

# Pareto-Secure Machine Learning (PSML): Fingerprinting and Securing Inference Serving Systems

Debopam Sanyal  
Georgia Institute of Technology

Jui-Tse Hung  
Georgia Institute of Technology

Manav Agrawal  
Georgia Institute of Technology

Prahlad Jasti  
Georgia Institute of Technology

Shahab Nikkhoo  
University of California, Riverside

Somesh Jha  
University of Wisconsin–Madison

Tianhao Wang  
University of Virginia

Sibin Mohan  
George Washington University

Alexey Tumanov  
Georgia Institute of Technology

**Abstract**—Model-serving systems have become increasingly popular, especially in real-time web applications. In such systems, users send queries to the server and specify the desired performance metrics (e.g., desired accuracy, latency). The server maintains a set of models (model zoo) in the back-end and serves the queries based on the specified metrics. This paper examines the security, specifically robustness against model extraction attacks, of such systems. Existing black-box attacks assume that a single model can be repeatedly selected for serving inference requests. Modern inference serving systems break this assumption. Thus, they cannot be directly applied to extract a victim model, as models are hidden behind a layer of abstraction exposed by the serving system. An attacker can no longer identify which model she is interacting with. To this end, we first propose a query-efficient fingerprinting algorithm to enable the attacker to trigger any desired model *consistently*. We show that by using our fingerprinting algorithm, model extraction can have fidelity and accuracy scores within 1% of the scores obtained when attacking a single, explicitly specified model, as well as up to 14.6% gain in accuracy and up to 7.7% gain in fidelity compared to the naive attack. Second, we counter the proposed attack with a noise-based defense mechanism that thwarts fingerprinting by adding noise to the specified performance metrics. The proposed defense strategy reduces the attack’s accuracy and fidelity by up to 9.8% and 4.8%, respectively (on medium-sized model extraction). Third, we show that the proposed defense induces a fundamental trade-off between the level of protection and system goodput, achieving configurable and significant victim model extraction protection while maintaining acceptable goodput (> 80%). We implement the proposed defense in a real system with plans to open source. Access to code provided for anonymous review<sup>1</sup>.

## 1. Introduction

The deployment of inference serving systems [17], [1], [16], [22], [50], [3] to serve machine learning models to users in interactive web applications [26], [61] is witnessing a significant surge, enabling applications to leverage efficient and scalable ML model predictions for a variety of use cases. As these applications are typically user-facing and interactive, ML inference must be performed in real-time, subject to strict latency constraints, known as a latency service objective (SLO) imposed on each individual request (e.g., < 100ms) [63]. The latency SLO defines the latency budget available to the system to serve a single query. Model-less inference serving systems [22], [50] abstract away the burden of explicit model selection from the set of registered models (known as the “model zoo”) by the client. These systems decouple applications from the models they use, allowing each to evolve independently. Furthermore, this obviates the need for “early binding”—making premature static model choice decisions (e.g., [17], [16]) and enables “late binding”—choosing the right-sized model *just-in-time* on a query to query basis (e.g., [22], [50]). This flexibility proves important for maximizing the fraction of queries for which the latency SLO is met (defined as latency SLO attainment), as dynamic deployment conditions (e.g., memory/network/storage bandwidth, power consumption, battery level) and application requirements (desired accuracy and response time) fluctuate [7]. This proliferation of model zoo inference serving systems raises significant concerns regarding the privacy and security of the registered models. The models stored span a large range of model sizes and accuracies, each potentially trained on proprietary datasets and further fine-tuned and specialized to different runtime conditions, exposing valuable intellectual property.

Each model in the model zoo is considered highly valuable because they are expensive to obtain [54]. Along with

1. Two links: PSML Repo and Modified Clockwork Repo

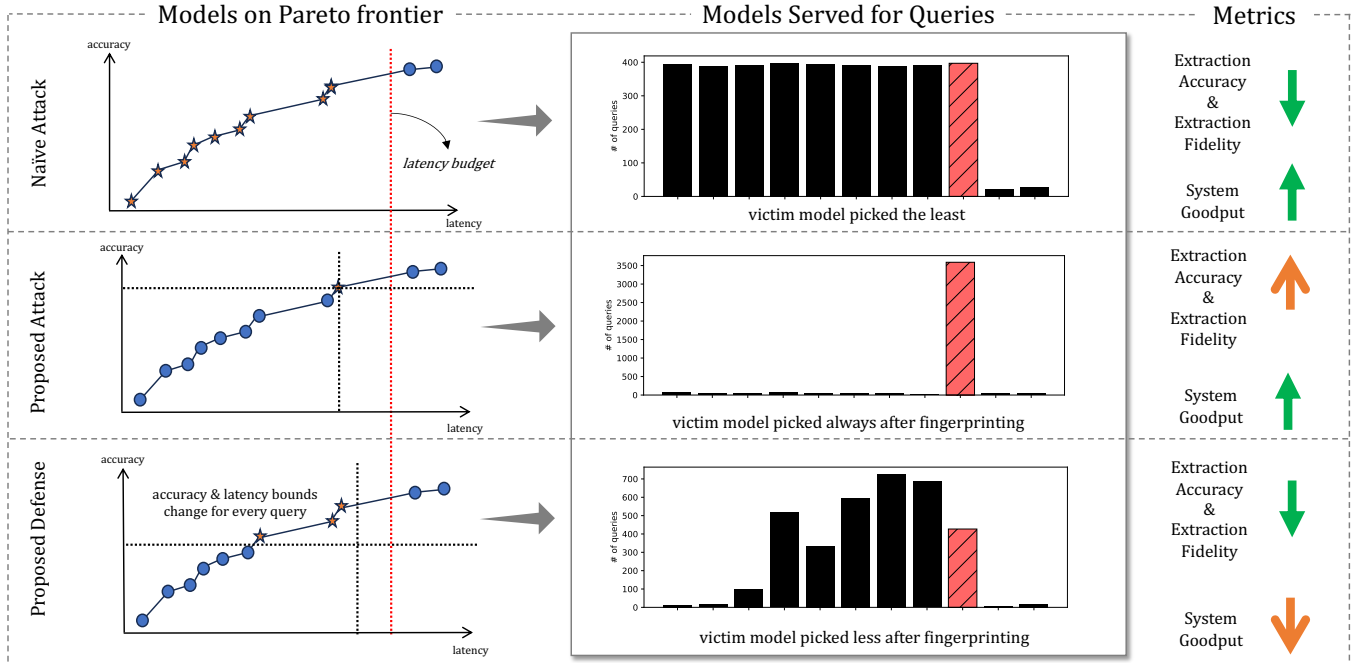


Figure 1: A high-level overview of this paper. The first Pareto frontier shows a naive attack. In the second Pareto frontier, the accuracy and latency specifications obtained from our fingerprinting algorithm successfully target the victim model, and it gets picked for serving most of the queries. In the third Pareto frontier, the accuracy and latency specifications after fingerprinting but with the defense become less accurate, and the victim model is not picked that often. Our fingerprinting algorithm shows that a single-model attack can be used on a model zoo. With our defense mechanism, we offer a protection method.

training the final model, the architecture, training algorithm, training data, and hyper-parameters are all valuable (e.g., Training GPT-3 [9] cost OpenAI more than \$4 million). Qualcomm is scheduled to make available Meta’s Llama 2 [56], which has up to 70 billion parameters, based AI implementation on smartphones in 2024 [4]. This implementation could be worth millions of dollars. The cost is further compounded by specializing the model to predictably execute with high probability under a certain latency budget leveraging latency-aware Neural Architecture Search, mixed-precision quantization, hardware-aware kernel fusion, etc. Thus, surprisingly, smaller models may actually hold higher value to the attacker. Model zoos can contain tens to hundreds of these vulnerable models in the cloud, each specialized for a different application requirement (e.g., Meta’s anomaly detection framework, Sigma, contained more than 100 distinct models running in production everyday in 2018 [26] and Meta’s personalized recommendation system contained more than 400 distinct models in 2020 [25]).

Black-box attacks [52], [21], [57], [6] pose a substantial threat to the privacy and security of inference serving systems. These attacks allow adversaries to exploit vulnerabilities within the system and extract sensitive information, including the victim model’s functionality, training data, architecture, and parameters. Surprisingly, despite the increasing adoption of model-less [22], [50] inference serving systems and the displacement of explicit model selection serving [17], [16], [29], [1], practical demonstrations of

black-box attacks on these systems have not been well studied. Indeed, state-of-the-art model extraction techniques assume the adversary’s ability to route all queries to the same, explicitly specified victim model. Importantly, we assert that this is an outdated and impractical assumption for state-of-the-art inference serving systems [50], which prioritize flexibility to cater to diverse user requirements. Consequently, a pressing need arises to explore effective mechanisms that can bridge this gap: adapt existing black-box attacks, such as model extraction, to state-of-the-art inference serving systems that are model-less, latency-aware, and operate under real-time latency constraints. To the best of our knowledge, this work is the first to showcase the viability of black-box attacks on inference serving systems *without requiring explicit model specification by the user*.

Since interactive applications have different latency SLOs for different requests, model zoo entries typically span a broad latency-accuracy tradeoff space. Inference serving systems typically expose and serve the **Pareto frontier** of optimality of this tradeoff space, such that the highest accuracy model is served for a given latency budget, and the lowest latency model is served for a given accuracy threshold. Since applications operate on diverse operating latency budgets, and target deployment devices may have diverse resource constraints, the highest accuracy model may not always be the attacker’s desired target. Given the latency constraint of attacker’s interest, she does, however, prefer the highest possible accuracy model that satisfies this

constraint. Indeed, extracting Pareto optimal points with *lower* accuracy/latency tradeoff may be more suitable for embedded devices or autonomous cyber-physical systems that operate on a tight sensory-actuator feedback control loop and are resilient to lower accuracy [30]. To target a given point of interest on the Pareto frontier requires a fingerprinting mechanism.

Thus, we propose a novel and query-efficient fingerprinting-based attack that enables model extraction performance on a model-less inference serving system (w.r.t. accuracy and fidelity) comparable to traditional black-box attacks conducted on a single, explicitly specified model. Our fingerprinting algorithm enables black-box extraction attacks to have accuracy and fidelity within 1% of the corresponding values in the single-model setting while spending the same number of queries. We show up to 14.6% gain in accuracy and up to 7.7% gain in fidelity than the naive attack without fingerprinting while spending only 4000 queries. This bridges the gap and recoups the loss in extraction performance thwarted by state-of-the-art model-less abstraction layer.

