Journal of Intelligent Computing and Networking
https://www.ffspub.com/index.php/jicn/index
ISSN 3079-9228 (print)
E-mail: jicn.office@ffspub.com

Article

Exploring A Model Type Detection Attack against Machine Learning as A Service

Yilong Yang[1], Xinjing Liu[2,†], Ruidong Han[3], Yang Liu [2]
[1] *School of Artificial Intelligence, Xidian University, Xi'an 710126, China*
[2] *School of Cyber Engineering, Xidian University, Xi'an 710126, China*
[3] *School of Computing and Information Systems, Singapore Management University, 188065, Singapore*
[†] *E-mail: liuxinjing_j@163.com*

Received: August 30, 2025 / Revised: November 1, 2025 / Accepted: November 11, 2025 / Published online: November 17, 2025

Abstract: Recently, Machine-Learning-as-a-Service (MLaaS) systems are reported to be vulnerable to varying novel attacks, e.g., model extraction attacks and adversarial examples. However, in our investigation, we notice that the majority of MLaas attacks are not as threatening as expected due to model-type-sensitive problem. Literally speaking, many MLaaS attacks are designed for only a specific type of models. Without model type info as default prior knowledge, these attacks suffer from great performance degradation, or even become infeasible! In this paper, we demonstrate a novel attack method, named SNOOPER, to resolve the model-type-sensitive problem of MLaaS attacks. Specifically, SNOOPER is integrated with multiple self-designed model-type-detection modules. Each module can judge whether a given black-box model belongs to a specific type of models by analyzing its query-response pattern. Then, after proceeding with all modules, the attacker can know the type of its target model in the querying stage, and accordingly choose the optimal attack method. Also, to save budget, the queries can be re-used in the latter attack stage. We call such a kind of attack as model-type-detection attack. Finally, we experiment with SNOOPER on some popular model classes, including decision trees, linear models, non-linear models and neural networks. The results show that SNOOPER is capable of detecting the model type with more than 90% accuracy.

Keywords: Machine learning security; deep learning models; machine learning as a service; model type detection; linear models
https://doi.org/10.64509/jicn.12.27

1 Introduction

Advancements in machine learning and cloud computing promote the development of Machine Learning-as-a-Service (MLaaS) platforms [1, 2]. By providing pay-per-query interfaces to well-trained models, MLaaS systems support a wide range of service instances for users, such as image analyzing and predicting [3]. However, the success of MLaaS not only leads to its popularity in applications but also attracts the interest of attackers. In the past decade, multi-type attacks against different machine learning models are proposed to breach the security of MLaaS systems, e.g., model extraction attacks (MEA) [4–6] and adversarial example (AE) attacks [7, 8].

1.1 Really Black-Box?

To enclose real-world attack scenarios, many state-of-the-art attacks against MLaaS are claimed implement in the black-box setting [4, 5, 9, 10]. Despite oracle access to MLaaS interfaces, these black-box attackers are claimed to obtain no prior knowledge of the victim models. However, an easily overlooked condition can breach the "black-box" claim, that is, the MLaaS attacks are usually conducted with the prior knowledge of the victim model's type. A typical example is MEA [4, 5, 9], in which each model extraction method is designed against only one type of model. Thus, in practice, the first default step of MEA is actually to choose a proper model extraction method according to the model type info. We call this kind of MLaaS attacks to be model-type-sensitive attacks. For these attacks, if victim interfaces hide mode type info

† Corresponding author: Xinjing Liu
*Academic Editor: Xiuli Bi

Table 1: Examples of MLaaS attacks against different types of models. Most attacks listed in the table are model-type-sensitive.

		Model Extraction	Adversarial Example	Model Inversion
Linear Model		Reverse engineering [11] Equation-solving [4]	Ensemble word addition [12]	Exhaustive search [13]
Decision Tree		Path-finding [4] Importance weighted active learning [5]	Zeroth order optimization [14, 15]	Map inverter [16]
Kernel Model		Extended adaptive training [5]	Transferability based attack[1] [17]	GAN-based attack [18] Style-GAN based attack [19]
Neural Network	MLP	Non-linear equation solving [20] Model distillation [21]	Square attack [22] Transferability based attack [23, 24] Substitute Training [25]	
	RNN	Output biasing [26]		
	CNN	Active learning [20] Copying with non-label data [27]		

[1] [17] shows the possibility to attack KM based on AE transferability, but some measurements to improve transferability (for NN) cannot be used in KM attack, e.g., the attention mechanism in [23]. Thus, we repeat it in two cells.

(like what is done by Google and Amazon AI clouds), detecting the target model type becomes the essential step to ensure attack effectiveness and black-box characteristic.

1.2 Our Work

To resolve the model-type-sensitive problem MLaaS attacks, we propose a novel attack, called model-type-detection attack (MTD-A). Our work of this paper is summarized as follows.

We propose a novel MTD-A method, called SNOOPER, which enables the adversary to run MLaaS attacks in a more restrictive black-box scenario where the model type info of MLaaS interfaces is hidden. In SNOOPER, two parts are involved to achieve model type detection for an unknown classifiers, including an attack sample pool generator (Generator), and a set of detection modules against different types of models (Module). These two parts of SNOOPER are designed to be generic and not based on particular dataset.

Specifically, Generator takes the responsibility for collecting random data points or publicly available raw data according to the requirement of Module. Module consists of multiple detection modules, each of which can answer whether a given black-box model belongs to a specific type of models by analysing its query-response pattern. Then, the adversary can select a proper attack method to launch attacks according to the judgement result of Module. For example, suppose an adversary wants to steal a model from a MLaaS platform and Module tells that there may be a linear model behind the target interface. Then, the adversary can directly invoke the linear model extraction method [4], rather than inefficiently trying every other way to find the effective one.

As the first attempt, SNOOPER focuses on developing four modules to detect four classes of models commonly discussed in MLaaS attacks, including decision trees (DTs) [28], linear models (LMs) [4], models with the non-linear kernel (KMs) [20] and neural networks (NNs) [29]. These modules are loosely coupled, and mainly implemented based on the unique characteristics of different models. Take DTs as an example. DTs are distinguished from other models because of its unique discrete mathematical structure. Hence, designing a simple discreteness test is the key point to implement a DT module. Moreover, since the NN structure can also affect the effectiveness of some attacks (discussed in Table 1), we also explore the feasibility to judge common NN architectures based on the transferability of AEs in the NN module.

Summary. Our contributions are summarized as follows.

- We propose a novel method, SNOOPER, to implement a first-of-its-kind MTD attack. This attack can break the model type privacy, and assist in current model-type-sensitive MLaaS attacks to work better in a more restrict condition.
- We design four model type detection modules in SNOOPER. These modules can be used to achieve MTD-A on four most commonly types of models, namely DTs, LMs, KMs and NNs. The modules show that model type privacy can be easily broken by utilizing model's inherent mathematical characteristics.
- We test SNOOPER on three real-world MLaaS platforms, including Google Cloud, Amazon Web Services and Huawei Cloud. Experimental results show that SNOOPER can always maintain more than 90% detection accuracy on 975 different models, and is able to save more than half of the query budget for MLaaS attackers compared with one-by-one trying the optimal attack method.

2 Background

In this section, we briefly review some background knowledge to implement SNOOPER.

2.1 Model Extraction Attack

MEA [4, 5] is a kind of attack against the confidentiality of model functionality. Suppose that there is a model owner

provides a pay-per-query interface of its model f. By launching the MEA attack, an adversary $\mathscr{A}$ can fit a local model f^* whose functionality is approximate, or even identical, to f within a limited number of queries such that f^* can be used freely by $\mathscr{A}$. Clearly, the proposal of MEA breaches the model privacy of MLaaS platforms.

More formally, the MEA process is illustrated as follows [5]. Fix a public hypothesis class $\mathbf{c} = \{f : \mathscr{X} \to \mathscr{Y}\}$ and an error function $Err : \mathbf{c} \times \mathbf{c} \to \mathbb{R}$. Let a specific model $f \in \mathbf{c}$ be deployed on the MLaaS interface. $\mathscr{A}$ tries to extract f by interacting with the interface within the maximum query budget q. The extraction process proceeds as:

1. $\mathscr{A}$ is given a description of $\mathbf{c}$ (including task info and model type) and oracle access to f, and then, sends query $x \in \mathscr{X}$ to get back $y \leftarrow f(x) \in \mathscr{Y}$. With at most q queries, $\mathscr{A}$ outputs the extracted model f^*.
2. If $Err(f, f^*)$ is less than the desired error bound ε, the attack is considered to a success.

As shown in Table 1, four types of models are commonly discussed in MEA, namely LMs (e.g, logistic regression), KMs, DTs, NNs. Among them, LMs can be directly extracted via equation-solving. Extraction of DTs executes path-finding or active learning. For NNs, different attack methods are used for different networks, like convolutional NN extraction [30] and recurrent NN extraction [26]. Particularly, an MEA attack method against a class of models can hardly be applied to the other type of models. For example, it is almost impossible to use the equation-solving method for LM extraction to extract an NN model due to the disparate parameter sizes of the two types of models. Thus, the above-mentioned MEA definition assumes a public description of the hypothesis class $\mathscr{F}$ to ensure attack effectiveness.

