI services to terminal devices. However, the edge AI paradigm is more vulnerable to attacks due to less potent security protocols on the resource-constrained edge hardware [8]. In the edge AI environment, attackers can easily masquerade as legitimate user terminals to generate malicious data online and attack the edge AI model. Therefore, it is imperative to evaluate potential attacks that can target AI models at the edge, especially in the context of smart cities.

Among the potential attacks, the most destructive attack is data poisoning attacks (DPA). Current offline DPA are not suitable for the online learning process used in the edge AI paradigm, where most learning tasks involve predicting continuous data rather than classification, unlike the image processing or classification scenarios that data poisoning attacks primarily focus on. Although some studies have investigated online DPA based on resource-rich environments, these methods are not applicable in resource-constrained IoT-enabled smart cities environments, where the problem of periodic overwriting of training samples cannot be handled. Moreover, existing online attack methods use randomly selected sample points for attacks, which are not ideal for expensive bi-level optimization attack strategies. Therefore, existing research on DPA is not suitable and there is a need to optimize existing online attacks to adapt to resource-constrained environments, while enhancing the efficiency of attacks under online mode. Therefore, the main contributions of this work are as follows:

• It proposes an online poisoning attack framework based on the edge AI environment of IoT-enabled smart city for the first time. The framework takes into account the limited storage space in the AI edge environment and proposes a rehearsal-based buffer mechanism to manipulate the model by incrementally polluting the sample data stream that arrives at the appropriately sized cache to optimize the efficiency of the attack.

• It proposes a maximum-gradient-based sample selection strategy that overcomes the problem of periodic overwriting of the sample data cache after training. This strategy converts the operation of traversing historical sample gradients into an online iterative computation method.

• It proposes a maximum-loss-based sample pollution strategy that solves the problem of each poisoning sample being updated only once in the gradient ascent direction in basic online DPA. This strategy transforms the bi-level optimization problem from the offline mode to the online mode.

• It implements online gray-box poisoning attack algorithms with the framework and strategies mentioned above. It evaluates the effectiveness and overhead of the proposed attack on edge devices of IoT-enabled smart city using an online data stream simulated with offline open-grid datasets.

The rest of this paper is organized as follows. Section 2 presents the related works on data poisoning attack. Section 3 describes the basic settings, symbol notations and related issues of the five elements relevant to offline and online DPA. Section 4 presents an online incremental poisoning attack framework in the edge AI environment of IoT-enabled smart cities and provides a detailed description of the proposed sample selection and pollution strategies. Section 5 introduces online algorithms for gray-box poisoning attack with maximum-gradient-based sample selection strategy and maximum-loss-based sample pollution strategy. Section 6 presents the experiment and result analysis. Finally, Section 7 concludes this paper.

2. Related works

Offline DPA has received extensive attention in the research community, mainly focusing on interfering with the training process of offline or batch learning algorithms. In this setting, attackers repeatedly poison randomly selected samples in the direction of maximum gradient and construct a poisoned sample set that maximizes the loss. At the end of attacks, the constructed poisoned sample set is inserted into the end of the legitimate sample set one-time. Since the pioneering work of the Biggio team [9], DPA has undergone significant development, and a large amount of research has been carried out based on their work. Among them, the Mei team [10] formalized the poisoning problem as a bi-level optimization problem. To improve efficiency, some teams have proposed label flipping [11] and statistically-based [12] poisoning methods, which do not require model fitting and reduce the algorithmic complexity. Although these two methods have a low algorithmic complexity, they are easily detected and discarded by human examiners or automated detectors. For most machine learning or artificial intelligence models [13], the gradient ascent method is the most computationally expensive method, but it is the most effective and confidential [14].

Online DPA has drawn increasing attention in recent years. In the setting of online DPA, attackers contaminate the arriving samples in a specific order to achieve the attack objective of accumulating loss. There are four main challenges brought to offline DPA in online environment. First, due to the inability to obtain the entire sample set, the baseline clean dataset for constructing poison samples can only be built from the current cache or historical sample set. Second, the order in which samples arrive is also a factor to be considered in poisoning attacks. Third, offline DPA can poison any position in the sample set, while in online mode, only the current cache samples can be poisoned. Fourth, high-cost attack methods in offline mode may become inappropriate. To solve the problem of unknown sample sets, Burkard and Lagesse [15] proposed heuristic attacks against support vector machines (SVM) learning from data streams. This method is more like fake online attacks (with full knowledge of future samples, referred to as the clairvoyant online DPA [16]), which obviously does not conform to the premise assumption of online mode. Zhang et al. [16] and Margiotta et al. [17] used the Markov decision process method to model the online DPA problem, which is also based on the premise of knowing the probability distribution of the samples. Although they also propose to build an increasingly accurate empirical distribution from historical sample data, it cannot solve the problem of high cost of model predictive control and sample distribution bias. Some papers [18–20] have studied the calculation and optimization methods of sample influence, but unfortunately, these methods are based on the hat matrix or Hessian matrix constructed from the entire sample set in offline mode and are not applicable to online mode. The work closest to ours is Wang and Chaudhuri [21], applied gradient-based offline methods to online DPA and proposed a sample selection method based on maximum recursive gradient. Moreover, in edge AI environments where historical samples are periodically overwritten, the absence of some historical samples makes it impossible to compute the recursive gradient. Therefore, our approach differs from theirs in that we adopt a rehearsal buffer-based method for calculating recursive gradient incrementally, which addresses the issue of periodic overwriting of the sample data cache after training in edge AI environment.

3. Preliminary

AI models typically contain five elements [22]: feature space, learning type (e.g., regression or SVM), learning algorithm, learning hyperparameters and training datasets. Based on attackers' degree of knowledge over these five elements and the type of elements involved, DPA can be classified into different types. This section describes the basic settings, symbol notation and related issues of the five elements relevant to offline and online DPA. The definitions of the symbols used in this paper are shown in Table 1.

Table 1. Notation description.

Symbol	Description
$\mathbb{R}^{N\times K}$	Feature space of dataset
D_{trn},D_{tst}	Training dataset and testing dataset
$X^{train}_{n_{trn}\times k}, y^{train}$	Feature matrix and label vector of training dataset
$X^{test}_{n_{tst}\times k}, y^{test}$	Feature matrix and label vector of testing dataset
$h(\boldsymbol{X})$	Learning model
$\mathcal{J}(D_{trn}, \boldsymbol{\theta})$	Objective function with learning parameter $\boldsymbol{\theta}$
$\boldsymbol{\mathcal{L}}(D_{trn}, \boldsymbol{\theta})$	Loss function with learning parameter $\boldsymbol{\theta}$
$\boldsymbol{\Omega}(\boldsymbol{\theta})$,$\boldsymbol{\lambda}$	The regularization term, regularization factor
$\boldsymbol{\theta}_{i-1}$,$\boldsymbol{\theta}_i$	Model parameter before and after one iteration of learning
$\nabla_{\boldsymbol{\theta}_i}\mathcal{J}(D_s, \boldsymbol{\theta}_i)$	Gradient of the objective function with respect to the model parameter
D_p	Poisoned sample set
$\boldsymbol{\theta}^*$,θ_p^*,$\boldsymbol{\theta}_t$	Parameter under normal training, parameter after attack, parameter of time slice t
α,ε	Learning rate, convergence condition
Π	Projection operator
b	Cache size
$D_{cache_1:cache_T}$	The samples that have been trained in the past time slices
γ	The poisoning rate
$n_p^{(t)}$	The number of poisoned samples in time slice
$\boldsymbol{x}_{max\_gradient}$	Feature vector of the sample with the highest gradient
p	The size of rehearsal buffer