We further propose a novel defense approach to counter our fingerprinting attack. The defense mechanism is based on the introduction of noise to the performance<sup>2</sup> specifications of queries, effectively disrupting the fingerprinting process. Our defense strategy reduces the accuracy and fidelity of the attack by up to 9.8% and 4.8%, respectively, compared to the scores obtained with our fingerprinting algorithm. It is particularly successful in protecting medium and small-sized victim models. Finally, we expose and study a fundamental tradeoff between the effectiveness of our defense and inference serving system performance. We measure the latter in goodput—fraction of queries with both accuracy and latency constraints satisfied. We demonstrate that our defense can give significant protection while maintaining acceptable goodput (> 80%). Fig. 1 captures a high-level conceptual overview.

We instantiate the proposed attack and defense mechanisms in a real Pareto-Secure Machine Learning system (PSML), integrating it with a state-of-the-art inference serving system [22]. Our paper instantiates the following principal contributions:

- A real model-less inference serving system with the implementation of proposed attack and defense mechanisms, processing queries in real time in accordance with accuracy and latency constraints.
- A generic and query-efficient fingerprinting algorithm that enables practical black-box extraction attacks on model-less inference serving systems.
- A noise-based defense strategy that effectively reduces the success of fingerprinting-based attacks on inference serving systems.
- Configurable levels of defense, inducing a trade-off space between system goodput and level of protection, providing the ability to achieve robust

2. Proposed attack and defense generalize to metrics other than latency.

protection while maintaining reasonable system performance.

## 2. Background and Related Work

### 2.1. Inference-Serving Systems

TensorFlow Serving [1] was one of the first dedicated ML serving systems, although it was limited to models in the TensorFlow framework. Clipper [17] was later developed to use general frameworks and make it modular for anyone to deploy a model. Amazon Sagemaker [29] and NVIDIA Triton Inference Serving [3] were some of the first publicly released serving systems officially offering inference as a service to satisfy business use cases. These systems have the advantage of large infrastructures backed by Amazon and NVIDIA, as Sagemaker autoscales based on the inference load, and Triton optimizes inference on NVIDIA GPUs.

All of these systems require the user to specify the model used for inference, which may not satisfy all use cases. InferLine [16] provides serving for a pipeline of models by planning resource allocation for each model and tuning as necessary during execution. A prominent problem is a lack of developer understanding of the trade-off of accuracy/latency among variants of a model, such as the ResNet family [27]. Therefore, instead of having the developer specify the model to query, model-less systems that query from model zoos have arisen. INFaaS [50] generates variants for every model deployed to its zoo during the profiling process, and it navigates the trade-off space of these variants on the user’s behalf. The clients simply provide the latency and accuracy bounds with their inputs in INFaaS.

However, many of these systems do not take advantage of the predictability of inference latency for a model, *i.e.*, deterministic forward pass latency of a deep neural network. Only recently has this attribute been studied, by systems such as Clockwork [23] and iGniter [62]. Clockwork achieves predictability by discarding queries that take too long, as well as by reducing the choice of resource allocation and program execution at the hardware, OS, and application level in order to reduce variability in end-to-end latency. This determinism allows for a model zoo to be represented as a Pareto frontier when plotted by its latency and accuracy, exhibiting the positive correlation between these two attributes in ML models. However, both of these systems require clients to specify the models they want along with the input.

Secure inference serving systems have been proposed to protect against malicious clients [38], [53], [11]. These works employ secure cryptographic protocols to protect against attacks that try to steal private inputs and parameters of the model. We consider an adversary (§3.2) that is only interested in extracting the functionality of a model and not its weights or training data.

**Model Zoo.** Models that are pre-trained for a specific task like image recognition can be coalesced into a repository known as a model zoo. Real-world examples have demonstrated the wide range of model zoo scales, with some

containing as few as five models [17], while others contain hundreds [25], [26]. Clockwork [23] uses the ONNX [2] and GluonCV [24] model zoos to keep 61 different models in its model zoo. INFaaS [50] contains 22 different architectures of models in its model zoo, each having a number of associated model variants combining for a total of 175 model variants. Each model variant is trained with different frameworks, compilers, batch sizes, and hardware platforms.

Developer APIs are used to interact with the model zoo of a system. For instance, the INFaaS [50] model-less interface has a declarative API that uses `register_model` to allow users to register models to its model zoo. To submit inference queries, developers use the `inference_query` API to specify high-level application requirements, such as accuracy and latency goals, without specifying the models. INFaaS then navigates the model zoo, selecting suitable model variants to meet the specified goals.

## 2.2. Black-box Model Extraction Attack

Black-box attacks are attacks where the adversary lacks knowledge of the victim model’s parameters, architecture, or training data. Machine Learning as a Service (MLaaS) are systems on which such attacks are usually carried out. In every query, the user typically submits an input and receives either a prediction vector or a class label from an already trained model hosted in the cloud. Most of these attacks are carried out during inference and, thus, on inference serving systems. Such attacks aim to obtain information not meant to be disclosed, such as the training data or details about the model.

The adversary’s goal in model extraction is to replicate the functionality of the victim model by creating an extracted model [10], [49], [33], [14], [57], [42], [31], [12], [48], [58], [47]. It leverages the ability to query the victim model and observe its outputs, which are utilized to train the extracted model. Task accuracy attacks involve creating a model that matches the victim model’s accuracy on a test set derived from the input data distribution. Fidelity attacks, however, aim to maximize the similarity between the victim and extracted models on the test set. Fidelity can be defined as the ratio of points in the test set on which both the victim and the extracted models have the same output labels. Attackers with problem domain data require fewer queries, and access to output labels alone is adequate for them to extract a model. In both scenarios, the adversary aims for efficiency, striving to minimize the number of queries used. One notable extraction attack is the MixMatch-based [8] extraction attack [31]. The MixMatch attack uses unlabeled task-specific data to improve model extraction via semi-supervised learning techniques. Here, the victim model architecture and the output prediction vector are not provided to the attacker; it only gets the output label to its input query. More details about the attack are provided in §C.

These attacks assume that the outputs received by the adversary are all from the victim model; hence, we call them single-model attacks in this work. The attack described

in [38] is on inference serving systems, but it assumes that the adversary can access the victim model repeatedly from the model zoo.

## 3. Threat Model and Motivation

### 3.1. System Model

We consider an inference serving scenario where models are loaded in the system like in Clockwork [23], and it is the responsibility of the inference serving system to select a model for each query that satisfies the accuracy and latency specified by the query, like in INFaaS [50]. Predictability in inference serving systems is extremely important, and for that reason, state-of-the-art inference serving frameworks support multi-tenancy in a non-interfering manner by mapping their query traffic flow to different model-serving workers. In addition to exclusive access to GPU workers, applications are also typically mapped to different queues in the router. Clockwork maintains a single queue per model served. The combination of dedicated queues inside the serving system as well as dedicated GPU workers leads to a multi-tenant system with highly predictable tail latency guarantees, leveraged by P<sub>S</sub>ML.

**3.1.1. Pareto Frontier and Feasibility Set.** For our purposes, the model zoo will be used to provide possible model selections for Clockwork to serve when performing inference. If we plot each model in the model zoo as a point on a scatter plot, such that the  $x$ -axis is the model’s inference latency and the  $y$ -axis is its accuracy, we can select a subset of points that are preferred to the other points. This subset constitutes a **Pareto frontier**. It is a subset of points such that no point in the subset is strictly better than any other point when plotted against its chosen attributes. In our context, it means that no model will have both a lower latency and a higher accuracy compared to any other model on the Pareto frontier. By definition, every model zoo has a Pareto frontier.

**Definition 1.** (*Pareto frontier*) Let  $a_m$  and  $l_m$  be the accuracy and latency values of any model  $m$  in the model zoo. For any  $(a_i, l_i), (a_j, l_j) \in \mathbb{R}^2$ , the Pareto frontier  $\mathcal{P}$  is:

$$\mathcal{P} = \{(a_i, l_i) | \{(a_j, l_j) | (a_j > a_i) \wedge (l_j < l_i)\} = \phi\}, \quad (1)$$

where  $\phi$  is the null set,  $(a_i, l_i) \neq (a_j, l_j)$ , and  $i \in \{1, \dots, n\}$  with  $n$  being the total number of models in the model zoo. Thus,  $\mathcal{P} \subseteq \mathbb{R}^{n \times 2}$ . Since it is preferable to minimize inference latency and maximize accuracy, the Pareto frontier will form at the top left of the region of points representing the model zoo, as shown in Fig. 4. The Pareto frontier will serve as the backbone for our fingerprinting algorithm, as it provides the adversary a path of traversal across the model zoo.

The **feasibility set** of a query is the set of models that satisfy the latency and accuracy specifications of the query. It is a subset of the Pareto frontier of the model zoo. The manner in which the serving system makes a

model selection from the feasibility set, such as aggregation, random selection, round-robin, lowest-cost, *etc.*, provided that the set is non-empty, is known as the system’s policy. If the accuracy and latency specifications of a query are  $acc$  and  $lat$ , respectively, then the feasibility set is  $\mathcal{F} = \{(a_i, l_i) \in \mathcal{P} | (a_i \geq acc) \wedge (l_i \leq lat)\}$ .

**3.1.2. Inference Serving.** To demonstrate that single-model attacks against a victim model from a model zoo can be successful in the simplest inference serving scenario, we employ the following two policies:

**Cross-hair interface:** The model serving endpoint accepts inference queries with specified minimum accuracy requirement  $a_{req}$  and maximum latency  $l_{req}$  requirement. A model will be randomly selected from the feasibility set defined by  $a_{req}$  and  $l_{req}$ . While it may make more sense to select the most cost-effective model from the feasibility set instead of random selection, we do not profile the resource requirements of each model and hence do not use it in the model selection process. The model selection policy can be easily changed to select the most cost-effective model, however, if the resource consumption is known to the profiler. If the feasibility set is empty, an “infeasible set error” will be returned to the client. Please note that the feasibility set is a subset of the Pareto frontier of the model zoo. This means the system serves models exclusively from the Pareto frontier like in [51]. A default value of 0 is set if  $a_{req}$  is not provided.