2.2 Adversarial Examples

Another research hotspot on MLaaS attacks is the adversarial example (AE) attack [31] whose definition is given below.

Definition 1 (Adversarial Example) Given an input $x \in \mathscr{X}$ labeled as $y \in \mathscr{Y}$ and a victim model f, an effective adversarial example attack allows $\mathscr{A}$ to explore an optimized perturbation δ about x such that the crafted sample $x+\delta$ is visually indistinguishable from x but can mislead f to output $y^* = f(x+\delta) \neq f(x)$. The optimization process to find minimized δ can be formulated as:

$$\underset{\delta}{\text{minimize}} \quad f(x+\delta, y^*) \quad \text{s.t. } x+\delta \in \mathscr{X}, \|\delta\|_\infty \leq \varepsilon, \tag{1}$$

where $\|\cdot\|_\infty$ denotes the ℓ_∞ norm item [32] to measure the degree of perturbation and ε is the perturbation bound.

Most existing AE attacks implemented in the black-box setting can be classified into two categories. The first is query-response based AE attacks. $\mathscr{A}$ directly extracts useful info from the outputs of the remote model to fit the perturbation noise via craftily designed optimization methods, e.g., zero-order gradient optimization [33]. The other is based on the transferability of AEs [34]. Instead of directly attacking the model hidden in the MLaaS interface, the transferability-based attacker launches white-box attacks on locally trained models to find AEs. Then, with probabilities, the locally generated AEs can also be used to fool the remote model. AEs often stay close to the distribution of normal data, making them highly subtle and difficult to detect [35]. Intuitively, the more similar the architecture of the local model is to that of the remote model, the more likely the attack can be effective. Thus, as the model architecture info is given, the adversary can usually generate more threatening AEs [34].

2.3 Other Attacks

Besides the above-mentioned two attacks, there are also other typical MLaaS attacks like membership inference attack (MIA) [36] and update leakage attack [37]. Since these attacks are not directly related to SNOOPER, we omit their detailed description for brevity. Table 1 reveals the fact that most MLaaS attacks have strong "exclusivity". That is, many MLaaS attack methods have their own unique characteristics, and thus, are only workable to one type of models. Due to the characteristic, the adversary $\mathscr{A}$, which wants to attack a MLaaS interface whose model type info is hidden, needs to try each attack method and searches for the most performable one. Clearly, such a straightforward attack method can increase the query complexity significantly and lowers the practicality of most attacks.

3 Problem Formulation

3.1 Threat Model

This work focuses on MTD-A and corresponding defenses in MLaaS systems. Here, we defines the attacker's capabilities, the defender's assumptions, and the applicable scope of the proposed framework.

Attacker Capability. The adversary is assumed to have black-box access to an online MLaaS model, meaning only query–response interactions are available. The attacker can submit crafted inputs within the normal service interface and collect outputs, which may include prediction probabilities (soft labels) or discrete class indices (hard labels). No internal parameters, gradients, or training data are accessible. The attacker's goal is to infer the underlying model type among several common categories.

Boundary of Applicability. SNOOPER currently targets four widely used model classes in MLaaS applications: (1) Decision Trees (DT), (2) Linear Models (LM), (3) Kernel Models (KM), and (4) Neural Networks (NN), including feed-forward (FNN), convolutional (CNN), and recurrent (RNN) architectures. These categories are chosen because they share the same input–output modalities and can perform similar supervised classification tasks, making them indistinguishable by task semantics alone. Other "proprietary" models with inherently different input–output structures—such as Reinforcement Learning (RL) [38] or Graph Neural Networks (GNNs) [39] agents—are beyond the current attack scope, as their data formats (e.g., node–edge pairs, state–action trajectories) already reveal model semantics and make MTD-A inapplicable. This is because for most cases, these models can be directly distinguished based on the learning tasks. For example, if the target interface is provided for social relation analysis with graph data, it is sure to be a graph model.

Defender Capability. The MLaaS provider aims to protect the confidentiality of model architectures deployed on cloud platforms (e.g., Google Vertex AI, AWS SageMaker,

Huawei Cloud). The proposed defense modules in SNOOPER are designed to detect and mitigate such MTD-A attempts through behavior regularization and query monitoring.

3.2 MTD-A

Prior to dealing with MTD-A, we first formulate model type detection problem (also illustrated in Figure 1).

We begin by describing the prototype of MTD-A in the context of MLaaS systems. The difference is that the MLaaS platform no longer provides a public model type description of the query interface but treats it to be private info. In this way, what a user can get from the interface is limited to be the oracle access to the model f provided by the interface, q query-response pairs $\{(x_i, y_i)|i \in [q]\}$, the input space $\mathscr{X}$, and the output space $\mathscr{Y}$, where q is the maximum query budget.

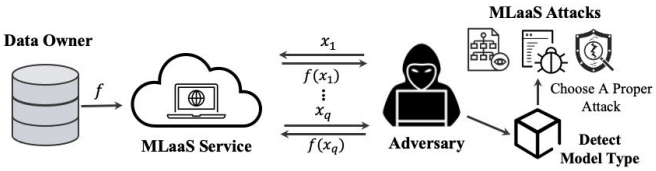


Figure 1: Illustration of MTD-A.

Formally, the model type detection process can be simulated by the following experiment. Denote a set of model classes as $\mathscr{C} = \{c_1, ..., c_n\}$. Fix a hypothesis model class $\mathbf{c} = \{f : \mathscr{X} \to \mathscr{Y}\}$ randomly selected from $\mathscr{C}$. Let a specific model $f \in \mathbf{c}$ be deployed on the MLaaS interface. Then, the adversary $\mathscr{A}$ tries to give an answer $c' \in \mathscr{C}$ by interacting with the interface of f within the maximum query budget q. The experiment $\text{EXP}_{\mathscr{C}}(f, \mathscr{A}, \mathbf{c}, q)$ considers to be successful if the answer c' is equal to $\mathbf{c}$. Specifically, $\text{EXP}_{\mathscr{C}}(f, \mathscr{A}, \mathbf{c}, q)$ proceeds as follows.

1. $\mathscr{A}$ is granted full oracle access to f and a description of $\mathscr{C}$ but without any description of $\mathbf{c}$. Then, $\mathscr{A}$ sends queries $x \in \mathscr{X}$ to get back $y \leftarrow f(x) \in \mathscr{Y}$. With at most q queries, $\mathscr{A}$ gives an answer $c' \in \mathscr{C}$ about the target model type.
2. The output of $\text{EXP}_{\mathscr{C}}(f, \mathscr{A}, \mathbf{c}, q)$ is 1 if c' is equal to $\mathbf{c}$, otherwise, the output is 0.

Given the above experiment $\text{EXP}_{\mathscr{C}}(f, \mathscr{A}, \mathbf{c}, q)$, we can further derive the following definition about MTD-A.

Definition 2 (Model Type Detection Attack) Suppose $\mathscr{C}$ to be a set of model classes. We say that an adversary that has black-box access to an MLaaS interface implements $\mathscr{C}$-MTD of complexity q and confidence p against class set $\mathscr{C}$ if there exists $\Pr[\text{EXP}_{\mathscr{C}}(f, \mathscr{A}, \mathbf{c}, q) = 1] \geq p$, for $f \in \mathbf{c}$ and $\mathbf{c} \in \mathscr{C}$.

Definition 2 constrains the range of model classes involved in a model type detection attack to be within $\mathscr{C}$. The constraint is reasonable because based by experience, the alternative types of models for common learning tasks are usually limited to a specific range. Moreover, the definition indicates that model type detection can be quantitatively evaluated by the query budget q and confidence bound p as enough evaluation instances are sampled from $\mathscr{C}$.

Attack Goal. Based on Definition 2, besides low query complexity, the other attack goal for $\mathscr{A}$ is to achieve high model type detection accuracy. The model type detection accuracy can be evaluated from two perspectives.

- **Overall Evaluation.** Given a fixed class set $\mathscr{C}$, overall evaluation indicates the detection confidence over the whole set. The evaluation indicator p_{all} is computed by $p_{all} = \Pr[\text{EXP}_{\mathscr{C}}(f, \mathscr{A}, \mathbf{c}, q) = 1]$. High p_{all} implies that the attack method can distinguish any type of model in $\mathscr{C}$ with high accuracy.
- **Module Evaluation.** Assume a subset $\mathscr{C}' \subset \mathscr{C}$, and limit the output of $\mathscr{A}$ to be within $\mathscr{C}' \cup \perp$ where $\perp$ means unknown types of models. Correspondingly, the experiment condition is changed to that if the answer c' of $\mathscr{A}$ is equal to $\perp$ and $\mathbf{c} \in \mathscr{C}/\mathscr{C}'$, $\text{EXP}_{\mathscr{C}'}(f, \mathscr{A}, \mathbf{c}, q)$ also outputs 1. At this time, we call the evaluation result $\Pr[\text{EXP}_{\mathscr{C}}(f, \mathscr{A}, \mathbf{c}, q) = 1]$ to be the module confidence p_{mod}. Here, p_{mod} can be used to evaluate whether each attack module in SNOOPER can judge the model type with high confidence effectively and independently. Note that the independence of the attack modules counts a lot for the practicality of MTD-A in applications. Only independent modules can be used in any combinations to launch attacks for different application scenarios.