3.1. Basic setting and notation

For the feature space $\mathbb{R}^{N\times K}$, the total number of feature vectors and the dimension of each feature vector are represented as $N(N\sim\infty)$ and k, respectively. Given a training sample set and a test sample set, denoted as $D_{trn}=\{X^{train}_{n_{trn}\times k}, y^{train}\}$ and $D_{tst}=\{X^{test}_{n_{tst}\times k}, y^{test}\}$, $X^{train}_{n_{trn}\times k}, X^{test}_{n_{tst}\times k}\in\mathbb{R}^{N\times K}, n_{trn}, n_{tst}<N$. $X^{train}_{n_{trn}\times k}$ and $X^{test}_{n_{tst}\times k}$ represent the training feature matrix and the test feature matrix consisting of n_{trn} and n_{tst} feature vectors from the feature space, y^{train} and y^{test} represent the corresponding label vectors. Given the learning model $y=h(\boldsymbol{X})$ and objective function $\mathcal{J}(D_{trn},\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the learning parameter of the model, the normal training goal is to calculate the optimal parameter $\boldsymbol{\theta}^*$ for the minimum objective function shown in Eq (1). Equation (2) gives the expression of the objective function, where $\boldsymbol{\mathcal{L}}(D_{trn},\boldsymbol{\theta})$ represents the loss function, $\boldsymbol{\Omega}(\boldsymbol{\theta})$ and $\boldsymbol{\lambda}$ represent the regularization term and their corresponding regularization factor. Equation (3) gives the iterative process for solving the objective function using a learning algorithm (taking gradient descent algorithm as an example), where α represents the learning rate of the iteration, $\boldsymbol{\theta}_{i-1}$ and $\boldsymbol{\theta}_i$ represents the model parameter before and after one iteration of learning and $\nabla_{\boldsymbol{\theta}_i}\mathcal{J}(D_s,\boldsymbol{\theta}_i)$ represents the gradient of the objective function with respect to the model parameter. The learning algorithm terminates and obtains the optimal parameter when meeting the convergence condition ε in Eq (4), which is usually set to 1×10^{-8}.

$$\boldsymbol{\theta}^*=argmin\mathcal{J}(D_{trn},\boldsymbol{\theta}) \tag{1}$$

$$\mathcal{J}(D_{trn},\boldsymbol{\theta})=\boldsymbol{\mathcal{L}}(D_{trn},\boldsymbol{\theta})+\boldsymbol{\lambda\Omega}(\boldsymbol{\theta}) \tag{2}$$

$$\boldsymbol{\theta}_i=\boldsymbol{\theta}_{i-1}-\alpha\nabla_{\boldsymbol{\theta}_{i-1}}\mathcal{J}(D_{bactch_{i-1}},\boldsymbol{\theta}_{i-1}) \tag{3}$$

$$|\mathcal{J}(D_{trn},\boldsymbol{\theta}_i)-\mathcal{J}(D_{trn},\boldsymbol{\theta}_{i-1})|<\varepsilon \tag{4}$$

Learning algorithms can be divided into two types: offline and online. The former mainly includes algorithms such as stochastic gradient descent (SGD), mini-batch gradient descent (MBGD) and batch gradient descent (BGD). The latter primarily consists of algorithms like online gradient descent (OGD) and online mini-batch gradient descent (OMBGD). The biggest difference between offline and online learning algorithms is the way in which the training sample set is obtained and used [23]. The sample set used by offline learning algorithms is known and fixed (can be trained repeatedly), while the sample set used by online learning algorithms is gradually obtained over time (each sample is only trained once) and future samples are unknown. Therefore, DPA is also divided into offline DPA and online DPA based on the different learning algorithms.

3.2. Offline DPA

For the above AI models, the basic offline DPA attack can be formalized as a bi-level optimization problem as shown in Eq (5), where D_{tst} represents the clean test sample set, D_p represents the poisoned sample set, θ_p^* represents the poisoned parameter learned by the model and D_{trn} is the baseline clean dataset used for constructing D_p. According to the definition of Eq (1), we can know that the inner optimization $\theta_p^*\in argmin_\theta\mathcal{J}(D_{trn}\cup,\theta)$ in Eq (5) represents the usual minimization of the model loss during the fitting of a model on both the clean training dataset D_{trn} and the poisoned dataset D_p and that the outer optimization $argmax_{D_{tst}}\mathcal{J}(D_{tst},\theta_p^*)$ represents the maximization of the prediction loss under the influence of the poisoned parameter θ_p^*. Equation (6) uses gradient ascent to contaminate data points p_i, making their sample values $D_{p_i}^{(t)}$ contaminated as $D_{p_i}^{(t+1)}$, where $\nabla_{D_{p_i}^{(t)}}\mathcal{J}\left(D_{tst},\theta_{p_i}^{(t)}\right)$ represents the gradient of the objective function at the data point p_i, α represents the learning rate of iteration and t is the number of iterations. Π represents the projection operator, which projects the contaminated sample values into the feasible domain of the feature space.

$$argmax_{D_{tst}}\mathcal{J}(D_{tst},\boldsymbol{\theta}_p^*)\ s.t.\boldsymbol{\theta}_p^* \in argmin_{\boldsymbol{\theta}}\mathcal{J}(D_{trn} \cup D_p,\boldsymbol{\theta}) \tag{5}$$

$$D_{p_i}^{(t)} = \Pi(D_{p_i}^{(t-1)} + \alpha\nabla_{D_{p_i}^{(t-1)}}\mathcal{J}\ (D_{tst},\boldsymbol{\theta}_{p_i}^{(t-1)})) \tag{6}$$

Figure 1 shows the schematic diagram of offline DPA. In the figure, rectangles represent training samples, where green rectangles represent normal samples and red rectangles represent poisoned samples. Rounded squares represent model parameters, with red rounded squares representing poisoned parameters after the poisoning attack is completed. Diamonds represent decision conditions. Black solid arrows indicate the normal training process, which is demonstrated using the MBGD algorithm as an example in the figure, where the batch size is b ($b = 1$ for SGD algorithm and $b = n$ for BGD algorithm). The algorithm fits the model and computes parameters once using Eq (3) for each batch of samples until convergence is reached. Red dashed arrows represent the inner optimization loop mentioned in Eq (5), i.e., obtaining new convergent parameters through gradient descent after the poisoned sample is added to the training set. Red solid arrows represent the outer optimization loop mentioned in Eq (5), i.e., updating and maximizing the loss on the training sample set using Eq (6), obtaining the optimal poisoned parameter $\boldsymbol{\theta}_p^*$ finally. To maintain the generalization of the model, the sample set is randomly reordered after each traversal and D_p is also selected from the training set randomly, which demonstrates that offline DPA does not consider any order of samples.

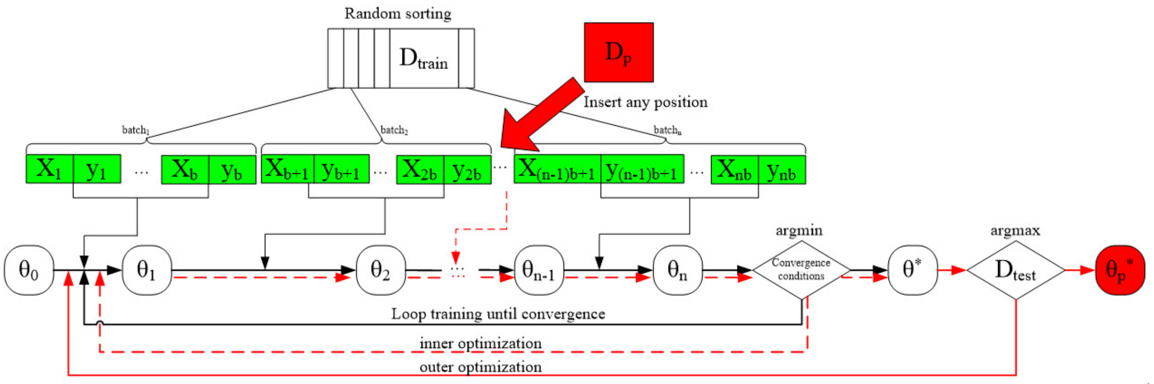


Figure 1. Schematic diagram of offline DPA.

3.3. Basic Online DPA

Figure 2 illustrates the schematic diagram of basic online DPA. In the figure, rectangles still represent samples, with red ones indicating poisoned samples and green ones representing normal samples. Rounded squares denote model parameters and red rounded squares indicate the poisoned parameters after the poisoning attack is completed. White hollow arrow represents training order of sample data stream. The black arrows demonstrate the normal training process using OMBGD as an example. Here, $cache_1 \sim cache_T$ represent the samples that have been trained in the past time slices (each time slice contains$b$samples, defined as$D_{cache_t} = \{(\boldsymbol{x}_i, y_i)\}_{i=(t-1)b+1}^{tb}$ and since each sample can only be used once, the objective function is defined as $regret$ in Eq (7)). The online training goal is to minimize $regret$ as in Eq (8). From the definition of the formula, $regret$ reflects the gap between the cumulative loss $\sum_{t=1}^{T}\mathcal{J}(D_{cache_t},\boldsymbol{\theta}_t)$ and the minimized loss

$\min_{D_{cache_1:cache_T} \in \mathbb{R}^{N\times K}} \sum_{t=1}^{T} \mathcal{J}(D_{cache_1:cache_T}, \boldsymbol{\theta}_t)$ up to time slice T. The minimized loss is equivalent to the optimal loss obtained by offline training using all historical samples up to time slice T. In the normal training process, the model parameters are updated using the samples in each time slice according to Eq (9). When the convergence condition in Eq (10) is reached, the optimal parameter $\boldsymbol{\theta}^*$ are obtained. By substituting Eq (7) into Eq (10), it can be concluded that the convergence condition is equivalent to judging whether the loss of samples on the current time slice is sufficiently small. Due to the inability to use training samples for fitting the model iteratively as in the offline mode, it is necessary to determine whether the learning algorithm has reached the convergence condition after each parameter update.