**Granularity and Boundary:** There is a granularity for accuracy ( $acc_g$ ) and a granularity for latency ( $lat_g$ ) that the inference serving system keeps track of beyond which adjacent values are indistinguishable.  $acc_g$  is less than the minimum difference between the accuracy values of any two consecutive models on the Pareto frontier. Similarly,  $lat_g$  is less than the minimum difference between the latency values of any two consecutive models on the Pareto frontier. For instance, the inference serving system might only keep track of accuracy to 0.1% and latency to 1 millisecond. Additionally, there is an upper bound for latency  $l_{up}$  in the system.

### 3.2. Adversary Model

**Adversary Goal:** To extract the most accurate model from the model zoo, given a specific latency budget, with a reasonably small number of queries. The adversary does not know the accuracy or latency specifications needed to target this model. It is important to note that this is not the same as simply extracting the most accurate model from the model zoo, as that model may have an inference latency greater than the latency budget of the adversary. This makes fingerprinting necessary as we will see in §4.2. The latency budget ( $L$ ) is associated with every query. The adversary selects  $L$  based on its intended deployment scenario for the extracted model. The latency budget can be thought of as the latency SLO of the application or task associated with the query, like image classification or language translation. The adversary will try its best not to violate this budget for as

many queries as possible so that it does not incur additional costs per query.

**Type of Attack:** This is an end-user attack because the adversary disguises itself as any other client trying to query the inference serving system for getting predictions on its input, using the cross-hair interface. Since the models are hidden in a model zoo behind a model-less interface, neither does the adversary know the accuracy or latency values of any model in the model zoo nor does it know the number of models present in the model zoo.

**Information Leakage:** The critical information that gets leaked is the accuracy and latency of each query along with the prediction made by the selected model. The (accuracy, latency) information can be seen as auxiliary information acquired by the adversary. This is crucial information that enables the attacker to learn more about the Pareto frontier of the model zoo with every query it sends to the system.

## 4. Attack on Inference Serving Systems

### 4.1. Naive Attack

TABLE 1: Accuracy and Fidelity scores of MixMatch model extraction in the single-model and the model zoo setting. Here, the victim model is DenseNet-161 and the extracted model is ResNet-50. The adversary naively tries to adapt single-model attacks to the model zoo setting, *i.e.*, without fingerprinting.

Setting	accuracy	fidelity
single model	92.00	87.13
model zoo	81.04	79.84

The MixMatch attack can be used without any modification to extract a model from a model zoo as per the adversary goal. The client has to simply specify its latency budget to the inference serving system. Since it does not know the accuracy of its victim model, it does not provide an accuracy specification (*i.e.*, the default value 0 is selected). We conduct the MixMatch attack on a model zoo (Fig. 4) that we trained on the CIFAR-10 dataset [36] and separately run experiments for the single-model setting as well, where the only model is the victim model. Tab. 1 shows the accuracy and fidelity values obtained after training MixMatch for 1024 epochs. It is clear that the attack is significantly weaker in the model zoo setting as compared to the single-model setting. The model zoo can function as a layer of protection for the victim model, which means that naively using black-box single-model attacks will not be very successful on a model zoo unless there is a definitive way to route the queries to the victim model.

### 4.2. Fingerprinting the Pareto Frontier

To make sure queries to the model zoo are routed to the victim model, we propose to fingerprint the Pareto frontier of the model zoo. Specifically, given black-box access to the model serving system, an adversary wants to find the

---

**Algorithm 1** Our Fingerprinting Algorithm. Accuracy is a discrete value between 0 and 1 and has a step size of  $acc_g$ . Similarly, latency is a discrete value ranging from 0 to a predetermined upper boundary,  $l_{up}$ , with an incremental step size of  $lat_g$ .

---

```

1: procedure FINGERPRINT( $l_{up}$ )
2:    $\mathcal{M} \leftarrow [ ]$ 
3:    $acc_{up} \leftarrow 1$ 
4:    $lat_{up} \leftarrow l_{up}$ 
5:   while  $acc_{up} \geq acc_g$  do
6:      $acc \leftarrow \text{FIND\_MAX\_ACC}(acc_{up}, lat_{up})$ 
7:      $lat \leftarrow \text{FIND\_LAT}(acc)$ 
8:     if  $acc \geq acc_g$  then
9:        $\mathcal{M}.\text{add}((acc, lat))$ 
10:    end if
11:     $acc_{up} \leftarrow acc - acc_g$ 
12:     $lat_{up} \leftarrow lat - lat_g$ 
13:  end while
14:  return  $\mathcal{M}$ 
15: end procedure
16: procedure FIND_MAX_ACC( $acc_{up}, lat_{up}$ )
17:    $acc_{low} \leftarrow 0$ 
18:    $acc_{hi} \leftarrow acc_{up} + acc_g$ 
19:   while  $acc_{hi} - acc_{low} \geq acc_g$  do
20:      $acc_{mid} \leftarrow (acc_{low} + acc_{hi})/2$ 
21:      $R \leftarrow \text{infer}(acc_{mid}, lat_{up})$ 
22:     if  $R$  is error then
23:        $acc_{hi} \leftarrow acc_{mid}$ 
24:     else
25:        $acc_{low} \leftarrow acc_{mid} + acc_g$ 
26:     end if
27:   end while
28:   return  $acc_{low} - acc_g$ 
29: end procedure
30: procedure FIND_LAT( $acc$ )
31:    $lat_{low} \leftarrow 0$ 
32:    $lat_{hi} \leftarrow l_{up} + lat_g$ 
33:   while  $lat_{hi} - lat_{low} \geq lat_g$  do
34:      $lat_{mid} \leftarrow (lat_{low} + lat_{hi})/2$ 
35:      $R \leftarrow \text{infer}(acc, lat_{mid})$ 
36:     if  $R$  is error then
37:        $lat_{low} \leftarrow lat_{mid} + lat_g$ 
38:     else
39:        $lat_{hi} \leftarrow lat_{mid}$ 
40:     end if
41:   end while
42:   return  $lat_{low} - lat_g$ 
43: end procedure

```

---

accuracy and latency profile of every model on the Pareto frontier  $\mathcal{P}$  (Eq. 1), and to leverage this information to send inference queries that consistently trigger the victim model in the model zoo.

**4.2.1. Our Fingerprinting Algorithm.** The challenge for fingerprinting is to use as few queries as possible. We propose a binary search-style algorithm. The intuition is that the system serves models exclusively from the Pareto

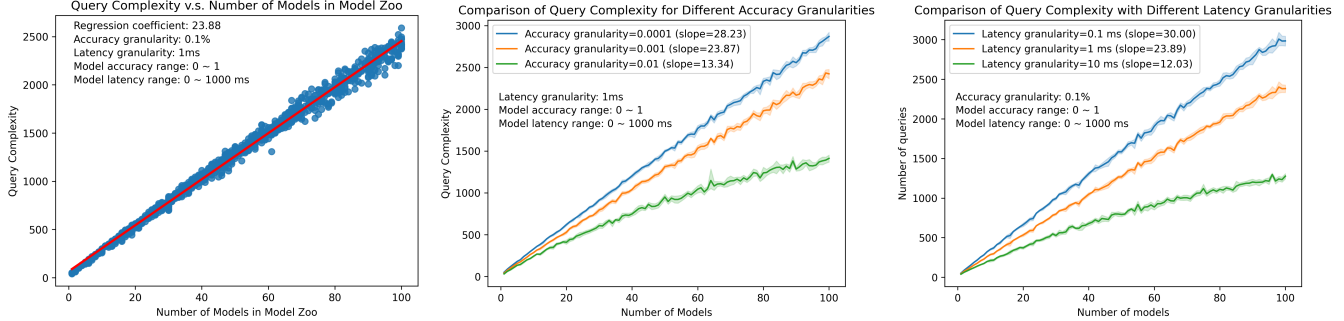
frontier (§3.1). Due to the accuracy-latency correlation in the Pareto frontier, the search space is already sorted. So we use this path for traversing the search space in our algorithm.

First, given an inference serving endpoint, we perform a binary search to find the accuracy of the most accurate model on the Pareto frontier of the model zoo. To do this, we first send a query with  $(a_{req}, l_{req}) = (0.5, \infty)$ . If the query gets a response without error, it means there is at least one model in the model zoo with accuracy  $\geq 0.5$ , and we send another query with  $(a_{req}, l_{req}) = (0.75, \infty)$ . On the other hand, if the query gets an error in response indicating that no model in the model zoo satisfies the requirement, we send another query with  $(a_{req}, l_{req}) = (0.25, \infty)$ . We stop the binary search process when we find  $a_{max}$  such that querying with  $(a_{req}, l_{req}) = (a_{max}, \infty)$  gets a successful response and querying with  $(a_{req}, l_{req}) = (a_{max} + a_g, \infty)$  gets an error.

Secondly, we use binary search to find the latency of the most accurate model on the Pareto frontier of the model zoo. We first send a query with  $(a_{req}, l_{req}) = (a_{max}, \frac{1}{2} \times l_{up})$ . If the query gets a response without error, it means the most accurate model in the model zoo has latency greater than  $\frac{1}{2} \times l_{up}$ , and we send another query with  $(a_{req}, l_{req}) = (a_{max}, \frac{3}{4} \times l_{up})$ . On the other hand, if the query gets an error in response indicating that the most accurate model has latency less than  $\frac{1}{2} \times l_{up}$ , we send another query with  $(a_{req}, l_{req}) = (a_{max}, \frac{1}{4} \times l_{up})$ . We stop the binary search process when we find  $l_{max}$  such that querying with  $(a_{req}, l_{req}) = (a_{max}, l_{max})$  gets a successful response and querying with  $(a_{req}, l_{req}) = (a_{max}, l_{max} - l_g)$  gets an error.

Next, in a similar manner, we perform a binary search to find the accuracy and latency of the second most accurate model in the model zoo within the boundary  $0 \leq a < a_{max}$  and  $0 \leq l < l_{max}$ . Then, we adjust the boundary and find the third most accurate model. The process continues till we find the accuracy and latency of all models in the model zoo. We return the result as a 2D matrix,  $\mathcal{M}$ , where the first column is  $f_{acc}(\mathcal{P})$  and the second column is  $f_{lat}(\mathcal{P})$ . Algorithm 1 describes the fingerprinting algorithm.