4 Design of SNOOPER

In this section, we detail our design of SNOOPER.

4.1 Outline of SNOOPER

At a high level, our framework, SNOOPER, can be outlined into two parts, namely an attack sample generator (Generator) and a set of detection modules against different types of models (Module).

Generator. As the prebox, Generator prepares all queries required by Module to detect the target model type. Like other MLaaS attacks, to save query budget, the MTD-A adversary has to carefully craft the query inputs to ensure the maximum info gain for each query. Normally speaking, the queries of MLaaS attacks can be crafted with the data collected from public data resources, or directly synthesized according to specific rules. Especially, since most prior attacks only need to breach the security of one type of models, their query generation rules are usually singular and straightforward. However, for SNOOPER, the target model type is unknown, thereby leading to the involvement of hybrid rules to serve multiple modules against different model types. Thus, we separate Generator as an individual component to achieve: 1) filtering the repeated rules to avoid redundant queries in Module, 2) and collecting all query-response info for repeated usage in the latter attacks.

Module. This component is the core of SNOOPER to breach model type privacy. In Module, multiple modules are involved against different types of models as depicted in Figure 2. Here, four modules against four popular types of models are considered, including the LM module, DT module, KM module and NN module. Each module can answer whether the target model belongs to a specific type of models. The answer of every module does not depends on any result of other modules, except the NN module.

Especially, considering that NN architectures can also affect the effectiveness of some attacks, there are also two sub-modules involved in the NN module to further detect

whether the target NN architecture belongs to full-connection network (FNN), convolutional neural network (CNN) or recurrent neural network (RNN).

Since the data generation rules of Generator totally depend on the design of Module, the following description will start with Module, followed by Generator.

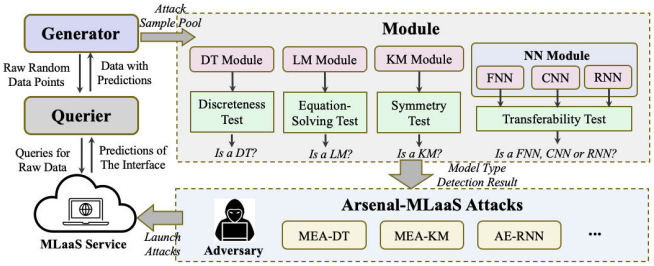


Figure 2: Outline of the attack workflow of SNOOPER. In this figure, MEA-DT, MEA-KM and AE-RNN mean the MEA and AE attack methods (listed in Table 1) that are designed against DT, KM, RNN models, respectively.

4.2 Modules of SNOOPER

Here, the design goal of SNOOPER is to develop a series of modules to pass the experiment $\text{EXP}_{\mathscr{C}_0}(f,\mathscr{A},\mathbf{c},q)$, where $\mathscr{C}_0=\{\mathbf{c}_{\text{DT}},\mathbf{c}_{\text{LM}},\mathbf{c}_{\text{KM}},\mathbf{c}_{\text{NN}}\}$. All modules proceed with the assumption that the interface of f responds with the confidence score for every query [26]. The following presents the details of each module.

4.2.1 Module for Decision Trees (or Forest)

For the DT model (or called discrete model in [40]), one of its characteristics distinguished from other models is to have a finite output space that is determined by the leaf nodes. Based on the characteristic, we have the following observation.

Observation 1 Given a model $f\in\mathbf{c}$, denote its input vector as $x=(x_1,...,x_n)$ where x_i corresponds to the i-th input feature of f. If and only if f is a DT model, it can be formulated as a key-function table $\mathscr{T}=\{(1,\varphi_1),...,(n,\varphi_n)\}$. For $1\le i\le n$, φ_i is a piecewise function:

$$\varphi_i(x_i)=\begin{cases}\mathbb{Y}_1, & \text{if } \beta_{i,0}\le x_i<\beta_{i,1}\\ \dots & \\ \mathbb{Y}_m, & \text{if } \beta_{i,m_i-1}\le x_i<\beta_{i,m_i}\end{cases}, \tag{2}$$

where $\mathbb{Y}$ specifies a finite output (confidence value) space corresponding to the leaf nodes of the DT model, $\mathbb{Y}_i\subset\mathbb{Y},\beta_i\le\beta_{i+1}$, and $[\beta_{i,0},\beta_{i,m_i}]$ denotes the input range of the i-th feature. Then, we can characterize the prediction process of a DT model as:

$$Y=f_{DT}(x)=\varphi_1(x_1)\cap...\cap\varphi_n(x_n),\ \text{s.t.},\ Y\in\mathbb{Y}, \tag{3}$$

Clearly, there can be only one element in Y, corresponding to the leaf node of the DT model on which the input falls.

From the observation, the discreteness of DTs determines that with a high confidence p, there exists a specific feature range that falls in the piecewise function of $\mathscr{T}$, which makes the output Y to not deviate as the input is only changed within the range. We call such a range as constant interval. By utilizing the unique characteristic of DTs, a discreteness test based method is proposed in Algorithm 1 to distinguish DTs with other models in $\mathscr{M}_0$. We say that the test passes if and only if the target model is DTs because any other model in $\mathscr{M}_0$ cannot limit its predictions to be within a finite set. Moreover, the following condition analysis states that the discreteness test can always pass if the target model type is DTs.

- If there exists a continuous feature in $\mathbb{A}$, the test passes, where $\mathbb{A}$ is the feature set of f. This is because with small enough interval and finite DT nodes, the constant interval can always be found over the continuous feature.
- If all features are discrete but one or more of them is not contained in the target model, the test passes. At this time, any input change over the non-related features does not affect the output.
- If all features are discrete, the test only fails as if for every $(i,\varphi_i)\in\mathscr{T}$, the piecewise condition of φ_i satisfies $\beta_{i,j}=\beta_{i,j+1}-1$ for all $0<i\le n, 0\le j<m_i$. Empirically, the failure condition can be hardly satisfied by common DTs.

Above all, we can conclude that SNOOPER can theoretically achieve $\Pr[\text{EXP}_{\{DT,\mathscr{M}_0/DT\}}(f,\mathscr{A},\mathscr{C},q)=1]\approx 1$ via the discreteness test, where $\mathscr{M}_0/DT=\mathscr{M}_0-\{DT\}=\{LM,KM,NN\}$.

Algorithm 1 DT module based on discreteness test.

Require: The target model f, a threshold t to test discreteness, the feature space $\mathbb{A}$ and input space $\mathscr{X}_i$ for each feature in $\alpha_i\in\mathbb{A}$.

Ensure: If the output is 1, f is judged to be a DT model, otherwise, to be others.

Randomly select several data points from $\mathscr{X}_i$ to form an attack pool B.
for each $(b_i,y_i)\in B$ **do**
  Randomly select a subset $\mathbb{A}'\subset\mathbb{A}$.
  Record the label of b_i as y_i and set a counter $count\leftarrow 0$.
  for each $\alpha_j\in\mathbb{A}$ **do**
    Modify the feature value of b_i over α_j at a fixed interval and ensure the modified feature value is still within $\mathscr{X}_i$.
    Query the modified data point through the interface of f to get a new prediction y_i'.
    If y_i' is equal to y_i, update $count\leftarrow count+1$, otherwise, reset $count\leftarrow 0$.
    if $count>t$ **then**
      return 1
    end if
  end for
end for
As the loop is completed, output 0.

4.2.2 Module for Linear Models

In SNOOPER, what is concerned by the LM module is the models that can be uniformly formulated into the following format, e.g., support vector machine (SVM) and logistic regression (LR).

$$f_{LM}(x)=w\cdot x+b, \tag{4}$$

where $w,x\in\mathbb{R}^{kn}$, $b\in\mathbb{R}^k$, $k\ge 1$ indicates $(k+1)$-classification, and n is the feature number. For some learning

tasks, there can also be a sigmoid function [41] to transform the output space, or a linear kernel function to transform the input space. While, referring to [4], all kinds of LMs follow the following observation.