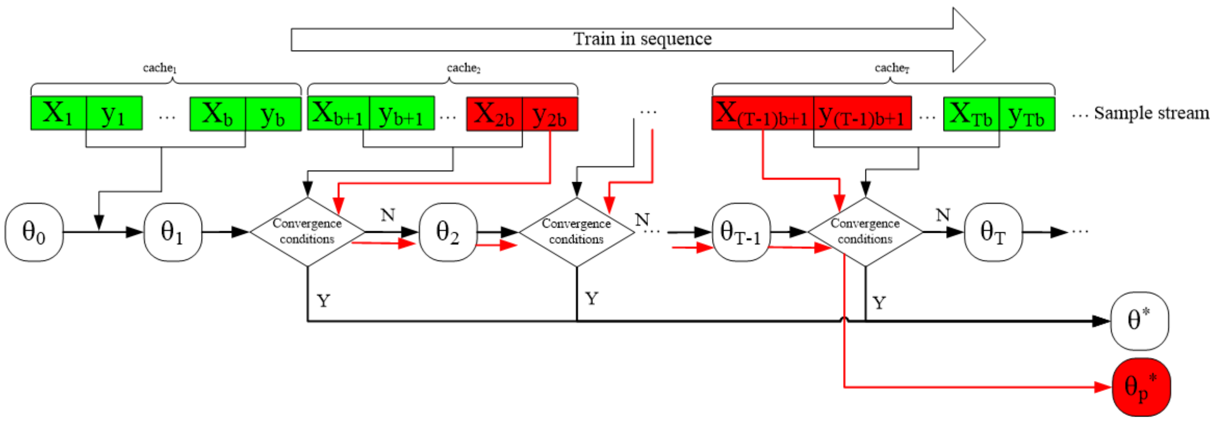


Figure 2. Schematic diagram of basic online DPA.

The red arrows show the process of online DPA. In this process, the attacker first selects an attack sample point in the current time slice, such as choosing the sample point $D_{p_{2b}} = \{\boldsymbol{x}_{2b}, y_{2b}\}$ from time slice $cache_2$ to start the attack. According to Eq (11), the sample is poisoned, $D_{p_i}^{(t-1)}$ and $D_{p_i}^{(t)}$ represent the samples before and after poisoning. Then, the poisoned sample point is trained together with other normal sample points. In the same way, points are selected, polluted and the model is trained in the subsequent time slices $cache_3$ to $cache_T$. Finally, at time slice T, the convergence condition is reached and the optimal poisoned parameter $\boldsymbol{\theta}_p^*$ is obtained.

$$regret\ (T) = \sum_{t=1}^{T} \mathcal{J}(D_{cache_t}, \boldsymbol{\theta}_t) - \min_{cache_1:cache_T \in \mathbb{R}^{N\times K}} \sum_{t=1}^{T} \mathcal{J}(D_{cache_1:cache_T}, \boldsymbol{\theta}_t) \tag{7}$$

$$\boldsymbol{\theta}^* = argmin(regret) \tag{8}$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \nabla_{\boldsymbol{\theta}_{t-1}} \mathrm{regret}(t-1) \tag{9}$$

$$|regret\ (t) - regret\ (t-1)| = |\mathcal{J}(D_{cache_t}, \boldsymbol{\theta}_t)| < \varepsilon \tag{10}$$

$$D_{p_i}^{(t)} = \Pi(D_{p_i}^{(t-1)} + \alpha \nabla_{D_{p_i}^{(t-1)}} \mathcal{J}(D_{cache_{t-1}}, \boldsymbol{\theta}_{t-1})) \tag{11}$$

4. Attack model

We define our attack model following the framework proposed in [24], which involves identifying attacker's goals and describing their knowledge and capabilities. This information is then utilized to define attack strategies. To simplify the problem description and express the proposed method clearly, from this section onward, we assume that the target model for the online DPA attack is linear regression model $y = h(\boldsymbol{X}) = \boldsymbol{\theta}^T \boldsymbol{X}$, with the regularization term set to zero and the batch size as b. This means that the learning algorithm employs the OMBGD algorithm and the objective function is defined as the mean squared error loss function in Eq (12).

$$\mathcal{J}(D_{cache_t}, \boldsymbol{\theta}_t) = \frac{1}{b} \cdot \sum_{i=(t-1)b+1}^{tb} (\boldsymbol{\theta}_t \cdot \boldsymbol{x}_t - y_t)^2 \, , t = 1, \ldots, T \tag{12}$$

4.1. Attacker's goal

Consistent with offline DPA and basic online DPA objectives, the goal is to poison specific samples to cause mis-predictions selectively, while the availability attack aims to indiscriminately corrupt learning models by poisoning training samples. Unlike the basic online DPA goal in section 4.1, we attempt to maximize the attack effect for each poisoned sample point.

4.2. Attacker's knowledge

Based on understanding of the five elements mentioned in section 3, attacks can be divided into white-box, black-box and gray-box types [22]. Since online DPA is unknown for future sample streams, attackers can only be aware of some training samples and cannot possess the knowledge of a white-box attack. In reality, a completely black-box attack is also infeasible, as attackers need to understand partial training samples at least. Therefore, we consider a gray-box attack method for online DPA, where the attackers are assumed to have knowledge of the learning type (e.g., regression), learning algorithm and partial training samples but do not know the trained parameters. Another difference from basic online DPA is that it assumes that the sample data stream will be permanently stored on the AI service's device after training is completed. However, in the edge AI environment, data streams will be periodically overwritten, meaning that attackers can only be aware of the samples in the time slice stored in the buffer and they are unaware of both future samples and some historical samples.

4.3. Attacker's capability

The attacker's capability is limited to manipulating the training sample data; that is, altering the training process is not allowed. In basic online DPA, attackers have full control over samples in current or historical time slices. However, in this paper, the defined attacking capability is limited to having full control over the samples stored in the buffer of current time slice. The attack is constrained within a certain range, that is, the poisoning rate up to the current time slice T cannot exceed a certain limit γ, as it would expose the attack. We define the poisoning rate as $\gamma = (\sum_{t=1}^{T} n_p^{(t)})/Tb$, where $n_p^{(t)}$ is the number of poisoned samples in time slice t. The attacker can choose to attack same number of samples in each time slice or vary the number of poisoned samples. In this paper, we assume that the number of poisoned samples in each time slice is same, making it easy to prove that $\gamma = n_p^{(i)}/b$.

4.4. Attack strategy

Online DPA can be divided into four stages [24]: sample monitoring, attack point selection, data polluting and stream poisoning. The primary focus of the core strategy setting is on the attack point selection and data polluting stages. These two stages are used to construct the poisoned sample set. Then, the training samples containing the poisoned sample set are replayed to the target model in the final stage. After the attack point selection is complete, pollution of the sample points can involve polluting the feature vectors of the training sample stream, polluting the labels or polluting both simultaneously. Extensive literature has demonstrated that polluting the feature vectors yields the most optimal results. Therefore, the pollution strategy in this section will also focus on polluting the feature vectors. This section will emphasize the description of attack point selection and pollution strategies.