**Complexity Analysis.** We measure the efficiency of the fingerprinting algorithm in terms of query complexity, *i.e.*, the total number of queries an adversary needs to expend to get the accuracy and latency of all models in the Pareto frontier. The worst case query complexity of the fingerprinting algorithm is  $\mathcal{O}(n(\log \frac{1}{a_g} + \log \frac{l_{up}}{l_g}))$ , where  $n$  is the number of models in the Pareto frontier. This is because we perform binary searches for both accuracy and latency for every single model in the model zoo. However, based on our experiments in which we simulated model zoos containing models with random accuracy and latency values, we found that the fingerprinting algorithm has a linear time average query complexity (see Fig. 2a). Further, Fig. 2b and Fig. 2c illustrate the effect on the linear regression coefficient by changing the accuracy or latency granularities.



(a) The query complexity of the fingerprinting algorithm is  $\mathcal{O}(n)$ . (b) The effect of accuracy granularity on query complexity. (c) The effect of latency granularity on query complexity.

Figure 2: Fingerprinting algorithm query complexity analysis. The slope can be interpreted as the number of queries an adversary has to expend per model in the model zoo. **Left:** We see that our fingerprinting algorithm takes  $\mathcal{O}(n)$  queries in the average case, where  $n$  is the number of models in the Pareto frontier of the model zoo. **Center and Right:** We see that the query complexity remains  $\mathcal{O}(n)$  for different values of accuracy granularity and latency granularity.

## 5. Defending Against Fingerprinting

### 5.1. Proposed Defense

A wide range of defenses against single-model extraction has been proposed. One way is to perturb the output of models (*e.g.* [37], [43], [34], [35]), but it is not a viable option for the system as a legitimate client also receives perturbed outputs. Watermarking (*e.g.* [5], [55], [32]) is another defense method against model extraction. It embeds a secret pattern in the model during inference or training, but it requires post hoc analysis and the model owner to have access to the extracted model. Another possible way is to detect malicious clients (*e.g.* [33], [46]), but these methods assume that adversarial queries have small  $l_2$  distances between them and are a mix of natural and synthetic queries. This assumption does not hold in the MixMatch-based extraction attack [31].

Since our primary goal is to stop single-model attacks on the model zoo, we concentrate on obscuring the Pareto frontier during inference serving so that the auxiliary information is less useful to the attacker. Since the inference serving system only serves points from the Pareto frontier, we only want to protect the models on the Pareto frontier, not those under it. To this end, we employ a Laplace noise addition mechanism that changes the feasibility set for every query by adding noise to the accuracy and latency specifications of the query. The mechanism results in a modified feasibility set which offers more protection by potentially including a model that was not in the original feasibility set or by potentially excluding a model that was in the original feasibility set. This technique comes at the cost of utility as the initial accuracy and latency specifications may not be satisfied.

A defense should increase security while not causing harm to legitimate clients. Therefore, it is useful to measure the drop in performance with increased security. We measure the system’s performance using the **goodput**. In our context,

it is defined as the ratio of successful queries served by the inference serving system while satisfying the accuracy and latency specifications of the queries. With the defense, some models that are served will violate either the accuracy specification or latency specification, or both. This is because our noise addition mechanism modifies the feasibility set for every query. While the noise addition mechanism decreases the accuracy and fidelity of the extracted model, it inevitably reduces the goodput of the inference serving system.

The idea of injecting noise to achieve privacy is also used in differential privacy (DP) [20], [19], as well as in diverse applications like inference [59], [40], feature extraction [45], [44], cloud [39], and systems [28], [15], [60], [13]. The novelty in our defense lies in the way we employ noise-addition to protect an inference-serving system, *i.e.*, by adding noise to the accuracy and latency specifications of a declarative model-less inference serving API.

### 5.2. PSML Defense Algorithm

In order to protect the victim model in a model zoo, we must find a way to prevent fingerprinting from being successful. It is easy to see from Algorithm 1 that adding noise to the accuracy and latency values of models on the Pareto frontier would disrupt fingerprinting. However, the accuracy of a model in the zoo cannot be changed, and the latency can only be increased (by adding delay), not decreased. Another possibility is to add noise to the profiled values of accuracy and latency of each model in the PSML server (Fig. 3). Fingerprinting now would return noise-induced accuracy and latency values of the victim model. However, since we do not know when the adversary is fingerprinting and when it is collecting labeled examples for MixMatch, the modified values would continue to be used for picking models to be served. Thus, the adversary would still be able to target the victim model with the noise-induced accuracy and latency specifications it received from the fingerprinting step.

Instead, we propose adding noise directly to the input query’s accuracy and latency specifications. This causes the fingerprinting algorithm to function incorrectly. The disruption happens in lines 21 & 35 of Algorithm 1, where *infer* means querying the system. The accuracy and latency specifications of the query are perturbed. The system reads in these perturbed values and serves a model from the “modified” feasibility set. Since the latency specification of every query is the latency budget, there is a probability of inferior models being served (the system never violates the latency specification (see Algorithm 2)). The noise addition scheme will remain even after the fingerprinting step because the system does not know when a malicious client is fingerprinting. This makes the subsequent querying process uncertain, adding extra protection to the system. It is easy to see that our defense strategy will work against any single-model attack, not just model extraction. However, for ease of experimentation (§7.3), we only use the MixMatch extraction attack to show the viability of our defense.

We introduce two functions:  $f_{acc}^L(\mathcal{P}) : \mathbb{R}^{n \times 2} \rightarrow \mathbb{R}$  and  $f_{lat}^L(\mathcal{P}) : \mathbb{R}^{n \times 2} \rightarrow \mathbb{R}$ , that return the accuracy and latency of the victim model, respectively, given the adversary’s latency budget ( $L$ ). After the fingerprinting step, the adversary has the accuracy and latency values of all the models in the Pareto frontier. Next, it has to select the victim model based on these values and its latency budget  $L$ . In Algorithm 1, we fingerprint every model, regardless of the adversary’s latency budget, so that the adversary does not have to fingerprint the Pareto frontier again. Even if the adversary’s latency budget changes (increases or decreases) in the future, it can just pick the new victim model based on the values obtained from the fingerprinting step. However, the flexibility offered by fingerprinting every model comes at a price. Because the adversary violates its latency budget for some queries during fingerprinting, it incurs an extra cost for these queries.

The adversary picks row  $k$  in the Pareto frontier matrix  $\mathcal{P}$ , which has the largest latency value,  $l_k$ , below its latency budget  $L$ . Thus,  $f_{lat}^L(\mathcal{P}) = l_k$ . Then, it selects the accuracy value  $a_k$  from row  $k$ . Thus,  $f_{acc}^L(\mathcal{P}) = a_k$ . The obtained pair of values  $(a_k, l_k)$  serve as the adversary’s accuracy and latency specifications of every subsequent query to the inference serving system. Therefore, the latency budget is strictly obeyed by the adversary after fingerprinting.

We add Laplace noise to  $f_{acc}^L(\mathcal{P})$  and  $f_{lat}^L(\mathcal{P})$  while using a single parameter ( $\epsilon$ ) to quantify the amount of noise added. More formally, we need to quantify the maximal possible change of both accuracy and latency values, denoted by  $\Delta f_{acc}^L$  and  $\Delta f_{lat}^L$ , respectively. This way, we can measure the strengths of noise for different functions using  $\epsilon$ . We discuss implementation details in §6. Algorithm 2 describes how we serve models with the defense. The key idea is not to serve models with latency values greater than the latency specification. The input latency specification,  $lat$ , is assumed to be the latency budget for every query after the fingerprinting step. Noise addition will change this latency value to  $\tilde{lat}$ .

---

**Algorithm 2** Model Serving with Defense for a Query

---

```

1: procedure SERVE_MODEL( $\mathcal{P}, L$ )
2:    $\mathcal{F} \leftarrow [ ]$ 
3:    $acc \leftarrow f_{acc}^L(\mathcal{P})$ 
4:    $lat \leftarrow f_{lat}^L(\mathcal{P})$ 
5:    $Y_{acc} \leftarrow Y \sim Lap(\Delta f_{acc}^L/\epsilon)$ 
6:    $Y_{lat} \leftarrow Y \sim Lap(\Delta f_{lat}^L/\epsilon)$ 
7:    $\tilde{acc} \leftarrow acc + Y_{acc}$ 
8:    $\tilde{lat} \leftarrow lat + Y_{lat}$ 
9:   for model in  $\mathcal{P}$  do
10:    if (model.acc  $\geq \tilde{acc}$ ) and (model.lat  $\leq \tilde{lat}$ ) and
        (model.lat  $\leq lat$ ) then
11:       $\mathcal{F}.add(model)$ 
12:    end if
13:  end for
14:  return PICK_RANDOM_MODEL( $\mathcal{F}$ )
15: end procedure

```

---

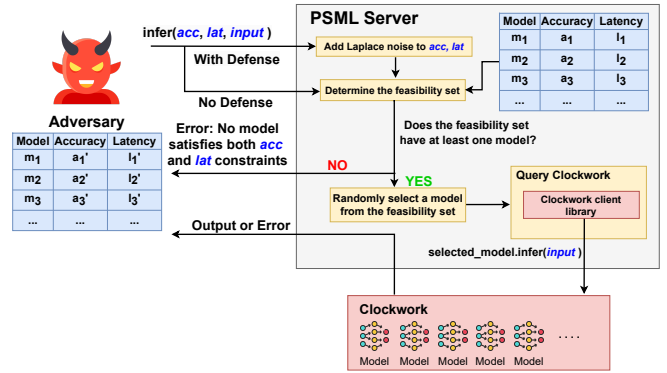


Figure 3: The system design and workflow for PSML server. After the models are uploaded and profiled in the PSML server, inference serving takes place through the interaction between the server process and the Clockwork process. The PSML server routes the model architecture to be served to Clockwork, and Clockwork returns the output back to the server, which is then returned to the client.

## 6. System Design and Implementation

We build a model-less inference serving system on top of Clockwork [23], with plans to open source to the community. Figure 3 illustrates our system design used to perform experiments with our proposed attack and defense mechanisms. Building upon the policies outlined in §3.1.2, we leveraged Clockwork as the model server. Over Clockwork’s client library, we implemented a shim layer (PSML Server) in C++ that exposes an inference API. This API allows clients to submit inference queries with specific minimum accuracy specification  $acc$ , maximum latency specification  $lat$ , and input to perform the inference. Thus, the entire system becomes model-less.