Observation 2 Denote a target model belonging to $\mathscr{M}_0$ as f whose feature space and label space are $\mathbb{A} = \{\alpha_1, ..., \alpha_n\}$ and $\mathscr{L} = \{l_1, ..., l_k\}$, respectively. If and only if f is a LM, we can get a copy f' of f by solving the equations about $k \cdot n + k$ input-output pairs, and f' satisfies:

$$Err(f, f', \mathscr{X}) \leq \varepsilon, \tag{5}$$

where $Err(\cdot)$ is an error function [4] to evaluate the output distance between two models, and ε is a negligible error bound. Literally speaking, in $\text{EXP}_{\mathscr{M}_0}(f, \mathscr{A}, \mathscr{C}, q)$, the target model belongs to LMs iff. it is equation-resolvable.

Based on the above observation, we can simply derive an equation-solving test based LM module, which achieves $\Pr[\text{EXP}_{\{LM, \mathscr{M}_0/LM\}}(f, \mathscr{A}, \mathscr{C}, q) = 1] \approx 1$. Especially, if the target model is trained for a multi-class task, the outputs obtained from the MLaaS interface are usually masked by the softmax function. At this time, as suggested by [4], a reversion function should be fitted to unmask the outputs before solving the equations. Also, the equation-solving test can be efficiently implemented by common optimizers, e.g., BFGS and L-BFGS [42].

4.2.3 Module for Kernel Models

We now turn to the models with non-linear kernels. The kernel function is a common trick to enable LMs to resolve a non-linear problem by transforming original non-linear inputs to a higher-dimensional space that is operational for linear algorithms [43]. Due to the unique characteristic, KMs, as a type of model extended from LMs, are also often discussed separately in some MLaaS attacks [4, 5]. Formally, KMs can be expressed as Equation 6.

$$f_{LM}(x) = w \cdot K(x) + b, \tag{6}$$

where $K(\cdot)$ is the non-linear kernel function. Here, for brevity, we focus on describing the way for the KM module to detect the Radial-Basis Function (RBF) kernel (or its variants like the exponential kernel and Laplacian kernel) based KMs, which is the mostly used in applications. For other kernels, they can be detected with a similar idea.

To identify KMs, the core idea of SNOOPER is to make use of the unique mathematical characteristic of the non-linear kernel function. According to our investigation, RF kernel, and all other Gauss-type kernels, follow the following observation.

Observation 3 Denote a target model belonging to $\mathscr{M}_0$ as f. If and only if it is a KM applied with a Gauss-type non-linear kernel, f has good symmetry. Alternatively, suppose the center point of the RBF kernel to be μ. A KM model ideally satisfies the following condition.

$$f_{KM}(\mu + x) = f_{KM}(\mu - x). \tag{7}$$

Based on Observation 3, it seems that KM module can be simply implemented through symmetry test by selecting proper inputs to compute Equation 7. However, in applications, the central point μ is usually derived from the training data, and can hardly be determined by the adversary. To overcome the problem, we notice that even two inputs x and x' are different, there still exists $K(x) \approx K(x')$ if both of x and x' are close to the $\mathscr{H}$ or $-\mathscr{H}$, where $\mathscr{H}$ represents a huge number. Therefore, instead of directly computing Equation 7, an alternative way is to select $x \to \mathscr{H}$ and $x' \to -\mathscr{H}$ and do the symmetry test defined in Equation 8.

$$\|f_{KM}(x) - f_{KM}(x')\| \leq \varepsilon, \tag{8}$$

where $\|\cdot\|$ is the L_1 function to evaluate the distance between $f(x)$ and $f(x')$, and ε is a small error bound. Suppose the failure rate of each test to be p_{KM}. The KM module achieves $\Pr[\text{EXP}_{\{KM, \mathscr{M}_0/KM\}}(f, \mathscr{A}, \mathscr{C}, q) = 1] = 1 - p_{KM}$.

Remark: In practice, the infinity values can be filled with the boundary values of the input space for most cases.

4.2.4 Module for Neural Networks

When it comes to NN module, what we concern is only to distinguish different NN models, i.e., FNN, CNN and RNN. To detect different architectures of NNs, the NN module follows the idea of transferability based AE attacks [44]. Intuitively, as mentioned in Section 2, AEs usually enjoys higher transferability as the architecture of substitute model does not deviate a lot from the victim model, i.e., Observation 4.

Observation 4 Given a fixed attack sample set, the transfer rate of some AE generation algorithms have a correlation with local substitute model. As the architecture of the substitute model is similar to the remote model, the AEs usually achieve a higher attack success rate, and vice versa.

Our experiments in Section 5 also validate the above observation. Based on the observation, we propose a transferability test based NN module to detection NN types. In the test, three sets of models F_{FNN}, F_{CNN}, F_{RNN} with different network architectures are first selected to serve as the local substitute models. Then, the adversary generates multiple AEs on these models, with the same AE generation algorithm. Given a target model f and a local model f', suppose that there are N_o AEs that attack f' successfully. Among the N_o AEs, N_t of them also attack f successfully. Then, the transfer rate tr is computed as follows.

$$tr = \frac{N_t}{N_o} \times 100\%. \tag{9}$$

For each set of models, the one with the highest transferability rate is assumed to have the same type of architecture as the target model. Recall that the goal of the NN module is to achieve model type detection of NNs. Therefore, SNOOPER cares nothing about the attack ability of AEs against local models (or the target model) but only their sensitivity to model types. Hence, state-of-the-art AE generation algorithms are not always the best choice to implement NN module. For example, in our evaluation (detailed in Section 5), some classic algorithms, e.g., Fast Gradient Sign Method [45] (FGSM), achieve a better performance to detect NN model type. Finally, represent the detection confidence of the NN module for FNN, CNN and RNN to be p_{FNN}, p_{CNN} and p_{RNN}. The NN module achieves $\Pr[\text{EXP}_{\{NN, \mathscr{M}_0/NN\}}(f, \mathscr{A}, \mathscr{C}, q) = 1] = (p_{FNN} + p_{CNN} + p_{RNN})/3$.

4.3 Attack Sample Generator

With the modules presented above, it can be concluded that Generator only has to collect two types of attack samples.

1. Generator randomly select several features and some data points within the input space. For the selected features, increase (or decrease) its original feature values at a predefined interval to provide attack samples for the DT module. The modified feature values should contain the boundary values for the KM module. Meanwhile, all these samples can be used in the equation-solving test in the LM module.
2. Besides the random data points, Generator also needs to generate some random data points according to the AE generation rules. These data can be repeatedly used to search AEs for different types of networks in the NN module.

Above all, the former three detection modules for DT, LM and KM can share the same attack sample pool. While, the NN module needs to generate attack samples independently. Moreover, all samples queried in the detection stage can also be utilized to launch other MLaaS attacks. Therefore, in the real-world black-box scenarios, the overall query complexity of SNOOPER is much lower than trying every MLaaS attack method and find the effective one by guessing.

5 Implementation & Experiments

5.1 Environment Preparation

Attack Scenarios. In the experiments, we mainly evaluate SNOOPER in the following two scenarios.

- Local Simulation. Build a local server that can provide model prediction service for SNOOPER.
- Real-World Platforms. Use SNOOPER to attack real-world MLaaS platforms, including Google Cloud[1], Amazon Web Services[2], and Huawei Cloud[3].

In both of the two scenarios, SNOOPER can only query the MLaaS interface in the black-box format like a normal user. That is, SNOOPER needs to provide valid inputs to the MLaaS interface, and can only obtain the predictions of these inputs. Except the queries and responses, any other info, e.g., decision paths and model type, is untouchable for SNOOPER. This setting is different from all previous researches. In our experiments, local simulation is used to do ablation study on SNOOPER for choosing optimal attack parameters and testing its robustness.

Datasets. We comprehensively evaluate SNOOPER on seven commonly used public datasets, including Iris, Adults, Breast Cancer, Digits, MNIST, Fashion MNIST, and CIFAR-10. The first four are tabular classification datasets, and the rest is image classification datasets. The detailed dataset info is shown in Table 2. These datasets cover the settings of most MLaaS attack researches [4, 5, 9, 10]. For each dataset, we randomly choose 80% of data to train model and 20% to validate model performance.

[1]https://cloud.google.com/ai
[2]https://aws.amazon.com/
[3]https://www.huaweicloud.com

Models. Totally, we select 975 different models to test the performance of SNOOPER. For every dataset, there are different models trained for each type of model, all of which are trained with different hyperparameters. The model info is listed in Table 3. To ensure fair evaluation, the models prepared for testing SNOOPER are varied from parameter sizes, degrees of overfitting, optimization strategies, and etc. The detailed training conditions are given below.