4.4.1. Maximum-gradient-based sample selection strategy

In online DPA, modifying training points at certain positions in the stream may yield high benefits. This strategy could be potentially exploited by a successful attack to reduce the search space. Equation (13) presents a gradient-based selection strategy, where at time slice t, the target function calculates the gradient for all samples prior to t and the sample with the highest gradient is chosen as the poisoning sample. The rationale behind this strategy is that the gradient is an indicative measure of the target function's variation. A higher gradient at a node implies that the target function changes rapidly at that point. By polluting the sample point in the direction of the gradient ascent, the target function will increase rapidly, thus achieving the desired attack effect.

$$\boldsymbol{x}_{max\_gradient} = argmax_{\boldsymbol{x}_i \in \boldsymbol{x}_1 : \boldsymbol{x}_{tb}} \left\| \frac{\partial \mathcal{J}(D_{cache_1:cache_t}, \boldsymbol{\theta}_t)}{\partial \boldsymbol{x}_i} \right\|_2 \quad (13)$$

To compute the gradient of the target function with respect to the samples, we use the recursive gradient given by Eq (14).

$$\frac{\partial \mathcal{J}(D_{cache_1:cache_t}, \boldsymbol{\theta}_t)}{\partial \boldsymbol{x}_i} = \begin{Bmatrix} 0 & if\ \boldsymbol{x}_i \in cache_t \\ \frac{\partial F(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t} \cdot \frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{x}_i} & if\ \boldsymbol{x}_i \in cache_{t-1} \\ \frac{\partial F(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t} \cdot \frac{\partial \boldsymbol{\theta}_t}{\partial \boldsymbol{\theta}_{t-1}} \cdot \ldots \cdot \frac{\partial \boldsymbol{\theta}_{i+2}}{\partial \boldsymbol{\theta}_{i+1}} \cdot \frac{\partial \boldsymbol{\theta}_{i+1}}{\partial \boldsymbol{x}_i} & if\ \boldsymbol{x}_i < cache_1 \sim cache_{t-2} \end{Bmatrix} \quad (14)$$

By applying the chain rule and substituting Eq (12) into (14) and to simplify the expression, we set $\mathrm{b} = 1$ in Eq (12), which leads to Eq (15).

$$\frac{\partial \mathcal{J}(D_{cache_1:cache_t}, \boldsymbol{\theta}_t)}{\partial \boldsymbol{x}_i} = \begin{Bmatrix} 0 & if\ i \geq t \\ (\boldsymbol{\theta}_t \boldsymbol{x}_t - y_t) \cdot \boldsymbol{x}_t \cdot \alpha \cdot (y_{t-1} - 2\boldsymbol{\theta}_{t-1} \boldsymbol{x}_{t-1}) & if\ i = t-1 \\ (\boldsymbol{\theta}_t \boldsymbol{x}_t - y_t) \cdot \boldsymbol{x}_t \cdot (\boldsymbol{I} - \alpha \boldsymbol{x}_{t-1} \boldsymbol{x}_{t-1}{}^T) \cdot (\boldsymbol{I} - \alpha \boldsymbol{x}_{t-2} \boldsymbol{x}_{t-2}{}^T) \cdot \cdots \cdot (\boldsymbol{I} - \alpha \boldsymbol{x}_{i+1} \boldsymbol{x}_{i+1}{}^T) \cdot \alpha \cdot (y_i - 2\boldsymbol{\theta}_i^T \cdot \boldsymbol{x}_i) & if\ i < t-1 \end{Bmatrix} \quad (15)$$

From Eq (15), it is clear that to compute the gradient of the current target function with respect to each sample point at time slice t, one needs to know the feature vectors of all historical sample points. However, in edge AI environments, due to storage limitation, the sample data cache is periodically overwritten after training. This leads to a situation we define as cache strategy with forgetting. We define a cache strategy similar to that of the sliding window. Figure 3 illustrates the strategy of storing the online training sample stream arriving at the edge node, the solid and dashed boxes represent the samples cached at time slice t and $t + 1$, respectively. The capacity of the cache is b. According to this strategy, at time t, the sample $D_{cache_t} = \{(\boldsymbol{x}_i, y_i)\}_{i=(t-1)b+1}^{tb}$. The new samples $D_{cache_{t+1}}$ will completely overwrite the historical samples D_{cache_t} at time slice $t + 1$.

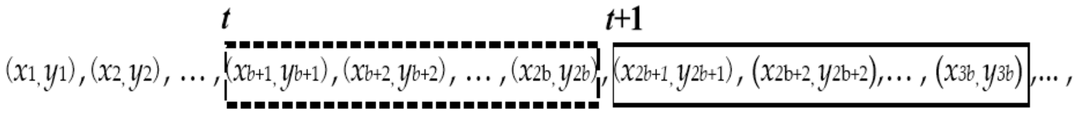

t
t+1
$(x_1, y_1), (x_2, y_2), \ldots, (x_{b+1}, y_{b+1}), (x_{b+2}, y_{b+2}), \ldots, (x_{2b}, y_{2b}), (x_{2b+1}, y_{2b+1}), (x_{2b+2}, y_{2b+2}), \ldots, (x_{3b}, y_{3b}), \ldots,$

Figure 3. Cache strategy with forgetting.

Based on this storage strategy, the forgotten samples cannot be used to calculate the gradient of the target function with respect to that point using Eq (15), rendering the selection strategy inapplicable. To address this issue, we propose a gradient calculation algorithm based on rehearsal. The main principle of this algorithm is that after each maximum gradient sample point is selected at current time slice, both the point and its corresponding model parameters are stored in a dedicated buffer called the rehearsal buffer. The rehearsal buffer can remember partial historical information when samples are overwritten, and its size is less than or equal to the total number of poisoning sample points. When the next time slice arrives, the sample points in the rehearsal buffer are combined with those in the newly arrived time slice to calculate the new maximum gradient point, as shown in Eq (16). To simplify the expression, we still set the cache size $\mathrm{b} = 1$. Meanwhile, we assume the size of rehearsal buffer is p, in which the samples are denoted as $(x_{RB_i}, y_{RB_i}) \in \{(x_{RB_1}, y_{RB_1}), (x_{RB_2}, y_{RB_2}), .., (x_{RB_p}, y_{RB_p})\}$.

$$\frac{\partial \mathcal{J}(D_{cache_t} \cup D_{rehearsal\,buffer}, \boldsymbol{\theta}_t)}{\partial x_i} = \left\{\begin{matrix} 0 & if\ i \geq t \\ (\boldsymbol{\theta}_t \boldsymbol{x}_t - y_t) \cdot \boldsymbol{x}_t \cdot \alpha(y_{t-1} - 2\boldsymbol{\theta}_{t-1}\boldsymbol{x}_{t-1}) & if\ i = t-1 \\ (\boldsymbol{\theta}_t \boldsymbol{x}_t - y_t)\boldsymbol{x}_t(\boldsymbol{I} - \alpha \boldsymbol{x}_{t-1}\boldsymbol{x}_{t-1}{}^T) \cdot (\boldsymbol{I} - \alpha \boldsymbol{x}_{RB_p}\boldsymbol{x}_{RB_p}{}^T) \cdot \cdots \cdot \alpha \cdot (x_{RB_i} - 2\boldsymbol{\theta}_i^T \cdot \boldsymbol{x}_{RB_i}) & if\ i < t-1 \end{matrix}\right\} \quad (16)$$

4.4.2. Maximum-loss-based sample pollution strategy

Section 3.3 presents the basic online DPA, in which each poisoning sample is only updated once in the gradient ascent direction and cannot be updated iteratively in the same direction as in offline mode to maximize attack effectiveness. This may result in a prolonged attack process. Online DPA is highly time-sensitive, so that if the attack is not completed within a limited time, it can be easily detected by defense algorithms. Therefore, this paper still attempts to update each poisoning point in the gradient ascent direction to maximize the attack effect. Equations (17) and (18) provide a bi-level optimization formula for the online mode, which is called the rehearsal-based poisoning strategy. In the inner optimization of Eq (18), the model is trained using the samples from the current time slice and the poisoned samples from the rehearsal buffer to obtain the inner layer parameter $\boldsymbol{\theta}_p^*$. In the outer optimization of Eq (17), the model generates the maximum prediction loss under the influence of poisoned parameter $\boldsymbol{\theta}_p^*$ using the samples of newly arrived time slice.