Upon loading models into the model zoo, the PSML server maintains a copy of the accuracy and latency profiles for each model. In the absence of any defense mechanism, upon receiving an inference query, the PSML Server ran-



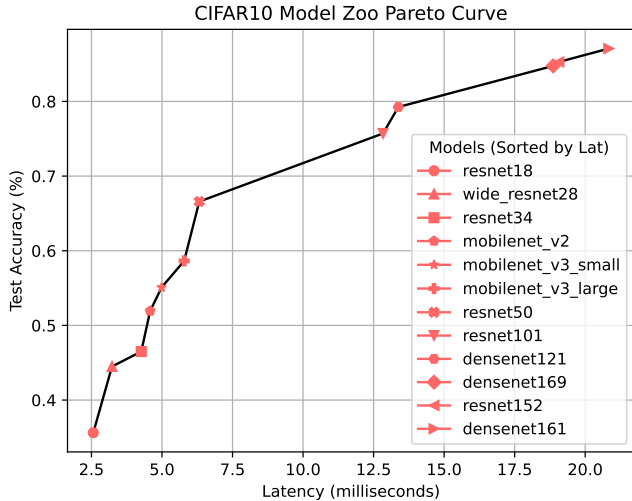


Figure 4: The Pareto frontier of the model zoo that we trained on CIFAR-10 and use in our experiments. We maintain a good heterogeneity of model architectures for the image classification task. Models are not trained to their full potential to represent a realistic Pareto frontier and allow us to experiment in large, medium, and small model extraction settings.

domly selects a model from the feasibility set based on a uniform distribution. In other words, every model in the feasibility set has an equal probability of being chosen. The PSML Server then forwards the inference query input to Clockwork, which serves the query using the selected model, and relays the output from Clockwork back to the client. An error is returned if no model satisfies both the accuracy and latency constraints (*i.e.*, the feasibility set is empty).

With the defense mechanism active, the PSML server introduces Laplace noise to *acc* and *lat* before determining the feasibility set (Algorithm 2). The Laplace noise is calculated using the `boost::math::laplace_distribution` and `boost::math::quantile` functions from the C++ `boost` library. Since the latency specification of any query cannot be violated, the system does not serve models with latency values greater than the input latency specification. This case may arise when the noise-induced latency exceeds the latency specification.

## 7. Evaluation

### 7.1. Experiment Setup

We evaluate our fingerprinting-based attack and our noise-based defense using the PSML server on top of a real-world inference serving system, Clockwork[23]. The training and profiling of models are done using NVIDIA GeForce RTX 2080 Ti GPUs. We employ a model extraction attack (the MixMatch attack in [31]), which we treat as a black-box, on a zoo of image classification models trained on the CIFAR-10 [36] and SVHN [41] datasets, like in

the original paper. To populate the model zoos for the two datasets, we did not train the models to their full potential on the respective training sets, so that we could represent a realistic Pareto frontier. Training all the models we selected to their full potential on CIFAR-10 or SVHN will result in all models being in the high-accuracy region ( $\geq 90\%$  accuracy), which does not reflect a realistic Pareto frontier. Additionally, this would not allow us to show that a medium or small model ( $< 80\%$  accuracy) can be successfully:

- 1) extracted using our fingerprinting algorithm, or
- 2) protected against the fingerprinting-based attack using our noise-based defense

We loaded the models, shown in Fig. 4 and Fig. 9, into Clockwork. The extracted model (*i.e.*, the model that the attacker starts with) is a ResNet-50, which can reach  $> 95\%$  accuracy on CIFAR-10 and SVHN. Thus, the model has enough expressive power to learn weights from the CIFAR-10 and SVHN train sets. Based on the client’s desired accuracy and latency specifications, the query is routed to a model that satisfies them. Otherwise, the system doesn’t fulfill the query and sends an “infeasible set” error message to the client. It is important to note that our attack and defense have nothing to do with the training, architecture, or weights of the neural network models in the zoo. Therefore, our methods are plug-and-play, *i.e.*, no modification is needed on real-world model zoos that are usually dense and have models trained to their full potential on complex datasets like ImageNet [18].

The 12 image classification models on the Pareto frontiers of our model zoos are comprised of various architectures of ResNets, WideResNets, DenseNets, and MobileNets. All of these models have predictable inference latency values. According to the definition of the Pareto frontier of a model zoo, models with larger inference latency have higher accuracy. While training our models, we followed this accuracy-latency correlation, as shown in Figure 4. We consider three querying scenarios with differing latency budgets available to the adversary: large, medium, and small latency budgets. Since with a larger latency budget, the adversary’s victim model is a larger-sized model, we also call these scenarios the large, medium, and small model extraction cases. The query budget available to the adversary is 4000 queries in all cases. The inputs of these queries are randomly selected from the training set of the datasets. Hence, the adversary has to do the fingerprinting and querying for MixMatch within 4000 queries. We show how the adversary’s success varies with different query budgets in Fig. 5.

We set up a shim layer on top of Clockwork (see §6), which includes the model selection logic that utilizes the accuracy and latency specifications of the client. The selected model information is relayed to the Clockwork controller node, which subsequently assigns the inference task to the appropriate worker node, and the inference is performed on this model using the input provided by the client. Please note that the actual inference time of the query is not used by the adversary at all in our attack.

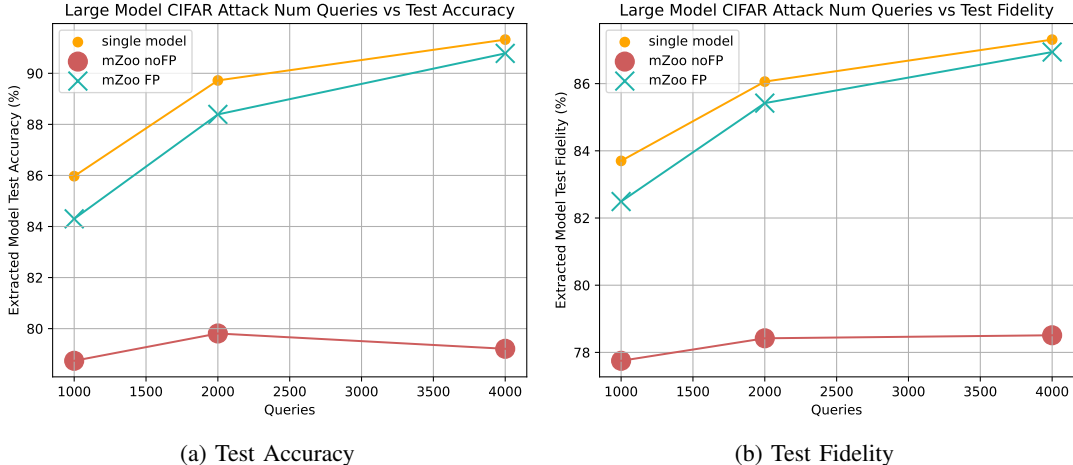


Figure 5: Accuracy and Fidelity of the extracted model for different query budgets in the large model setting. These experiments are done on CIFAR-10. We see that the fidelity and accuracy of the extracted model increase with increasing query budget of the adversary. However, in some cases, increasing the number of queries will not result in a better accuracy or fidelity score, as seen in the no-fingerprinting case (accuracy is higher with 2000 queries). This is because having more labeled points doesn’t necessarily mean higher quality labeling in the no-fingerprinting case, as many of these points are labeled by very low accuracy models.

## 7.2. Attack

We aim to demonstrate our fingerprinting algorithm’s potential in adapting single-model attacks to inference serving systems. We show that our fingerprinting algorithm improves the extracted model accuracy and fidelity compared to using the black-box attack without the fingerprinting step. It is important to note that fingerprinting takes a constant number of queries for a given Pareto frontier, when  $acc_g$  and  $lat_g$  are fixed. This is because Algorithm 1 is deterministic.

We run attacks in each setting on both datasets using a predetermined query budget. A query budget of  $q$  means that the attacker can query the system at most  $q$  times. Thus, it has to do the fingerprinting and get labeled points within these  $q$  queries. The attacker’s latency budget per query for the large-model setting is 21 ms, while it is 13 ms for the medium-model setting and 5 ms for the small-model setting. Thus, the victim model according to our adversary goal in §3.2 is DenseNet-161 in the large-model setting (top right corner of the Pareto frontiers in Fig. 4 and Fig. 9), ResNet-101 in the medium-model setting (middle of the Pareto frontiers) and MobileNetV3-small in the small-model setting (bottom left of the Pareto frontiers). In each test, we let the MixMatch attack train for 1024 epochs. Fidelity is measured against the victim model on the test set of the datasets.

**7.2.1. Experiment Results.** The experiment results are shown in Tab. 2 and Tab. 3. Fig. 6 shows the training plots for the medium model. Results with SVHN are in Tab. 5 and Tab. 6 in Appendix A. We show a relation between the number of queries available to the attacker and the fidelity and accuracy of the extracted model in Fig. 5. In the single-model attack, all 4000 queries were answered by the victim model. In the no-fingerprinting and fingerprinting attacks,

all the queries were served using the PSMML server. A total of 411 queries were used for fingerprinting the model zoo trained on CIFAR-10. According to our threat model in §3.2, the attacker does not know the accuracy of the highest-accuracy model in the model zoo that has latency lower than its latency budget. On the other hand, the accuracy and latency values for all the models in the zoo were determined by the fingerprinting step in the fingerprinting attack. Therefore, in the subsequent step of querying the system, the attacker simply chooses the accuracy and latency specifications corresponding to the model with the highest inference latency below its latency budget (see §4.2).

It is clear from Tab. 2 and Tab. 3 that the adversary can fingerprint the entire model zoo by expending a relatively low number of queries. This ability enables it to determine precise accuracy and latency values of the victim model. The test fidelity and test accuracy results show that a black-box attack on a model zoo can be as successful as an attack on a single model. We can also see that the single-model attack is slightly better. This is because the number of queries used to fingerprint the model zoo is subtracted from the total query budget of the adversary. This means there are fewer labeled points from the victim model for the MixMatch attack. The main takeaway from these experiments is that an adversary can extract the highest accuracy (or largest) model with latency lower than a given latency budget from a model zoo using our fingerprinting algorithm.