- **Linear Models.** There are totally 20 LRs and 20 SVMs with linear kernel trained for each dataset. To obtain different LRs, we change: the inverse of regularization strength, the norm of the penalty, the optimizer, and the iteration number. To obtain different SVMs, we change: the regularization coefficient, the Kernel coefficient, and the iteration number.
- **Decision Trees.** There are totally 10 ID3 models, 10 Random Forests, 10 XGBoosts, and 10 AdaBoosts trained for each dataset. To obtain different decision trees, we change: the split criterion, the split strategy and the maximum tree depth. To obtain different forest models, we change: the maximum number of trees, the maximum tree depth, and the split criterion.
- **Kernel Models.** There are totally 20 SVMs with RBF kernel and 20 SVMs with Logistic kernel trained for each dataset. RBF kernel and Logistic kernel are both non-linear kernels. To obtain different SVMs, we change: the regularization coefficient, the Kernel coefficient, and the iteration number.
- **Neural Networks.** There are totally 15 CNN, 15 FNN, and 15 RNN trained for MNIST, Fashion MNIST and CIFAR-10, respectively. To obtain different NN models, we change: the number of hidden layers, the number of neurons in each layer, and the number of fully connected layers.

Table 2: The detailed information of Datasets.

Name	Classes	Size	Features
Iris	3	178	4
Adults	2	49k	14
Breast Cancer	2	659	30
Digits	10	1.8k	8*8
MNIST	10	60k	28*28
Fashion MNIST	10	60k	28*28
CIFAR-10	10	60k	3*32*32

Table 3: Models used for evaluation.

Name	Model Type	Number
Iris	LMs, DTs, & KMs	40 * 3
Adults	LMs, DTs, & KMs	40 * 3
Breast Cancer	LMs, DTs, & KMs	40 * 3
Digits	LMs, DTs, & KMs	40 * 3
MNIST	LMs, DTs, KMs & NNs	40 * 3 + 30
Fashion MNIST	LMs, DTs, KMs & NNs	40 * 3 + 30
CIFAR-10	LMs, DTs, KMs & NNs	40 * 3 + 30

Especially, we do not train NNs for the former four datasets because NN is rarely used on these datasets due to serious overfitting problems.

Evaluation Indicator. There are mainly two indicators concerned by the evaluation of SNOOPER, namely detection accuracy and the number of queries (Query). The detection accuracy show whether SNOOPER can correctly judge the type of a black-box model. While, Query is a common indicate to evaluate MLaaS attacks, which indicates the cost to successfully launch an attack. To evaluate these two indicators, we first shuffle the previously trained models randomly and deploy on the victim interface. Then, the detection results and query numbers are recorded to compute these two indicators. Here, detection accuracy is computed as follows.

$$detection\ accuracy = \frac{n_{true}}{n_{total}} \times 100\%, \tag{10}$$

where n_{true} denotes the number of models whose types are detected correctly by SNOOPER, and n_{total} is the total number of models involved in the evaluation.

5.2 Overall Evaluation on SNOOPER

We summarize the overall evaluation result of SNOOPER in Table 4. The evaluate covers seven datasets and all 945 models. In the experiments, we first shuffle the order of all models randomly, and then, deploy these models on the three real-world MLaaS platforms. SNOOPER is utilized to conduct black-box detection of the types of all deployed models. The detection accuracy of each type of models is computed by measuring the rate of this type of models detected correctly by SNOOPER. All reported indicators in Table 4 are the average values on three platforms. Note that the indicators listed in Table 4 about NN module denote its detection accuracy to identify different types of NNs (as discused in Section 4).

From the results, the first observation is that SNOOPER is an effective attack against MLaaS platforms. For each dataset, SNOOPER achieves almost more than 90% detection accuracy on every type of models, and 92.14% average detection accuracy on all models. Moreover, without NN, SNOOPER only consumes tens of queries (positively related to the feature number) to complete its attack. While, even NN is involved, only thousands of queries are required for SNOOPER to launch attacks. Compared to other MLaaS attacks that always need tens of thousands of queries against black-box NNs [4, 9, 29, 36], such a cost is totally acceptable, as the pre-step attack.

Also, to validate the performance SNOOPER, we test it on three real-world platforms, respectively. Here, an interesting finding is that in the documents of Huawei Cloud, it is claimed that the cloud can monitor and block malicious queries to avoid attacks[4]. However, our evaluation shows that with our attack, the model type privacy is still broken successfully. This is because in current clouds, the defense mechanism against query-response analysis attacks, like SNOOPER, is only based on an access control monitor. As well as we control the query pattern and pay a little more running time, this defense can be easily circumvented.

Table 4: Overall performance of SNOOPER (Average over three platforms).

Dataset	Model Types	Queries	Detection Accuracy	Total Average	
				Queries	Detection Accuracy
Iris	DT	20.01	100.00%	10.11	95.83%
	LM	2	90.00%		
	KM	8.31	97.50%		
Adults	DT	20.12	97.50%	10.32	90.00%
	LM	3	80.00%		
	KM	7.83	92.50%		
Breast Cancer	DT	20.39	92.50%	10.46	89.17%
	LM	3	77.50%		
	KM	7.98	97.50%		
Digits	DT	20.00	100.00%	30.82	90.83%
	LM	65	72.50%		
	KM	7.45	100.00%		
MNIST	DT	20.27	97.50%	1790.52	95.76%
	LM	785	92.50%		
	KM	13.95	97.50%		
	NN	6342.85	95.56%		
Fashion MNIST	DT	20.89	97.50%	1774.79	90.83%
	LM	785	82.50%		
	KM	9.71	90.00%		
	NN	6283.56	93.33%		
CIFAR-10	DT	20.35	97.50%	3971.39	92.57%
	LM	2355	80.00%		
	KM	10.43	95.00%		
	NN	6342.43	97.78%		

5.3 Module Evaluation on SNOOPER

Now, we separately conduct evaluation on each detection module to give a closer look at SNOOPER. All following experiments mainly answer one essential question: is our design for each module practically useful to break model type privacy? Note that for easy analysis, the following experiments in Table 5 are all conducted with local simulation, utilizing the same model set as above.

Table 5: Performance on Google Cloud, Amazon Services,and Huawei Cloud (Average on MNIST).

Platform	Query	Detection Accuracy
Google Cloud	1731.41	93.07%
Amazon Web Services	1723.54	91.23%
Huawei Cloud	1767.55	92.69%

5.3.1 Module for Decision Tree

We carry out experiments to validate that DT module can be used to judge whether a given black-box model is a DT model. For this purpose, we first investigate the decision curves of different models. Four models are selected randomly, one for each type. Then, from the MNIST dataset, we arbitrarily choose an image, and perturb one of its pixel x_0 from x_0 to $x_0 + \delta \cdot \beta n$ where δ is the perturbation ratio, β is maximum

[4]https://res-static.hc-cdn.cn/cloudbu-site/intl/en-us/CAF/liaoyufei/CloudAdoptionFrameworkTechnicalWhitePaperV1.0.pdf

value of the chosen feature and n denotes the perturbation times. Next, the change curve between perturbed inputs and their corresponding predictions is plotted. In Figure 3, we show the experimental result with $\delta = 0.05$ and $n \in [1, 100]$. From the results, it can be observed that except the DT model, the predictions of all other types of models hold a tight correlation with inputs. Alternatively, as the input is changed, the predictions of other models are also changed with a very high probability. While, the DT model dose not hold the property due to its discreteness. The phenomenon validates that discreteness test is a reasonable way to determine whether a model belongs to DT.

Furthermore, the above experiment also points out a fact that the effectiveness of DT module is highly related to the choice δ. If δ is too large, it is possible to make the discreteness test cross the decision boundary in one step, and lead to a low detection accuracy. In Figure 4a and Figure 4b, we show the performance change of DT module with different δ. The result demonstrates the negative relation of the detection accuracy of DT module with δ. Also, the experiment shows that this parameter is not hard to tune in applications. Normally, with $\delta = 0.01$ or 0.05, our DT module is able to achieve satisfying performance.

5.3.2 Module for Kernel Models

Next, we conduct experiments to explore the effectiveness of KM module. Recall that KM module is implemented based on the symmetry test (Observation 3). Hence, to know whether KM module is practical, we need to investigate the decision patterns of different models over the whole input space. Remark that DT module concentrates on the microscopic characteristic of models, i.e., the prediction patterns over a very limited input range. Relatively, to understand KM module, a higher angle of observation view is required. Thus, we do an experiment as described below.

Same as before, we arbitrarily choose an image and perturb its pixels. The difference is that the pixel is not perturbed with its original value as the start point but directly replaced by values changed from the lower bound to the upper bound. Figure 5 plots the experimental result. Due to space limits, only the experimental result with MNIST, the most commonly used testing dataset, is presented. While the models trained with other datasets obey a similar rule according to our observation. From the result, we can find that among all types of models, only the decision curve of KM has prominent symmetry. Therefore, it concludes that symmetry test is an effective way to implement KM module.