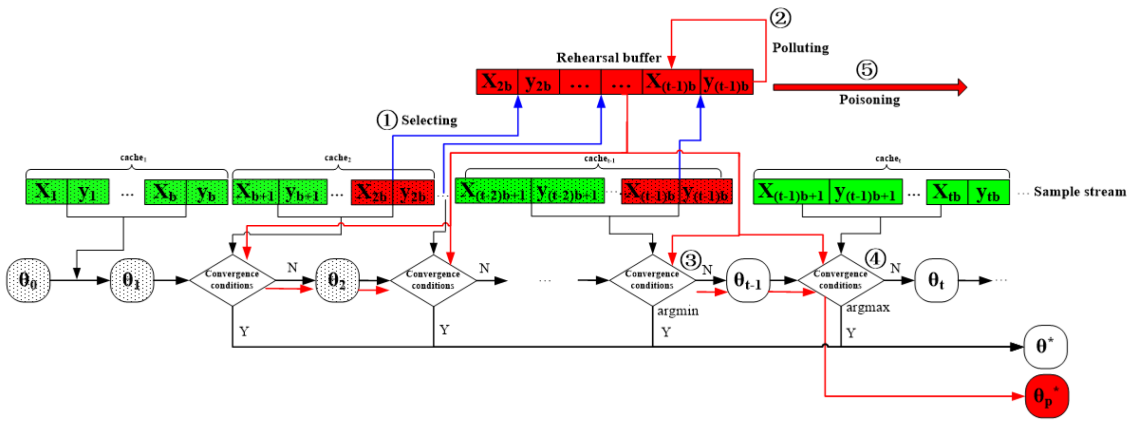
$$argmax_{D_p\ in\ \text{rehearsal buffer}} \mathcal{J}(D_{cache_t}, \boldsymbol{\theta}_p^*), \tag{17}$$

$$s.t.\ \boldsymbol{\theta}_p^* \in argmin_{\boldsymbol{\theta}} \mathcal{J}(D_{cache_{t-1}} \cup D_p\ in\ \text{rehearsal buffer}, \boldsymbol{\theta}) \tag{18}$$

5. Attack algorithm

5.1. Implementation

Figure 4 presents the schematic diagram of the rehearsal-based online DPA method. The rectangular, rounded square, diamond, black arrow and red arrow graphic elements are set up consistently with the definitions in Figure 2. The figure shows the time slices $cache_1 \sim cache_t$ arriving sequentially at the edge node. Once the samples in the time slice are trained, they will be overwritten by the subsequent time slices. The overwritten time slices are marked with dotted shading. The blue arrows in the figure indicate the selection and storage of the points with the largest gradients. The circled numbers in the figure represent the attack steps. Assuming the attack starts from time slice 2, in step 1, the point with the largest gradients is selected and stored from the combination samples of rehearsal buffer and current time slice based on the current model parameter. In step 2, the samples recorded in the rehearsal buffer are poisoned in the gradient ascent direction. In step 3, the poisoned samples are trained together with the other samples from the current time slice to obtain new model parameter and it is determined whether the convergence condition of Eq (18) is reached. In step 4, the samples of newly arrived time slice and the new model parameter are used together to determine whether the convergence condition of Eq (17) is reached. In the following, each time a new time slice arrives, the above operations are repeated until both convergence conditions of Eqs (17) and (18) are reached. Finally, in step 5, the poisoned samples from the rehearsal buffer are inserted one by one into each time slice of the target training data stream. This section describes the rehearsal based online poisoning attack algorithm step by step.



②
Polluting
Rehearsal buffer
⑤
Poisoning
X_{2b} y_{2b} … … $X_{(t-1)b}$ $y_{(t-1)b}$
① Selecting
$cache_1$
$cache_2$
$cache_{t-1}$
$cache_t$
X_1 y_1 … X_b y_b
X_{b+1} y_{b+1} … X_{2b} y_{2b}
$X_{(t-2)b+1}$ $y_{(t-2)b+1}$ … $X_{(t-1)b}$ $y_{(t-1)b}$
$X_{(t-1)b+1}$ $y_{(t-1)b+1}$ … X_{tb} y_{tb}
Sample stream
θ_0
θ_1
Convergence conditions
N
Y
θ_2
③
④
argmin
argmax
θ_{t-1}
θ_t
θ^*
θ_p^*

Figure 4. Schematic diagram of rehearsal based online DPA.

Algorithm 1 presents the overall implementation process of the rehearsal-based online DPA algorithm. Lines 1 to 6 initialize the attack. Starting from line 7, the main loop of the attack process has begun. Constant max_{iter} is used to prevent oscillation caused by inappropriate parameter settings. Lines 8 and 9 record the number of current time slice and the total number of samples. Lines 10–12 perform the point selection operation, with the details described in Algorithm 2. After the point selection operation, lines 13–14 pollute each poisoned sample point recorded in the rehearsal buffer one by one, i.e., updating each poisoned sample point in the direction of the maximum gradient. The projection operator Π in line 14 projects the polluted sample values into the feasible domain of the feature space. Line 14 adjusts the pollution speed by controlling the learning rate α and decay factor β. Line 15 completes the inner optimization process corresponding to Eq (18). Lines 16–18 complete the convergence condition judgment process, wherein the attack ends when the current model parameter achieve the minimum loss in time slice $D_{cache_{t-1}}$ and the maximum loss in time slice D_{cache_t}

simultaneously. Lines 19–24 update the maximum and minimum loss values of the model during the attack process.

Algorithm 1 Rehearsal based Online DPA

```
Input: training data stream, $D_{trn-s} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{T}, T \to \infty$, objective function, $\mathcal{J}$, termination condition, $\varepsilon$, poisoning rate, $\gamma$, learning rate, $\alpha$, line search decay ,$\beta$ and maximum number of iterations, $max_{iter}$
1:$t \leftarrow 1$(Initialization of time $t$)
2:Initialization of $\boldsymbol{\theta}_t$
3:$count \leftarrow 0$(Initialization of samples number)
4:$D_{reh\_buf} \leftarrow \emptyset$(Initialization rehearsal buffer)
5:Initialization of MSE_max
6:Initialization of MSE_min
7:while $t < max_{iter}$ do
8:    $t \leftarrow t + 1$
9:    $count \leftarrow count + len(cache_{t-1})$
10:   if $len(D_{reh\_buf}) <= count \cdot \gamma$ :
11:       $D_{reh\_buf} \leftarrow SelectSample\_MaxGradient(D_{cache_{t-1}}, D_{reh\_buf}, \boldsymbol{\theta}_{t-1}, count \cdot \gamma)$
12:   end if
13:   for range $\boldsymbol{x}_i$ in $D_{reh\_buf}$
14:       $\boldsymbol{x}_i = \Pi(\boldsymbol{x}_i + \alpha\beta\nabla_{x_i}\mathcal{J}(D_{cache_{t-1}} \cup D_{reh\_buf}, \boldsymbol{\theta}_{t-1}))$
15:   $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha\nabla_{\boldsymbol{\theta}_{t-1}}\mathcal{J}\ (D_{cache_{t-1}} \cup D_{reh\_buf}, \boldsymbol{\theta}_{t-1})$
16:   if $|\mathcal{J}\ (D_{cache_t}, \boldsymbol{\theta}_t) -$ MSE_max$| < \varepsilon$ and $|\mathcal{J}\ (D_{cache_{t-1}}, \boldsymbol{\theta}_t) -$ MSE_min$| < \varepsilon$:
17:       break
18:   end if
19:   if $\mathcal{J}\ (D_{cache_t}, \boldsymbol{\theta}_t) >$ MSE_max :
20:       MSE_max $\leftarrow \mathcal{J}\ (D_{cache_t}, \boldsymbol{\theta}_t)$
21:   end if
22:   if $\mathcal{J}\ (D_{cache_{t-1}}, \boldsymbol{\theta}_t) <$ MSE_min :
23:       MSE_min $\leftarrow \mathcal{J}\ (D_{cache_{t-1}}, \boldsymbol{\theta}_t)$
24:   end if
25:end while
26:Insert each poison sample from the rehearsal buffer into each time slice of the target data stream one by one
```

Algorithm 2 presents the implementation process of the point selection operation. Line 1 combines the sample points of the current time slice and the sample points in the rehearsal buffer. Line 2 calculates the gradient for each sample in the merged sample set D_{curr} according to Eq (16) and finds the sample point with the maximum gradient. Lines 3–11 complete the addition and replacement operations for the samples in the rehearsal buffer. Line 3 first determines whether the newly generated sample point $x_{max\_gradient}$ is already included in the rehearsal buffer. If it is not, the following steps are performed. Line 4 checks if the current size of the rehearsal buffer has reached the upper limit of the number of poisoned samples. If it has reached and the current sample is not included in the rehearsal buffer, lines 5–6 replace the sample point with the smallest gradient in the rehearsal buffer. If the current size of the rehearsal buffer has not yet reached the upper limit of the number of poisoned samples and the current sample is not included in the rehearsal buffer, line 9 directly inserts the sample point

$x_{max\_gradient}$ into the rehearsal buffer.