## 7.3. Defense

To demonstrate that our defense works on an actual inference serving system, we evaluate our noise-based defense mechanism on our inference serving system. The goal of the defense is to undermine the fingerprinting step. By introducing random noise in the accuracy and latency

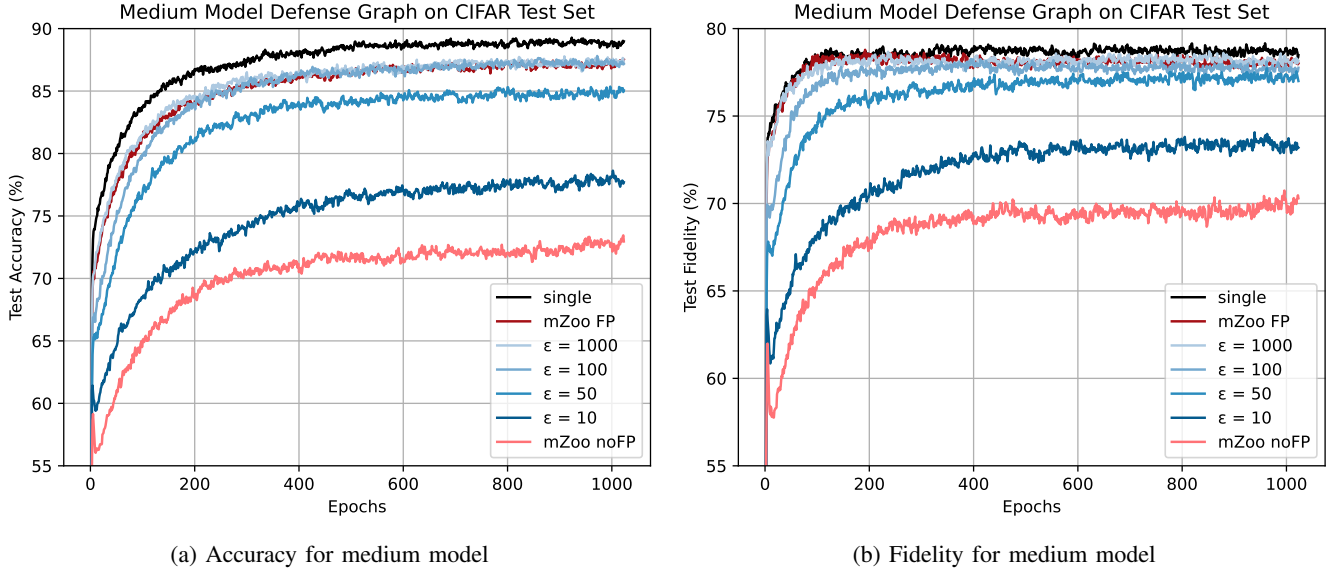


Figure 6: Fidelity and Accuracy values during extraction of the medium model with CIFAR-10. We see the training of the MixMatch method for different settings. Our fingerprinting algorithm successfully extracts the model with just 4000 queries. The level of defense can be configured with the  $\epsilon$  parameter: lower  $\epsilon$  means more protection.

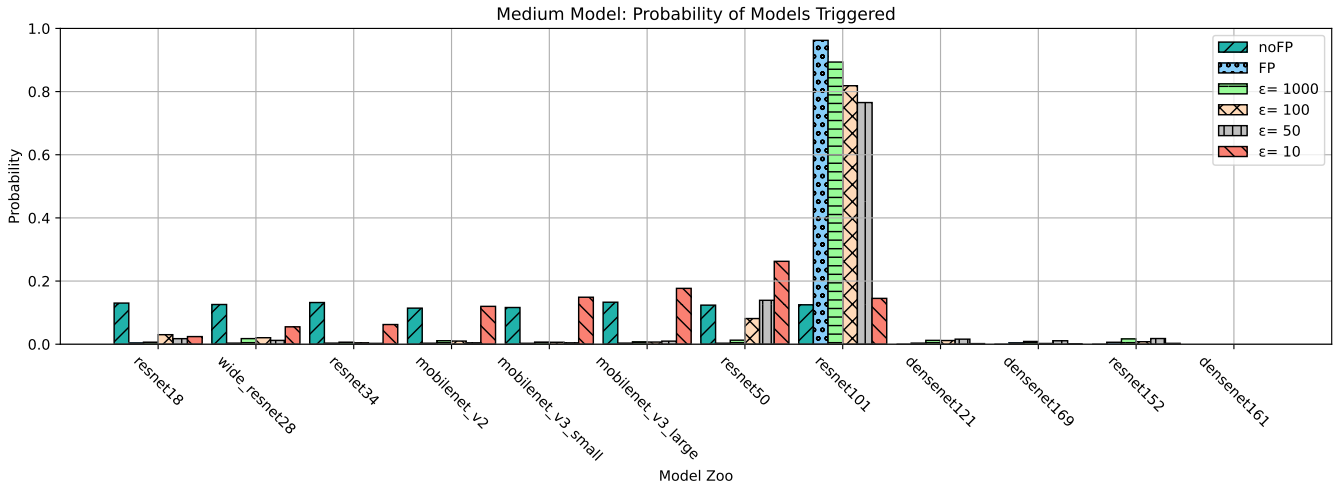


Figure 7: Probability mass functions plotted over the model zoo with different styles representing different attack/defense scenarios (legend). ResNet-101 is the victim model. The plot shows: (a) ineffective extraction attack of the victim model without fingerprinting (noFP); (b) effective attack with high probability of calling the target model with proposed fingerprinting (FP); (c) mitigating the efficacy of the proposed attack with noise-based defense, dissipating the PMF over the base.

specifications given as input by the attacker, we hope to reduce the accuracy of the fingerprinting step. At the same time, we do not want to destroy the utility of the inference serving system. Hence, we study the impact of adding noise on the system’s goodput. The Laplace mechanism described in §5 is applied to the fingerprinting algorithm that is treated as a function. Since protection against the extraction attack will be at odds with the system’s performance, we hope to show the trade-off between them in this section through our experiment results.

We conduct our defense experiments on the same set-

tings under which the attack is tested. Since the fingerprinting step is vital for the attack to succeed, we aim to see to what extent our defense mechanism reduces the effectiveness of the fingerprinting step. Ideally, we would like to reduce the fidelity and the accuracy of the extracted model to the values obtained when the attack was run without the fingerprinting step (Tab. 2, Tab. 3). We add noise to the latency and accuracy specifications of the client. The rationale behind this is that by changing the specifications of the model that the inference serving system has to pick, we reduce the chances of successfully fingerprinting any model

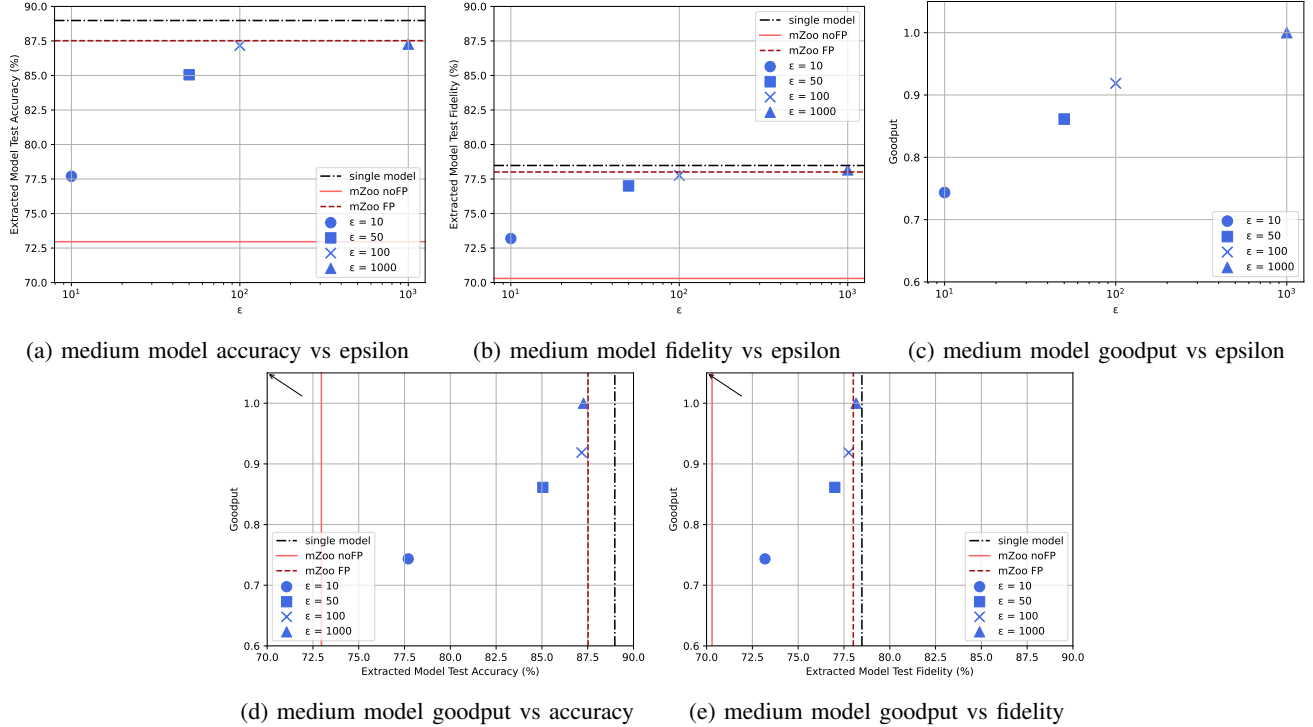


Figure 8: **Top row:** The relationship between goodput, fidelity, accuracy, and different  $\epsilon$  values for extracting medium models trained on CIFAR-10. **Bottom row:** Goodput plotted against accuracy and fidelity. The desirable direction is shown with black arrows. Clearly, with decreasing  $\epsilon$ , the accuracy and fidelity decrease, while the goodput also decreases. A good  $\epsilon$  value should reduce the attack’s success and have an acceptable goodput ( $\geq 80\%$ ). We see  $\epsilon = 50$  is the best  $\epsilon$  value out of the tested  $\epsilon$  values for medium models.

TABLE 2: Accuracy values of the extracted model under different settings after training the MixMatch method for 1024 epochs. The model zoo is trained on CIFAR-10, and the query budget is 4000. We see that the fingerprinting attack achieves accuracy close to that in the single-model setting for diverse model sizes. The defense successfully protects medium and small models from extraction with the given  $\epsilon$  values.

Setting	accuracy of extraction		
	small	medium	large
single model	67.29	88.98	92.00
mzoo no-FP	60.08	72.96	81.04
mzoo FP	66.29	87.52	91.35
Defense $\epsilon=1000$	68.21	87.27	91.24
Defense $\epsilon=100$	66.83	87.17	90.77
Defense $\epsilon=50$	65.19	85.05	90.81
Defense $\epsilon=10$	62.98	77.70	90.88

TABLE 3: Fidelity values of the extracted model under different settings after training the MixMatch method for 1024 epochs. The model zoo is trained on CIFAR-10, and the query budget is 4000. We see that the fingerprinting attack achieves fidelity close to that in the single-model setting for diverse model sizes. The defense successfully protects medium and small models from extraction with the given  $\epsilon$  values.