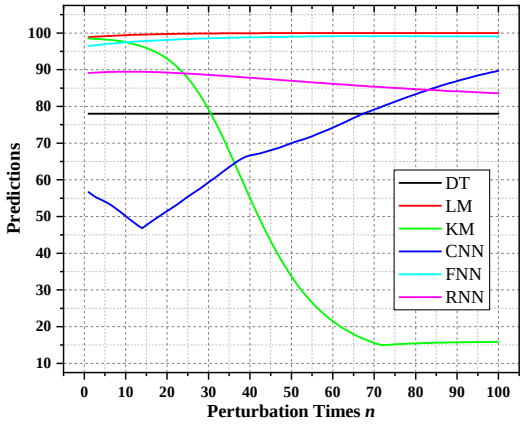
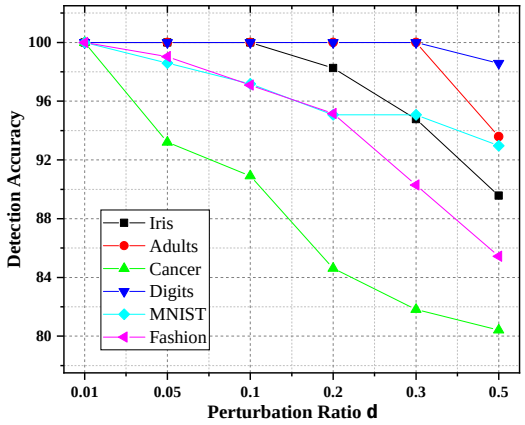
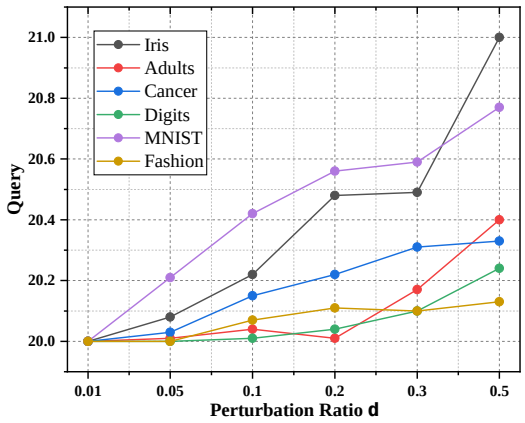


Figure 3: The prediction score distributions for different types of models with different perturbation times n.

Moreover, we test the performance of the SVM module as setting different error bounds ε (see Equation 8). The experimental results is depicted in Figure 6a and Figure 6b. It can be observed that with a lower error bound, SVM module tends to achieve better performance. This is because a lower error bound makes SVM module more sensitive to the difference between different types of models, causing less cost on detecting the model type.



(a) Detection Accuracy



(b) Query

Figure 4: The detection accuracy and queries of DT module with different δ.

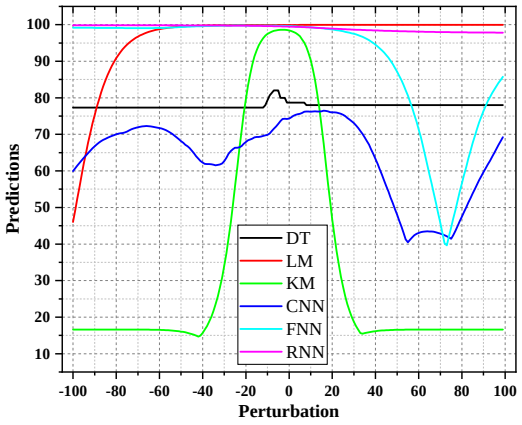


Figure 5: The prediction score distributions for different types of models with inputs changed from the lower bound to the upper bound.

5.3.3 Module for Linear Models

For LM module, we record the performance change of equation-solving test with increased γ, where γ denotes the ratio of query-response pairs used for equation solving. As $\gamma = 1.0$, it means that the number of query-response pairs is identical to the feature number. Clearly, if our design logic is rational, the performance of LM module should descend with $\gamma > 1.0$. This is because as more query-response pairs are provided, it is easier for other complicated models, i.e., NNs, to also enjoy high extraction performances with equation solving, like LMs. The experimental results shown in Figure 7 support our viewpoint. As γ is less than 1.0, LM module achieves similar detection accuracy. With the continuous increase of γ, the performance of LM is gradually degraded.

5.3.4 Module for Neural Networks

Different from other modules, NN module also provides the functionality of detecting The NN module is mainly implemented by utilizing the transferability of AEs. To show its rationality, we experiment with the transfer rates of substitute model based AE attacks against different types of models. Specifically, we first prepare nine NNs with different architectures, including 3 CNNs, 3 RNNs, and 3 FNNs. Then, each model is chosen as the local substitute model to generate AEs against every other model. The heatmaps shown in Figure 8 present the Attack Success Rates (ASRs) under each condition. It is clear that as the substitute model shares the same type of architecture with the victim model, the generated AEs enjoy higher attack success rates. This phenomenon explains why we can achieve NN model detection via AE transferability. Moreover, during the experiments, we notice that existing works about transferability based AE attacks mostly tend to find a way to break the dependency of AEs on model architectures. While, for MTD-A, such an improvement dose not contribute to the raise of its attack performance. State-of-the-art AE attack makes it hard to identify the ASR difference with varied models. Thus, in SNOOPER, we choose an early scheme, FGSM, to generate AEs.

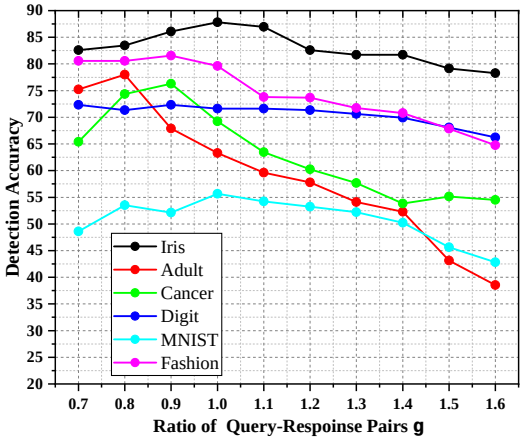
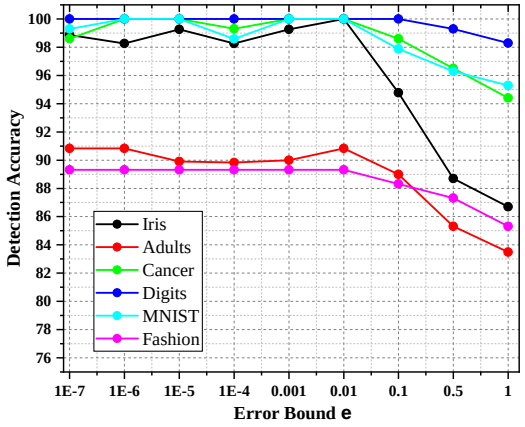
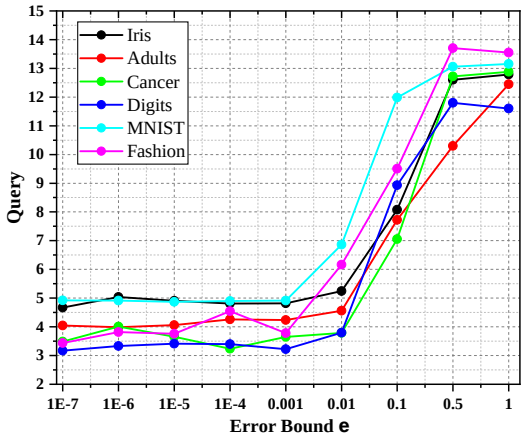


Figure 7: The performance change of LM module with different γ.

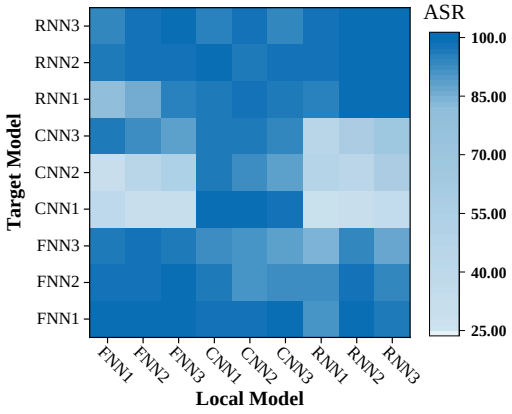


(a) Detection Accuracy

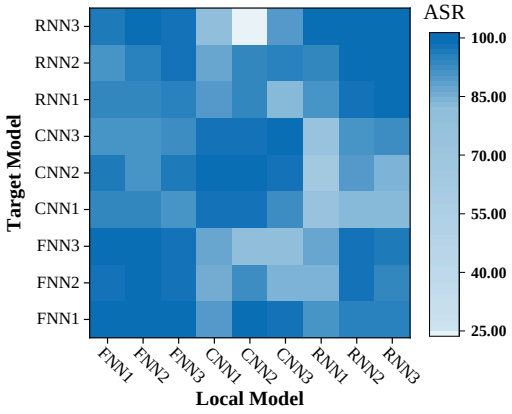


(b) Query

Figure 6: The detection accuracy and queris of SVM module with different ε.



(a) MNIST



(b) Fashion-MNIST

Figure 8: The transferring rates of AEs against different architectures of NN models.