Algorithm 2 SelectSample_MaxGradient

Input: Samples in the current time slice, $D_{cache_{t-1}}$, Samples in the current rehearsal buffer, D_{reh_buf}, current model parameter, $\boldsymbol{\theta}_{t-1}$, objective function, $\mathcal{J}$, the number of poisoned samples at present, $count\_poi$

```
1: $D_{curr} \leftarrow D_{cache_{t-1}} \cup D_{reh\_buf}$
2: $\boldsymbol{x}_{max\_gradient} = argmax_{x_i \in D_{curr}} \left\| \frac{\partial \mathcal{J}(D_{curr}, \boldsymbol{\theta}_{t-1})}{\partial \boldsymbol{x}_i} \right\|_2$
3: if $\boldsymbol{x}_{max\_gradient}$ $not\ in\ D_{reh\_buf}$:
4:     if $len(D_{reh\_buf}) == count\_poi$:
5:         $\boldsymbol{x}_{min\_gradient} = argmin_{x_i \in D_{reh\_buf}} \left\| \frac{\partial \mathcal{J}(D_{reh\_buf}, \boldsymbol{\theta}_{t-1})}{\partial \boldsymbol{x}_i} \right\|_2$
6:         $index_{min\_gradient} = D_{reh\_buf}.index(\boldsymbol{x}_{min\_gradient})$
7:         $D_{reh\_buf}[index_{min\_gradient}] = \boldsymbol{x}_{max\_gradient}$
8:     esle:
9:         $D_{reh\_buf}.append(\boldsymbol{x}_{max\_gradient})$
10:    end if
11: end if
12: return $D_{reh\_buf}$
```

5.2. Theoretical analysis of complexity

According to Eq (14), the complexity of calculating the gradient for each sample point is $O(k^2)$, where k denotes the dimension of the feature vector. Based on lines 2 and 5, the complexity of Algorithm 2 is $O((b + 2count\_poi)k^2)$. According to lines 11 and 14 in Algorithm 1, the complexity of Algorithm 1 is $O(t(2b + 3count\_poi)k^2)$. Assuming that the total number of samples in Algorithm 1 converges to n, then $t = \frac{n}{b}$, and at the same time, $count\_poi < b \ll n$. Therefore, the complexity of Algorithm 1 is $O(nk^2)$ approximately. The complexity of the offline DPA with the same number of samples is $O(iter\_num * n * k^3)$, where $iter\_num$ in the offline mode refers to the rounds of updating poisoned sample features according to the gradient ascent direction. Although the basic online DPA also has a complexity of $O(nk^2)$, it requires full storage of samples, resulting in a storage space of $O(n)$, while Algorithm 1 only requires storage space of $O(b + 2count\_poi)$.

Proof 1. $O(t(2b + 3count\_poi)k^2) < O(iter\_num * n * k^3)$.

Substituting $t = \frac{n}{b}$ into $O(t(2b + 3count\_poi)k^2)$, we get $O(t(2b + 3count\_poi)k^2) = O((2n + 3count\_poi * \frac{n}{b})k^2)$. Since $count\_poi < b \ll n$, we get $\frac{count\_poi}{b} < 1$. Then, $O((2n + 3count\_poi * \frac{n}{b})k^2) < O(5nk^2)$. By omitting the constant terms in the above inequality, we have $O(t(2b + 3count\_poi)k^2) < O(nk^2)$. Since $nk^2 < n * k^3$ and $iter\_num > 1$, we can obtain $O(t(2b + 3count\_poi)k^2) < O(nk^2) < O(iter\_num * n * k^3)$.

Proof 2. $O(b + 2count\_poi) < O(n)$.

$\frac{b+2count\_poi}{n} = \frac{b}{n} + 2\gamma$. Since $count\_poi < b \ll n$, we have $\frac{b}{n} \ll 2\gamma$. Therefore, $\frac{b+2count\_poi}{n} \approx 2\gamma$. Due to the constraint of the concealment condition of the attack, $\gamma < \frac{1}{2}$. Then, $\frac{b+2count\_poi}{n} < 1$. Thus, we can obtain $O(b + 2count\_poi) < O(n)$.

In summary, in terms of time and space complexity, Algorithm 1 has advantages compared to offline DPA and basic online DPA.

6. Experiment

This section evaluates the effectiveness of the rehearsal-based online DPA (RB-ODPA) with maximum-gradient-based point selection strategy and maximum-loss-based pollution strategy when applied to edge devices. The definitions of abbreviations used in this paper are shown Table 2.

Table 2. List of abbreviations.

Abbreviations	Description
DPA	Data poisoning attacks
LOT	Loss over time (LOT)
MSE	Mean squared error
OptP	Best performing optimization attack proposed by [12]
IA-ODPA	Incremental attack proposed by [21]
RB-ODPA	The rehearsal based online DPA proposed of this paper
SVM	Support vector machine

Experimental setup. In order to simulate the edge computing environment of IoT-enabled smart city, the attack algorithm was run in Linux OS in edge-embedded boards, which were mainly configured with a main chip with a cortex-A7 core, 1.2 GHz, 256 MB RAM and 512 MB ROM. The evaluation metrics are same as our previous work [24], which mainly include MSE loss, the running time of attack and the LOT. This paper adopts the OptP method proposed in [12] as the baseline algorithm of offline DPA and the incremental attack method (abbreviated as IA-ODPA) proposed in [21] as the baseline algorithm of basic online DPA.

Data set. To validate the application scenario of IoT-enabled smart cities, we selected data from intelligent power systems, which contains 9568 data samples. The features include the average temperature of the environment per hour, the average pressure of the environment per hour, the average relative humidity of the environment per hour, the exhaust vacuum per hour and the predicted label, which is the net energy output per hour. To simulate online data streams, we input these samples in batches in accordance with the strategy shown in Figure 3. We performed normalization on all sample values, resulting in the feasible range of features and labels being [0,1]. This normalization process ensured consistency in the range of values for both features and labels.

Basic parameters settings. In experiments, we set poisoning rate at 5, 10, 15 and 20%. The termination condition (ε) for algorithm convergence was set to 1e-8. The decay parameter (β) and learning rate (α) for polluting feature values of the poisoned sample points in the direction of gradient ascent was set to 0.05 and 0.01 respectively.

6.1. The relationship between cache size setting and point selection

In this section, we investigate the differences between offline and online modes of sample selection and study the impact of different cache sizes on sample selection in the online mode. We compare the time cost and selection accuracy of the basic online sample selection, offline sample selection and the sample selection method based on the rehearsal buffer. The experimental results are presented in Tables 3 and 4. The offline sample selection is performed by our proposed method with a given poisoning rate to select poisoning points once from the entire sample set, i.e., the execution results of Algorithm 2 when the cache size *b* = 9568.

Table 3. Comparison of sample selection.

Poison rate	Offline and basic online	Rehearsal based		
		b = 2300	b = 3100	b = 4700
0.05	478	427	442	457
0.1	956	866	887	918
0.15	1435	1299	1335	1368
0.2	1913	1773	1801	1844

Table 4. Comparison of sample selection time.

Poison rate	Time of basic methods (s)		Time of rehearsal based online (s)		
	Offline	Basic online	b = 2300	b = 3100	b = 4700
0.05	7.413719	14.827438	2.288950	3.085107	4.677420
0.1	7.41804	14.83608	2.686006	3.620269	5.488794
0.15	7.251866	14.503732	3.235627	4.361063	6.611934
0.2	7.365552	14.731104	3.810749	5.136228	7.787184

Table 3 shows the poisoning rate in the first column, followed by the number of selected poisoning sample points in offline and basic online modes in the second column. The third to fifth columns illustrate the count of same sample points selected by our proposed selection method and offline mode under different cache size settings. From the results presented in Table 3, we can observe that the similarity between the selected samples by our proposed method and those selected in offline mode can reach over 95% by adjusting the size of the cache for different poisoning rates. Figure 5 provides a more intuitive representation of the results in Table 3, where our proposed selection method can select sample points that are relatively close to those selected in offline mode when the cache sizes are 2300, 3100 and 4700.

Table 4 compares the time cost of sample selection methods between offline and online modes. The first column shows the poisoning rate, the second column shows the time cost of sample selection in offline mode, the third column shows the time cost of basic online DPA for sample selection and the fourth to sixth columns show the time cost of our proposed sample selection method under different cache size settings. From the results presented in Table 4, we can observe that our proposed sample selection method has a lower time cost compared to the first two methods, which is consistent with the results of the algorithm complexity analysis presented in Section 5. Figure 6 presents the time cost of our proposed sample selection method under different cache size settings for a poisoning rate of 0.05.

We can observe that as the cache size increases, the overall execution time of the algorithm increases. However, when the cache size is set to 2300, 3100 and 4700, the algorithm can select the maximum number of sample points that are the same as those selected in offline mode with a relatively small time cost.