Setting	fidelity of extraction		
	small	medium	large
single model	64.63	78.48	87.13
mzoo no-FP	56.54	70.30	79.84
mzoo FP	64.2	78.01	87.13
Defense $\epsilon=1000$	63.29	78.16	86.74
Defense $\epsilon=100$	61.13	77.76	86.51
Defense $\epsilon=50$	60.94	77.00	86.85
Defense $\epsilon=10$	58.66	73.19	86.74

from the zoo.

The performance of the inference serving system is measured in terms of goodput, defined in Section 5.1. The system’s goal is to be as performant as possible while maintaining a required degree of protection from fingerprinting. Our defense mechanism only changes the client’s accuracy and latency specifications to the system and not the quality of the models on the Pareto frontier.

In the experiments, we set the system with sensitivity values according to the model zoo shown in Fig. 4:

$\Delta f_{acc} = 1$  and  $\Delta f_{lat} = 18.271$  for CIFAR-10. For each dataset, we demonstrate extracting a large, medium, and small model. In these three scenarios, the latency budgets of the attacker are 21 ms, 13 ms, and 5 ms, respectively, for CIFAR-10. Our experiments verify whether launching the black-box single-model attack (MixMatch) with fingerprinting on a system that has our defense shows accuracy and fidelity comparable to that in the no-fingerprinting case (*i.e.*, a naive attacker using single-model attacks on a model zoo without fingerprinting the zoo first).

In the defense, the number of queries taken to do the fingerprinting is not constant for a given model zoo because of the noise in the system. Moreover, not all queries will be answered by the system. A breakdown of the number of queries used in each step is presented in Tab. 4. Since the number of queries taken for defense is not constant, we did three trials for each setting and reported the average. After querying, the MixMatch method is trained for 1024 epochs on the points in the training set. Fig. 7 shows the probability distribution of the models getting triggered in our different attack scenarios.

TABLE 4: Mean number of queries used in each step for medium model extraction trained with CIFAR-10. Query budget is 4000. Fingerprinting without defense always takes a constant number of queries. In the defense, query complexity of fingerprinting reduces with decreasing  $\epsilon$ . We also show the number of queries that pass/fail.

Setting	number of queries			
	fingerprinting	labeling	successful	failed
single model	0	4000	4000	0
mzoo no-FP	0	4000	4000	0
mzoo FP	411	3589	3589	0
Defense $\epsilon=1000$	901.33	3098.67	3093.33	5.33
Defense $\epsilon=100$	864.33	3135.67	2310.00	825.67
Defense $\epsilon=50$	867.33	3132.67	2007.00	1125.67
Defense $\epsilon=10$	406.00	3594.00	3075.33	518.67

**7.3.1. Experiment Results.** We observe that the attack is less successful in terms of accuracy and fidelity with the defense in place. With  $\epsilon = 10$ , we get up to 4.8% drop in fidelity and 9.8% drop in accuracy when compared with the fingerprinting-based attack without the defense, in the medium model case (Fig. 6). The closer the value of accuracy and fidelity is to those in the no-fingerprinting setting, the better the protection against the model extraction attack. From Tab. 2 and Tab. 3, we see that the fidelity and accuracy scores decrease with decreasing epsilon values. This is because lower epsilon results in more noise. And more noise in the system results in more queries being routed to models that are not the victim model.

Due to noise in the system, the adversary’s query may be routed to a model with a latency larger than the query’s latency specification. In such situations, the system does not send the output to the query, as shown in Algorithm 2. Since the adversary’s latency specification is its latency budget for all queries after the fingerprinting step, all the outputs after fingerprinting are from models that have inference latency less than the adversary’s latency budget. Due to the relationship between accuracy and latency on the Pareto frontier, this also means that all queries are answered by either the victim model or by models with accuracy less than the accuracy of the victim model.

Since higher protection comes with lower performance, we calculated the goodput scores of these runs. We observe in Fig. 8 that goodput decreases with decreasing epsilon values. The reason behind this is the fact that a lower epsilon value means more noise. More noise in the system

means there is a high chance the query is being served by a model from outside the feasibility set, which is defined by the client’s unmodified accuracy and latency specifications. Goodput greater than 0.8 ( $\geq 80\%$  queries meet specifications provided by the client) is generally agreed to be acceptable for an inference serving system. From Fig. 8, we can see that goodput greater than 0.8 is obtained with  $\epsilon = 50$  in the medium model setting, while the fidelity and accuracy of the attack reduce significantly. Hence, the system designer can use this  $\epsilon$  value to effectively defend against fingerprinting-based attacks while maintaining an acceptable quality-of-service (QoS). The main takeaway is that a balance between protection and performance can be reached by configuring  $\epsilon$  in our defense mechanism for medium and small models.

We see that the defense is more effective in the medium and small model settings than in the large model setting. This is because while extracting a large model, even a large amount of noise in the system may result in a high-accuracy model being served. Since high-accuracy models result in high-quality labeled examples, the MixMatch attack will not suffer much, and the resulting accuracy values will remain fairly high. We provide a lower bound on fidelity while extracting a large model in Appendix B. Thus, our defense mechanism with the tested  $\epsilon$  values is more effective for attacks targeting medium to small models. However,  $\epsilon$  can be configured to suit the system designer’s needs.

## 8. Conclusion

We make an observation that model extraction attacks make outdated assumptions that the victim model can be explicitly specified and directly queried. This is no longer true in state-of-the-art ML model serving systems. A novel, query-efficient fingerprinting algorithm re-enables model extraction, bridging the performance gap lost to implicit and dynamic model switching. Indeed, the proposed attack comes close to the single model serving setting, which  $\mathcal{P}_{\text{SML}}$  absorbs as a special case. The proposed attack is shown effective over a wide range of model latencies, successfully extracting large, medium, and small-sized models from the zoo hidden behind the layer of model-less abstraction. We defend against the proposed attack with a novel defense mechanism based on perturbation of (latency, accuracy) constraint specification with  $\epsilon$ -controlled noise. Doing so helps expose a practical tradeoff space between the system’s level of defense and its performance, captured by its goodput. We show that robust levels of defense can be achieved with acceptable loss in system goodput. The proposed attack and defense mechanisms are instantiated in a real system, with plans to open source to the community. With a growing number of proprietary models served via implicit model selection in state-of-the-art inference systems, we make the first step towards better understanding the security implications of such systems with respect to Intellectual Property theft through model extraction.

## References

- [1] TensorFlow Serving for Model Deployment in Production, 2018. <https://www.tensorflow.org/tfx/guide/serving>.
- [2] The ONNX Model Zoo. <https://github.com/onnx/models>, 2020.
- [3] Nvidia Triton. <https://developer.nvidia.com/nvidia-triton-inference-server>, May 2023.
- [4] Qualcomm Works with Meta to Enable On-device AI Applications Using Llama 2. <https://www.qualcomm.com/news/releases/2023/07/qualcomm-works-with-meta-to-enable-on-device-ai-applications-usi>, July 2023.
- [5] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1615–1631, 2018.
- [6] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 10(3):137–150, 2015.
- [7] Payman Behnam, Jianming Tong, Alind Khare, Yangyu Chen, Yue Pan, Pranav Gadikar, Abhimanyu Bambhaniya, Tushar Krishna, and Alexey Tumanov. Subgraph stationary hardware-software inference co-design. In *Proceedings of the 6th Conference on Machine Learning and Systems, MLSys’23*, 2023.
- [8] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. *Advances in neural information processing systems*, 32, 2019.
- [9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [10] Nicholas Carlini, Matthew Jagielski, and Ilya Mironov. Cryptanalytic extraction of neural network models. In *Advances in Cryptology—CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, pages 189–218. Springer, 2020.
- [11] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. {SIMC}:{ML} inference secure against malicious clients at {Semi-Honest} cost. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1361–1378, 2022.
- [12] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Exploring connections between active learning and model extraction. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pages 1309–1326, 2020.
- [13] Chien-Ying Chen, Debopam Sanyal, and Sibin Mohan. Indistinguishability prevents scheduler side channels in real-time systems. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 666–684, 2021.
- [14] Jacson Rodrigues Correia-Silva, Rodrigo F Berriel, Claudine Badue, Alberto F de Souza, and Thiago Oliveira-Santos. Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [15] Jorge Cortés, Geir E Dullerud, Shuo Han, Jerome Le Ny, Sayan Mitra, and George J Pappas. Differential privacy in control and network systems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4252–4272. IEEE, 2016.
- [16] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. Inferline: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 477–491, 2020.
- [17] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *NSDI*, volume 17, pages 613–627, 2017.
- [18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- [19] Cynthia Dwork, Krishnamurthy Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology—EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28–June 1, 2006. Proceedings 25*, pages 486–503. Springer, 2006.
- [20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4–7, 2006. Proceedings 3*, pages 265–284. Springer, 2006.
- [21] Neil Zhenqiang Gong and Bin Liu. You are who you know and how you behave: Attribute inference attacks via users’ social friends and behaviors. In *USENIX Security Symposium*, pages 979–995, 2016.
- [22] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving DNNs like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 443–462. USENIX Association, November 2020.
- [23] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving dns like clockwork: Performance predictability from the bottom up. *arXiv preprint arXiv:2006.02464*, 2020.
- [24] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, et al. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *The Journal of Machine Learning Research*, 21(1):845–851, 2020.
- [25] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. The architectural implications of facebook’s dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–501. IEEE, 2020.
- [26] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, et al. Applied machine learning at facebook: A datacenter infrastructure perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 620–629. IEEE, 2018.
- [27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision—ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.
- [28] Zhenqi Huang, Yu Wang, Sayan Mitra, and Geir E Dullerud. On the cost of differential privacy in distributed control systems. In *Proceedings of the 3rd international conference on High confidence networked systems*, pages 105–114, 2014.
- [29] Doug Hudgeon and Richard Nichol. Machine Learning for Business: Using Amazon Sagemaker and Jupyter. <https://aws.amazon.com/sagemaker>, 2020.
- [30] David Isele, Reza Rahimi, Akansel Cosgun, Kaushik Subramanian, and Kikuo Fujimura. Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2034–2039. IEEE, 2018.