5.4 Why We Need SNOOPER

To validate the threat of model type privacy protection to MLaaS platforms, we further conduct experiments to test the performance of different attacks with and without model type info. In more detail, we implement three different MLaaS attacks, including [4], [21] and [46]. The results are shown in Table 6.

In the experiments, all attacks are executed in two conditions. First, we run these attacks with SNOOPER. Alternatively, the attacker can know the type of the target model with SNOOPER to choose a proper attack method. At this time, the query number is recorded by summing the query costs of both SNOOPER and the launched MLaaS attack. While, under the other condition, the MLaaS attack is directly launched with an unknown target model type. Then, we record the averaged queries and attack success rates (ASR) after trying each possible attack method. From the experimental results of Table 6, it is observed that the application of SNOOPER can decrease more than half of the cost to launch these MLaaS attacks. Also, the average ASR is increased significantly. The phenomenon motivates us pay more attention on the research of model type privacy.

5.5 How to Defend Against Model Type Privacy Leakage

Finally, we propose two potential ways to defend against model type privacy leakage caused by SNOOPER, namely Truncation-based defense and Noise Perturbation-based defense. Both of the two way leverage the fact that SNOOPER is an attack art developed based on the elegance of the subtleties. Thus, if the MLaaS platforms can hide subtle information about model predictions, the attack can be block effectively. Our two defense are detailed below.

- Truncation. From our design, it can be noticed that most information about model types is derived from the decimal part of model predictions. Alternatively, current real-world MLaaS platforms can give predictions in more than ten-place decimal precision. In essence, SNOOPER utilizes this addition info, which is useless to common users, to launch attacks. Therefore, truncating the extra information of MLaaS predictions can be an effective way to defend against SNOOPER.
- Noise Perturbation. Another potential defense way is to perturb the predictions with noises and block SNOOPER to get useful information to infer model types. Such a defense strategy is also widely used in many defenses against other attacks [47, 48].

Table 6: Model Stealing with and without model type info.

Attack	with Known Model Type		with Unknown Model Type	
	ASR	Query	ASR	Query
[4]	99.99%	3925	23.72%	20933
[21]	96.98%	24384	61.08%	72823
[46]	95.27%	10205	87.68%	21456

Table 7 demonstrates the defense effects of our two defenses. In the experiments, the first defense is implemented by limiting the model predictions to be in the second decimal places. As for the second defense, the noise perturbation-based method, our implementation is slightly different from previous noise-based defenses [4, 49]. Specifically, to better preserve the performance of models, we choose a weighted noise addition strategy. According to the strategy, the prediction scores are treated as perturbation weights. Higher weights mean higher perturbations (larger random noises) will be added into the prediction scores. The random perturbations are obtained via Laplacian noises. This strategy has a noteworthy advantage that the added perturbation does not change the predicted class but can make the predictions not to obey its original rule. Such an advantage makes it able to achieve a satisfactory defense effect while maintaining model accuracy. Moreover, we record two metrics after applying our defenses, which are accuracy decrease of the target model and the detection accuracy decrease of SNOOPER. Our experimental results show that both two defense can effectively defend against SNOOPER. While, truncation is easier to implement in practice but has worse defense effect than the noise perturbation based method.

From the experimental results, we can find that the above two defenses are both effective against SNOOPER. The difference is that the truncation-based defense leads less negative effect on the target model. While, the noise perturbation-based method achieves better defense effect but causes higher performance loss to the target model. Choosing which defense depends on the application requirement. As such, our experiments validate that the model type privacy can be preserved via some straightforward but effective methods. In applications, adding these defenses is effortless but can significantly improve the security of our MLaaS platforms.

Table 7: The performance of SNOOPER with defenses.

Dataset	with Truncation		with Noise Perturbation	
	Acc.↓	Detection Acc.↓	Acc.↓	Detection Acc.↓
Iris	0.6%	64.7%	2.2%	91.3%
Adults	0.5%	64.6%	2.7%	89.3%
Digits	0.6%	64.8%	3.4%	88.9%
MNIST	0.6%	65.2%	4.3%	93.6%
Breast Cancer	0.4%	33.0%	3.2%	91.2%
Wine	0.4%	32.6%	4.7%	90.4%

Acc. means the accuracy decrease of the target model. Detection Acc. means the detection accuracy decrease of SNOOPER.

6 Conclusion

In this paper, we analyzed the previous attacks against MLaaS platforms, and revealed a fact that their success was greatly based on an assumption that the victim model's type is known by the attacker. Motivated by this fact, we proposed a novel attack that could detect the model type via the black-box MLaaS interface, which we called it as model-type-detection attack SNOOPER. Several modules were developed

in SNOOPER, which could be used detect the types of six mainstream models discussed in previous attacks. Furthermore, we paid a great deal of efforts to train hundreds of models on 8 datasets and conduct comprehensive experiments on them. The experimental results validated that our attack could achieve satisfactory attack effect. Finally, we also gave our recommendation to defend against the proposed attack. Extensions to emerging architectures (e.g., Transformers and ViTs) and adaptive attack scenarios will be explored in future work.

Funding

This work was supported by the National Natural Science Foundation of China (62501444, 62261160651, U23A20306, U23A20307), the Postdoctoral Fellowship Program of CPSF under Grant Number GZB20250406, the Postdoctoral Research Project of Shanxi Province (2024BSHEDZZ015).

Author Contributions

Conceptualization, Yilong Yang, Xinjing Liu and Yang Liu; methodology, Yilong Yang; software, Xinjing Liu and Yang Liu; validation, Ruidong Han; investigation, Yilong Yang ; resources, Ruidong Han; data curation, Xinjing Liu; writing—original draft preparation, Yilong Yang and Yang Liu; writing—review and editing, Xinjing Liu; visualization, Yang Liu; supervision, Ruidong Han; funding acquisition, Yang Liu. All authors have read and agreed to the published version of the manuscript.

Conflict of Interest

All the authors declare that they have no conflict of interest.

References

[1] Xie, S., Xue, Y., Zhu, Y., Wang, Z.: Skyml: A mlaas federation design for multicloud-based multimedia analytics. IEEE Transactions on Multimedia **27**, 2463–2476 (2025) https://doi.org/10.1109/TMM.2024.3521768

[2] Lin, Y., Zhang, T., Mao, Y., Zhong, S.: Crossnet: A low-latency mlaas framework for privacy-preserving neural network inference on resource-limited devices. IEEE Transactions on Dependable and Secure Computing **22**(2), 1265–1280 (2025) https://doi.org/10.1109/TDSC.2024.3431590

[3] Wang, X., Liu, B., Bi, X., Xiao, b.: Seam-carving localization in digital images. Journal of Intelligent Computing and Networking **1**(1), 28–42 (2025) https://doi.org/10.64509/jicn.11.17

[4] Tramèr, F., Zhang, F., Juels, A., Reiter, M.K., Ristenpart, T.: Stealing machine learning models via prediction apis. In Proceedings of the 25th USENIX Conference on Security Symposium, pp. 601–618 (2016)

[5] Chandrasekaran, V., Chaudhuri, K., Giacomelli, I., Jha, S., Yan, S.: Exploring connections between active learning and model extraction. In Proceedings of the 29th USENIX Conference on Security Symposium, pp. 1309–1326 (2020)

[6] Shen, Y., Zhuang, Z., Yuan, K., Nicolae, M.-I., Navab, N., Padoy, N., Fritz, M.: Medical multimodal model stealing attacks via adversarial domain alignment. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 6842–6850 (2025). https://doi.org/10.1609/aaai.v39i7.32734

[7] Eisenhofer, T., Schönherr, L., Frank, J., Speckemeier, L., Kolossa, D., Holz, T.: Dompteur: Taming audio adversarial examples. In 30th USENIX Security Symposium (USENIX Security 21), pp. 2309–2326 (2021)

[8] Zhang, C., Zhou, L., Xu, X., Wu, J., Liu, Z.: Adversarial attacks of vision tasks in the past 10 years: A survey. ACM Computing Surveys **58**(2), 1–37 (2025) https://doi.org/10.1145/3743126

[9] Chen, K., Guo, S., Zhang, T., Xie, X., Liu, Y.: Stealing deep reinforcement learning models for fun and profit. In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, pp. 307–319 (2021). https://doi.org/10.1145/3433210.3453090

[10] Li, Z., Zhang, Y.: Membership leakage in label-only exposures. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 880–895 (2021). https://doi.org/10.1145/3460120.3484575

[11] Lowd, D., Meek, C.: Adversarial learning. In Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 641–647 (2005)

[12] Xie, Y., Gu, Z., Fu, X., Wang, L., Han, W., Wang, Y.: Misleading sentiment analysis: Generating adversarial texts by the ensemble word addition algorithm. In 2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), pp. 590–596 (2020)

[13] Fredrikson, M., Lantz, E., Jha, S., Lin, S., Page, D., Ristenpart, T.: Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In 23rd USENIX Security Symposium (USENIX Security 14), pp. 17–32 (2014)