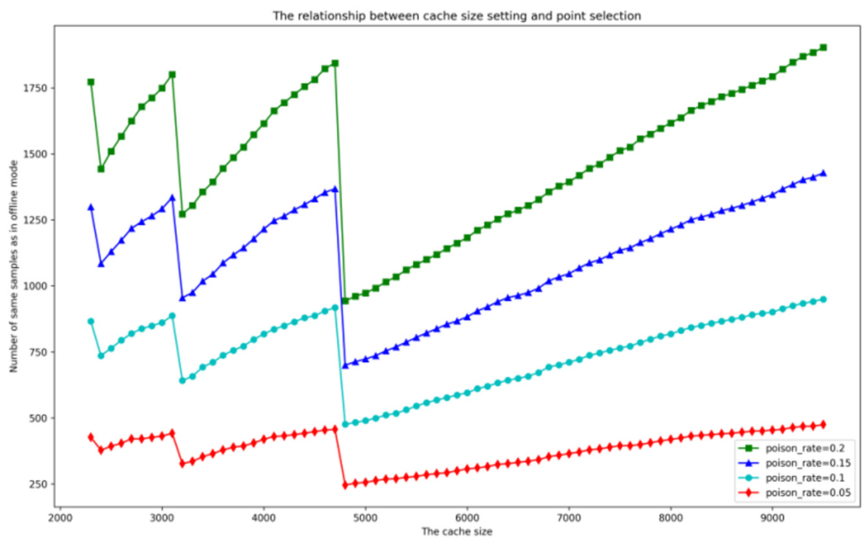


Figure 5. The variation of the same samples number with cache size.

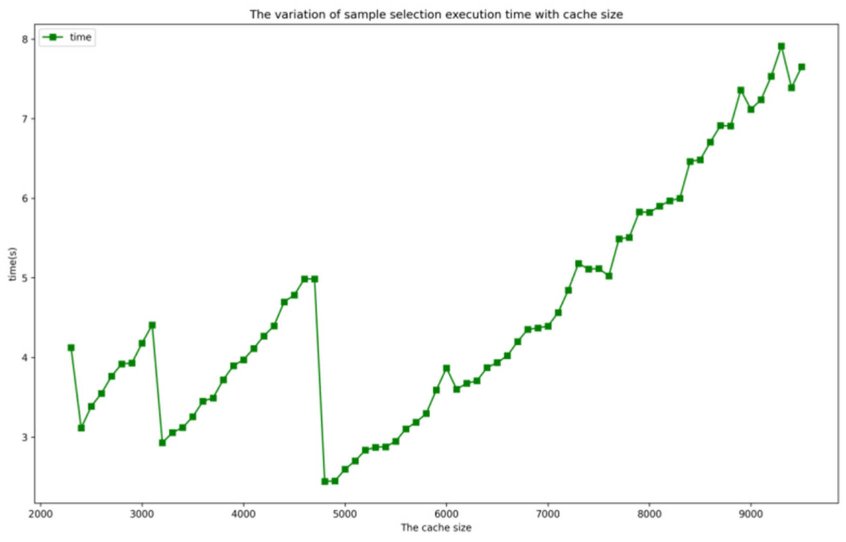


Figure 6. The variation of execution time of sample selection with cache size (poisoning rate = 0.05).

6.2. Effectiveness comparison of proposed attacks and baseline attacks

In this section, we conducted a detailed analysis of the MSE, time and LOT of the three algorithms, based on the experimental settings described in the previous section. The results are presented in Tables 5 and 6 and the comparative values of LOT are in Table 7.

It is observed that all three attacks can mislead the predictive performance and the change in MSE is also linear and upward with the increase in poisoning rates. The red line in Figure 7, representing the RB-ODPA algorithm, shows the best performance with the highest loss, which demonstrates the effectiveness of our method.

Table 5. MSE comparison of attacks.

Poison rate	MSE			
	Unpoison	OptP	IA-ODPA	RB-ODPA
0.05	0.0035	0.017	0.006	0.019
0.1	0.0035	0.024	0.015	0.031
0.15	0.0035	0.036	0.021	0.040
0.2	0.0035	0.043	0.029	0.052

Table 6. Time cost and LOT of sample selection under different cache size.

Poison rate	Time of attack (s)	Time of selection（s）			LOT		
		b = 2300	b = 3100	b = 4700	b = 2300	b = 3100	b = 4700
0.05	4.343159	2.288950	3.085107	4.677420	0.00286485	0.002557797	0.002106295
0.1	8.686318	2.686006	3.620269	5.488794	0.002725916	0.002518976	0.002186932
0.15	13.029477	3.235627	4.361063	6.611934	0.002459253	0.002300101	0.002036514
0.2	17.372636	3.810749	5.136228	7.787184	0.002454754	0.002310201	0.002066787

Table 7. Comparison of attack time and LOT (cache size b = 4700).

Poison rate	Time of method (s)			LOT		
	OptP	IA-ODPA	RB-ODPA	OptP	IA-ODPA	RB-ODPA
0.05	8.67830975	14.827438	4.343159 + 3.085107	0.0019589	0.0004046	0.0026027
0.1	17.3566195	14.83608	8.686318 + 3.620269	0.0013827	0.0010110	0.0025619
0.15	26.03492925	14.503732	13.029477 + 4.361063	0.0013827	0.0014479	0.0023121
0.2	34.713239	14.731104	17.372636 + 5.136228	0.00123872	0.0019686	0.0022222

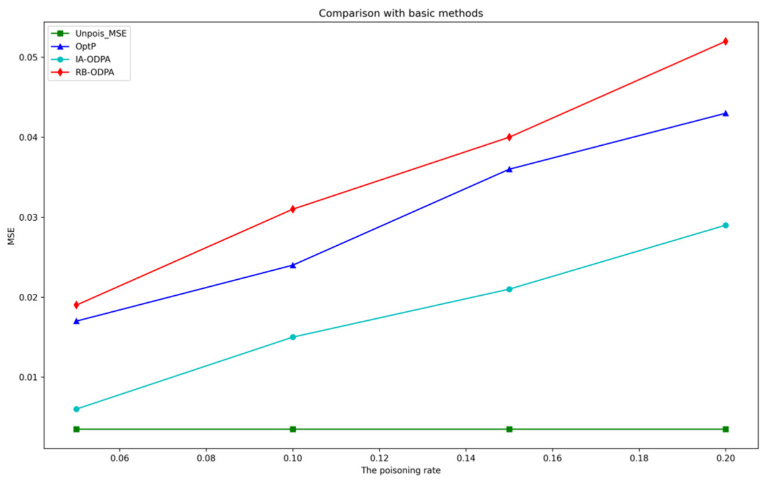


Figure 7. MSE comparison of attacks.

Table 7 indicates that the selection of attack algorithms should not be limited to the degree of improvement in MSE alone, but should consider LOT comprehensively. It clearly shows the performance comparison of the three algorithms in the LOT index, in other words, RB-ODPA achieves the maximum MSE loss with the minimum time cost.

6.3. Optimal setting cache size

In this section, we investigate the issue of cache size setting and evaluate the attack effectiveness of our proposed method under different cache size settings using the LOT metric. The experimental results are presented in Table 6 and Figure 8. According to the experimental results, our proposed attack method outperforms other methods in terms of the LOT metric. Our proposed method also exhibits different LOT results under different cache size settings, with a cache size setting of b = 2300 yielding the optimal attack effectiveness. Therefore, we set this as the optimal cache size setting for our proposed attack method.