[14] Chen, H., Zhang, H., Boning, D., Hsieh, C.-J.: Robust decision trees against adversarial examples. In International Conference on Machine Learning, pp. 1122–1131 (2019)

[15] Cheng, M., Le, T., Chen, P.-Y., Zhang, H., Yi, J., Hsieh,

C.-J.: Query-efficient hard-label black-box attack: An optimization-based approach. In International Conference on Learning Representation (ICLR), pp. 1–14 (2019)

[16] Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322–1333 (2015). https://doi.org/10.1145/2810103.2813677

[17] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 506–519 (2017). https://doi.org/10.1145/3052973.3053009

[18] Zhang, Y., Jia, R., Pei, H., Wang, W., Li, B., Song, D.: The secret revealer: Generative model-inversion attacks against deep neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 253–261 (2020). https://doi.org/10.1109/CVPR42600.2020.00033

[19] Wang, K.-C., Fu, Y., Li, K., Khisti, A., Zemel, R., Makhzani, A.: Variational model inversion attacks. In Proceedings of the 35th International Conference on Neural Information Processing Systems, pp. 9706–9719 (2021)

[20] Esser, P., Fleissner, M., Ghoshdastidar, D.: Non-parametric representation learning with kernels. In Proceedings of the AAAI Conference on Artificial Intelligence, pp. 11910–11918 (2024). https://doi.org/10.1609/aaai.v38i11.29077

[21] Miura, T., Shibahara, T., Yanai, N.: Megex: Data-free model extraction attack against gradient-based explainable ai. In Proceedings of the 2nd ACM Workshop on Secure and Trustworthy Deep Learning Systems, pp. 56–66 (2024). https://doi.org/10.1145/3665451.3665533

[22] Andriushchenko, M., Croce, F., Flammarion, N., Hein, M.: Square attack: a query-efficient black-box adversarial attack via random search. In European Conference on Computer Vision, pp. 484–501 (2020). Springer

[23] Wu, W., Su, Y., Chen, X., Zhao, S., King, I., Lyu, M.R., Tai, Y.-W.: Boosting the transferability of adversarial samples via attention. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1161–1170 (2020). https://doi.org/10.1109/CVPR42600.2020.00124

[24] Suya, F., Chi, J., Evans, D., Tian, Y.: Hybrid batch attacks: Finding black-box adversarial examples with limited queries. In 29th USENIX Security Symposium (USENIX Security 20), pp. 1327–1344 (2020)

[25] Zhou, M., Wu, J., Liu, Y., Liu, S., Zhu, C.: Dast: Data-free substitute training for adversarial attacks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 234–243 (2020). https://doi.org/10.1109/CVPR42600.2020.00031

[26] Takemura, T., Yanai, N., Fujiwara, T.: Model extraction attacks on recurrent neural networks. Journal of Information Processing **28**, 1010–1024 (2020)

[27] Correia-Silva, J.R., Berriel, R.F., Badue, C., Souza, A.F., Oliveira-Santos, T.: Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2018). https://doi.org/10.1109/IJCNN.2018.8489592

[28] Cong, K., Das, D., Park, J., Pereira, H.V.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 563–577 (2022). https://doi.org/10.1145/3548606.3560702

[29] Rakin, A.S., Chowdhuryy, M.H.I., Yao, F., Fan, D.: Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In 2022 IEEE Symposium on Security and Privacy (SP), pp. 1157–1174 (2022). IEEE

[30] Yu, H., Yang, K., Zhang, T., Tsai, Y.-Y., Ho, T.-Y., Jin, Y.: Cloudleak: Large-scale deep learning models stealing through adversarial examples. In Network and Distributed System Security (NDSS) Symposium, pp. 1–16 (2020). https://doi.org/10.14722/ndss.2020.24178

[31] Wu, T., Luo, T., Wunsch, D.C.: Gnp attack: Transferable adversarial examples via gradient norm penalty. In 2023 IEEE International Conference on Image Processing (ICIP), pp. 3110–3114 (2023). https://doi.org/10.1109/ICIP49359.2023.10223158

[32] Gangurde, R.A.: Web page prediction using adaptive deer hunting with chicken swarm optimization based neural network model. International Journal of Modeling, Simulation, and Scientific Computing **13**(06), 2250064 (2022) https://doi.org/10.1142/S1793962322500647

[33] Akhavan, A., Chzhen, E., Pontil, M., Tsybakov, A.: A gradient estimator via l1-randomization for online zero-order optimization with two point feedback. Advances in Neural Information Processing Systems **35**, 7685–7696 (2022)

[34] Byun, J., Cho, S., Kwon, M.-J., Kim, H.-S., Kim, C.: Improving the transferability of targeted adversarial examples through object-based diverse input. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 15244–15253 (2022). https://doi.org/10.1109/CVPR52688.2022.01481

[35] Lu, F., Zhu, K., Zhai, W., Cao, Y., Zha, Z.-J.: Likelihood-aware semantic alignment for full-spectrum out-of-distribution detection. Journal of Intelligent Computing and Networking **1**(1), 1–13 (2025) https://doi.org/10.64509/jicn.11.10

[36] Tao, J., Shokri, R.: Range membership inference attacks. In 2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML), pp. 346–361 (2025). https://doi.org/10.1109/SaTML64287.2025.00026

[37] Hou, S., Li, S., Jahani-Nezhad, T., Caire, G.: Priroagg: Achieving robust model aggregation with minimum privacy leakage for federated learning. IEEE Transactions on Information Forensics and Security **20**, 5690–5704 (2025) https://doi.org/10.1109/TIFS.2025.3577498

[38] Chaudhari, S., Aggarwal, P., Murahari, V., Rajpurohit, T., Kalyan, A., Narasimhan, K., Deshpande, A., Silva, B.: Rlhf deciphered: A critical analysis of reinforcement learning from human feedback for llms. ACM Computing Surveys **58**(2) (2025) https://doi.org/10.1145/3743127

[39] Rivera, A., Uribe, J.: Graph based machine learning for anomaly detection in iot security. Electronics, Communications, and Computing Summit **3**(2), 40–48 (2025)

[40] Adibi, M.A.: Single and multiple outputs decision tree classification using bi-level discrete-continues genetic algorithm. Pattern Recognition Letters **128**, 190–196 (2019) https://doi.org/10.1016/j.patrec.2019.09.001

[41] Mugunthan, S., Vijayakumar, T.: Design of improved version of sigmoidal function with biases for classification task in elm domain. Journal of Soft Computing Paradigm (JSCP) **3**(02), 70–82 (2021) https://doi.org/10.36548/jscp.2021.2.002

[42] Chang, D., Sun, S., Zhang, C.: An accelerated linearly convergent stochastic l-bfgs algorithm. IEEE transactions on neural networks and learning systems **30**(11), 3338–3346 (2019) https://doi.org/10.1109/TNNLS.2019.2891088

[43] Wan, J., Wang, Q., Chan, A.B.: Kernel-based density map generation for dense object counting. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(3), 1357–1370 (2020) https://doi.org/10.1109/TPAMI.2020.3022878

[44] Zhong, F., Cheng, X., Yu, D., Gong, B., Song, S., Yu, J.: Malfox: Camouflaged adversarial malware example generation based on conv-gans against black-box detectors. IEEE Transactions on Computers **73**(4), 980–993 (2023) https://doi.org/10.1109/TC.2023.3236901

[45] Nowroozi, E., Mekdad, Y., Berenjestanaki, M.H., Conti, M., El Fergougui, A.: Demystifying the transferability of adversarial attacks in computer networks. IEEE Transactions on Network and Service Management **19**(3), 3387–3400 (2022) https://doi.org/10.1109/TNSM.2022.3164354

[46] Liu, Y., Wen, R., He, X., Salem, A., Zhang, Z., Backes, M., De Cristofaro, E., Fritz, M., Zhang, Y.: Ml-doctor: Holistic risk assessment of inference attacks against machine learning models. In 31st USENIX Security Symposium (USENIX Security 22), pp. 4525–4542 (2022)

[47] Hassan, M.U., Rehmani, M.H., Chen, J.: Differential privacy techniques for cyber physical systems: a survey. IEEE Communications Surveys & Tutorials **22**(1), 746–789 (2019) https://doi.org/10.1109/COMST.2019.2944748

[48] Lv, Z., Chen, D., Cao, B., Song, H., Lv, H.: Secure deep learning in defense in deep-learning-as-a-service computing systems in digital twins. IEEE Transactions on Computers **73**(3), 656–668 (2023) https://doi.org/10.1109/TC.2021.3077687

[49] Carlini, N., Chien, S., Nasr, M., Song, S., Terzis, A., Tramer, F.: Membership inference attacks from first principles. In 2022 IEEE Symposium on Security and Privacy (SP), pp. 1897–1914 (2022). https://doi.org/10.1109/SP46214.2022.9833649