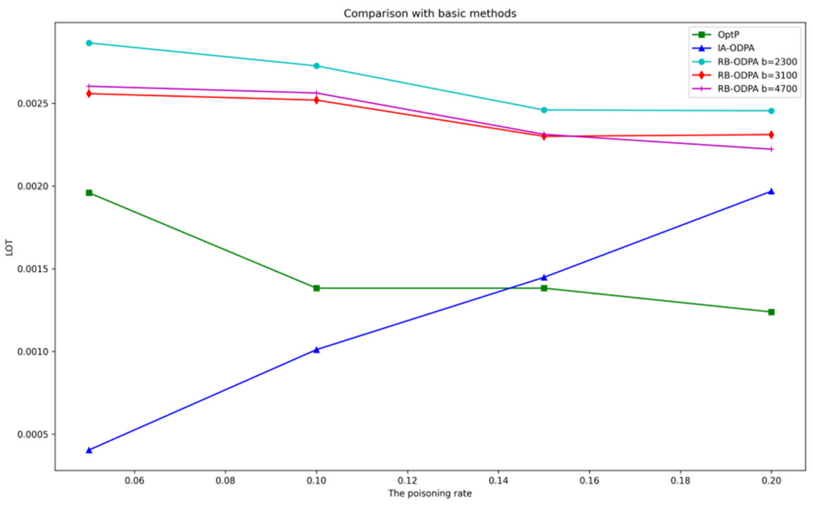
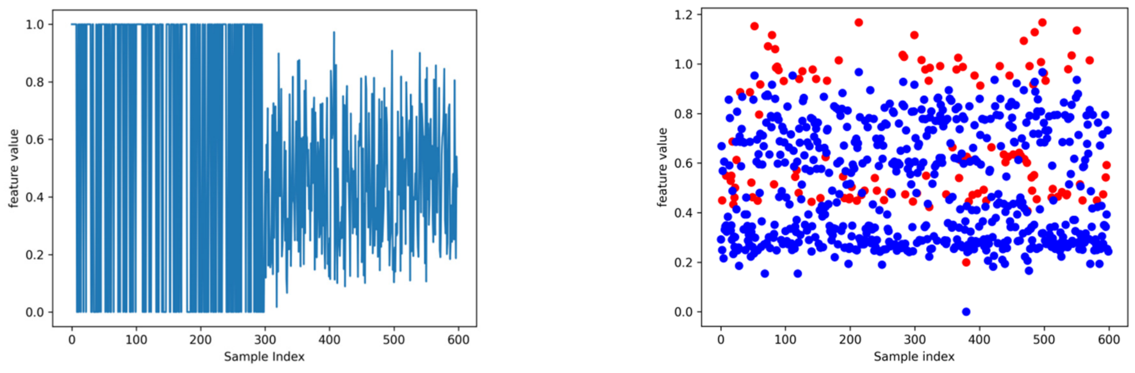


Figure 8. LOT comparison of attacks.

6.4. Comparison of actual attack effects

Figure 9 illustrates the attack effectiveness of offline DPA and online DPA on a real system, with Figure 9(a) showing the offline attack effectiveness and Figure 9(b) showing the online attack effectiveness. In Figure 9(a), the offline attack method places the poisoned samples at the beginning (or end) of the normal samples, resulting in continuous abnormal values at the beginning, which is easily exposed. In contrast, in Figure 9(b), the blue dots represent normal sample values, while the red dots represent the values of poisoned samples after attack. Since the poisoned samples are mixed with normal samples, the offline attack is more covert.



(a) Offline (b) Online

Figure 9. Comparison of offline and online DPA attack effects.

7. Conclusions

In conclusion, this paper proposes an online poisoning attack framework for edge AI environments in IoT-enabled smart cities, which takes into account the limited storage space in the AI edge environment and proposes a rehearsal-based buffer mechanism to incrementally pollute the sample data stream that arrives at the appropriately sized cache to optimize the efficiency of the attack. A maximum-gradient-based sample selection strategy is proposed to overcome the periodic overwriting of the sample data cache after training, while a maximum-loss-based sample pollution strategy solves the problem of each selected sample being polluted only once in the gradient ascent direction in basic online DPA. The proposed online gray-box poisoning attack algorithms are implemented and evaluated on edge devices of IoT-enabled smart cities using an online data stream simulated with offline open-grid datasets. The experimental results demonstrate the effectiveness and overhead of the proposed attack framework and strategies, which can provide a reference for researchers to design defenses against such attacks.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work is supported by National Key R&D Program of China (No.2019YFB1803204).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. Edge AI and Vision Alliance, 2023 Edge AI Technology Report, 2023. Available from: https://www.edge-ai-vision.com/2023/07/2023-edge-ai-technology-report/.

2. Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *Nature*, **521** (2015), 436–444. https://doi.org/10.1038/nature14539
3. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Edge intelligence: Paving the last mile of artificial intelligence with edge computing, in *Proceedings of IEEE*, **107** (2019), 1738–1762. https://doi.org/10.1109/JPROC.2019.2918951
4. Z. Zhou, Y. Shuai, X. Chen, Edge intelligence: a new nexus of edge computing and artificial intelligence, *Big Data Res.*, **5** (2019), 53–63. https://doi.org/10.11959/j.issn.2096-0271.2019013
5. X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, X. Chen, Convergence of edge computing and deep learning: A comprehensive survey, *IEEE Commun. Surv. Tutorials*, **22** (2020), 869–904. https://doi.org/10.1109/COMST.2020.2970550
6. S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, A.Y. Zomaya, Edge intelligence: The confluence of edge computing and artificial intelligence, *IEEE Internet Things J.*, **7** (2020), 7457–7469. https://doi.org/10.1109/JIOT.2020.2984887
7. Y. Li, Y. Yu, W. Susilo, Z. Hong, M. Guizani, Security and privacy for edge intelligence in 5G and beyond networks: Challenges and solutions, *IEEE Wireless Commun.*, **28** (2021), 63–69. https://doi.org/10.1109/MWC.001.2000318
8. M. S. Ansari , S. H. Alsamhi, Y. Qiao, Y. Ye, B. Lee, Security of distributed intelligence in edge computing: Threats and countermeasures, in *The Cloud-to-Thing Continuum*, Springer, (2020), 95–122.
9. B. Biggio, B. Nelson, P. Laskov, Poisoning attacks against support vector machines, preprint, arXiv:1206.6389.
10. S. Mei, X. Zhu, Using machine teaching to identify optimal training-set attacks on machine learners, in *Proceedings of the AAAI Conference on Artificial Intelligence*, **29** (2015), 2871–2877. https://doi.org/10.1609/aaai.v29i1.9569
11. N. Müller, D. Kowatsch, K. Böttinger, Data poisoning attacks on regression learning and corresponding defenses, in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, (2020), 80–89. https://doi.org/10.1109/PRDC50213.2020.00019
12. M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, B. Li, Manipulating machine learning: Poisoning attacks and countermeasures for regression learning, in *2018 IEEE Symposium on Security and Privacy (SP)*, (2018), 19–35. https://doi.org/10.1109/SP.2018.00057
13. T. Cerquitelli, M. Meo, M. Curado, L. Skorin-Kapov, E. E. Tsiropoulou, Machine learning empowered computer networks, *Comput. Networks*, **230** (2023), 109807. https://doi.org/10.1016/j.comnet.2023.109807
14. P. W. Koh, J. Steinhart, P. Liang, Stronger data poisoning attacks break data sanitization defenses, *Mach. Learn.*, **111** (2022), 1–47. https://doi.org/10.1007/s10994-021-06119-y
15. C. Burkard, B. Lagesse, Analysis of causative attacks against SVMs learning from data streams, in *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*, (2017), 31–36. https://doi.org/10.1145/3041008.3041012
16. X. Zhang, X. Zhu, L. Lessard, Online data poisoning attack, preprint, arXiv:1903.01666.
17. P. G. Margiotta, S. Goldt, G. Sanguinetti, Attacks on online learners: A teacher-student analysis, preprint, arXiv:2305.11132.
18. Z. Hammoudeh, D. Lowd, Training data influence analysis and estimation: A survey, preprint, arXiv:2212.04612.

19. M. Wojnowicz, B. Cruz, X. Zhao, B. Wallace, M. Wolff, J. Luan, et al., "Influence sketching": Finding influential samples in large-scale regressions, in *2016 IEEE International Conference on Big Data (Big Data)*, (2016), 3601–3612. https://doi.org/10.1109/BigData.2016.7841024
20. P. W. Koh, P. Liang, Understanding black-box predictions via influence functions, preprint, arXiv:1703.04730.
21. Y. Wang, K. Chaudhuri, Data poisoning attacks against online learning, preprint, arXiv:1808.08994.
22. M. A. Ramirez, S. Kim, H. A. Hamadi, E. Damiani, Y. J. Byon, T. Y. Kim, et al., Poisoning Attacks and Defenses on Artificial Intelligence: A Survey, preprint, arXiv:2202.10276.
23. L. Bottou, Large-scale machine learning with stochastic gradient descent, in *Proceedings of COMPSTAT'2010*, (2010), 177–186. https://doi.org/10.1007/978-3-7908-2604-3_16
24. Y. Zhu, H. Wen, R. Zhao, Y. Jiang, Q. Liu, P. Zhang, Research on data poisoning attack against smart grid cyber-physical system based on edge computing, *Sensors*, **23** (2023), 4509. https://doi.org/10.3390/s23094509