- [31] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pages 1345–1362, 2020.
- [32] Hengrui Jia, Christopher A Choquette-Choo, Varun Chandrasekaran, and Nicolas Papernot. Entangled watermarks as a defense against model extraction. In *USENIX Security Symposium*, pages 1937–1954, 2021.
- [33] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.
- [34] Sanjay Kariyappa, Atul Prakash, and Moinuddin K Qureshi. Protecting dnns from theft using an ensemble of diverse models. In *International Conference on Learning Representations*, 2021.
- [35] Sanjay Kariyappa and Moinuddin K Qureshi. Defending against model stealing attacks with adaptive misinformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2020.
- [36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.
- [37] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against neural network model stealing attacks using deceptive perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*, pages 43–49. IEEE, 2019.
- [38] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In *USENIX Security Symposium*, pages 2201–2218, 2021.
- [39] Sam Leroux, Tim Verbelen, Pieter Simoens, and Bart Dhoedt. Privacy aware offloading of deep neural networks. *arXiv preprint arXiv:1805.12024*, 2018.
- [40] Fatemehsadat Miresghallah, Mohammadkazem Taram, Prakash Ramrakhiani, Ali Jalali, Dean Tullsen, and Hadi Esmaeilzadeh. Shredder: Learning noise distributions to protect inference privacy. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–18, 2020.
- [41] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.
- [42] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4954–4963, 2019.
- [43] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Prediction poisoning: Towards defenses against dnn model stealing attacks. *arXiv preprint arXiv:1906.10908*, 2019.
- [44] Seyed Ali Osia, Ali Shahin Shamsabadi, Sina Sajadmanesh, Ali Taheri, Kleomenis Katevas, Hamid R Rabiee, Nicholas D Lane, and Hamed Haddadi. A hybrid deep learning architecture for privacy-preserving mobile analytics. *IEEE Internet of Things Journal*, 7(5):4505–4518, 2020.
- [45] Seyed Ali Osia, Ali Taheri, Ali Shahin Shamsabadi, Kleomenis Katevas, Hamed Haddadi, and Hamid R Rabiee. Deep private-feature extraction. *IEEE Transactions on Knowledge and Data Engineering*, 32(1):54–66, 2018.
- [46] Soham Pal, Yash Gupta, Aditya Kanade, and Shirish Shevade. Stateful detection of model extraction attacks. *arXiv preprint arXiv:2107.05166*, 2021.
- [47] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.
- [48] Adnan Siraj Rakin, Md Hafizul Islam Chowdhury, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1157–1174. IEEE, 2022.
- [49] Robert Nikolai Reith, Thomas Schneider, and Oleksandr Tkachenko. Efficiently stealing your machine learning models. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 198–210, 2019.
- [50] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. Infaas: Automated model-less inference serving. In *USENIX Annual Technical Conference*, pages 397–411, 2021.
- [51] Manas Sahni, Shreya Varshini, Alind Khare, and Alexey Tumanov. Compofa: Compound once-for-all networks for faster multi-platform deployment. *arXiv preprint arXiv:2104.12642*, 2021.
- [52] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.
- [53] Wenting Zheng Srinivasan, PMRL Akshayaram, and Popa Raluca Ada. Delphi: A cryptographic inference service for neural networks. In *Proc. 29th USENIX Secur. Symp.*, pages 2505–2522, 2019.
- [54] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.
- [55] Sebastian Szyller, Buse Gul Atli, Samuel Marchal, and N Asokan. Dawn: Dynamic adversarial watermarking of neural networks. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 4417–4425, 2021.
- [56] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [57] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX security symposium*, volume 16, pages 601–618, 2016.
- [58] Jean-Baptiste Truong, Pratyush Maini, Robert J Walls, and Nicolas Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4771–4780, 2021.
- [59] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S Yu. Not just privacy: Improving performance of private deep learning in mobile cloud. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2407–2416, 2018.
- [60] Yu Wang, Zhenqi Huang, Sayan Mitra, and Geir E Dullerud. Differential privacy in linear distributed control systems: Entropy minimizing mechanisms and performance tradeoffs. *IEEE Transactions on Control of Network Systems*, 4(1):118–130, 2017.
- [61] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE international symposium on high performance computer architecture (HPCA)*, pages 331–344. IEEE, 2019.
- [62] Fei Xu, Jianian Xu, Jiabin Chen, Li Chen, Ruitao Shang, Zhi Zhou, and F. Liu. igniter: Interference-aware gpu resource provisioning for predictable dnn inference in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 34:812–827, 2022.
- [63] Jeong-Min Yun, Yuxiong He, Sameh Elnikety, and Shaolei Ren. Optimal aggregation policy for reducing tail latency of web search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 63–72, 2015.
- [64] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

## Appendix A. Attack Results on SVHN

This section presents the results of using our fingerprinting-based attack on a model zoo trained on the SVHN dataset.

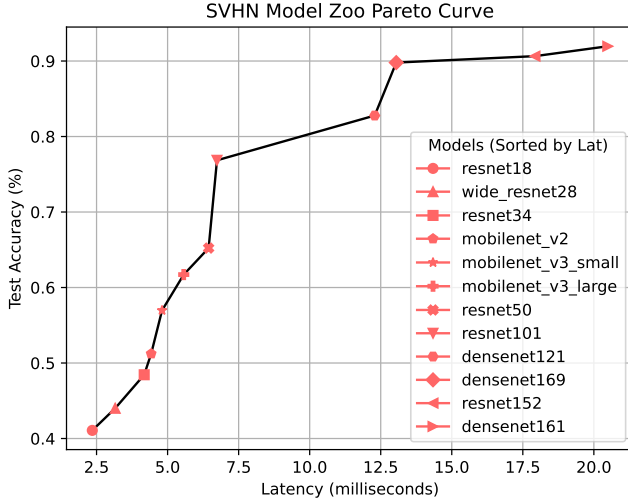


Figure 9: The Pareto frontier of the model zoo that we trained on SVHN.

TABLE 5: Accuracy values of the extracted models under different settings. The model zoo is trained on SVHN. MixMatch method is trained for 1024 epochs. The query budget is 4000.

Setting	small	medium	large
single model	56.67	91.46	95.66
mzoo no-FP	49.83	77.16	87.61
mzoo FP	56.12	88.98	95.69

TABLE 6: Fidelity values of the extracted models under different settings. The model zoo is trained on SVHN. MixMatch method is trained for 1024 epochs. The query budget is 4000.

Setting	small	medium	large
single model	56.34	76.75	92.40
mzoo no-FP	49.17	68.22	85.41
mzoo FP	56.30	76.70	92.39

## Appendix B. Proof of Lower Bound on Fidelity for Large Models

How is the fidelity of the extracted model with respect to the victim model related to their accuracy scores on the same test set?

To justify the defense, we need to show that it reduces the fidelity and accuracy scores. Extraction with noise in the system will result in an “aggregate” extracted model. Thus, our goal is to show that this aggregate extracted model is poor in terms of fidelity and accuracy, i.e., it has a low fidelity score with respect to the victim model. It is safe to assume that a high-accuracy feasibility set will result in a high-accuracy extracted model. So now we will try to **minimize** the fidelity score between an aggregate extracted model and a victim model.

Let us calculate the minimum fidelity possible for an aggregate extracted model (i.e., obtained with the defense) from a high-accuracy feasibility set.

*Proof.* Let  $D$  be the test set of points for a classification task with  $k$  classes. Let  $S_i$  be the set of points from  $D$  that model  $m_i$  classifies correctly. There are two models of concern here:  $m_v$  (the victim model) and  $m_e$  (the extracted model).

Let  $a_v$  and  $a_e$  be the accuracy scores on  $D$  of models  $m_v$  and  $m_e$ , respectively. We assume that  $a_v \geq 0.9$  and  $a_e \geq 0.9$ , as the extracted model  $m_e$  is obtained from querying a high-accuracy feasibility set and the victim model  $m_v$  lies in this feasibility set.

Let  $n(A)$  stand for the number of distinct elements in the set  $A$ . Let  $F$  be the fidelity set, i.e., the set of all such points in  $D$  for which  $m_v$  and  $m_e$  predict the same class.  $F$  can be broken down into two disjoint sets:

- 1) a set of points on which both models make the correct prediction, i.e.,  $(S_v \cap S_e)$ .
- 2) a set of points on which both models make incorrect predictions but predict the same class. We denote this set by  $P$ .

Thus,  $F = (S_v \cap S_e) + P$  and  $n(F) = n(S_v \cap S_e) + n(P)$

$$n(S_v \cap S_e) = n(S_v) + n(S_e) - n(S_v \cup S_e) = a_v \cdot n(D) + a_e \cdot n(D) - n(S_v \cup S_e)$$

Hence,  $n(F) = a_v \cdot n(D) + a_e \cdot n(D) - n(S_v \cup S_e) + n(P)$

We can minimize the above as follows:  
 $\min(n(F)) = a_v \cdot n(D) + a_e \cdot n(D) - \max(n(S_v \cup S_e)) + \min(n(P)) = a_v \cdot n(D) + a_e \cdot n(D) - n(D) + 0 = n(D) \cdot (a_v + a_e - 1)$

Thus, the minimum fidelity score is  $\frac{\min(n(F))}{n(D)} = (a_v + a_e - 1) = 0.8$

**Conclusion:** Attacking a high accuracy (or large) model results in a relatively high fidelity score ( $\geq 0.8$ ) even with the defense. Therefore, we need to attack a lower accuracy (or small) model to get a significant reduction in fidelity (or improvement in protection) with the defense. ■



## Appendix C.

### MixMatch Model Extraction

MixMatch [8] is a semi-supervised learning method that guesses low-entropy labels for data-augmented unlabeled examples and mixes labeled and unlabeled data using MixUp [64]. Given a batch of labeled examples with one-hot targets and an equally sized batch of unlabeled examples, MixMatch produces a processed batch of augmented labeled examples and a batch of augmented unlabeled examples with “guessed” labels. These two batches are then used in computing separate labeled and unlabeled loss terms.

Data augmentation is used on both labeled and unlabeled data. For each point in the batch of labeled data, an augmented version is generated. For each point in the batch of unlabeled data,  $K$  augmentations are generated. These individual augmentations are used to generate a “guessed” label using a label-guessing process. Both labeled examples and unlabeled examples with label guesses are used in MixUp. The loss function combines cross-entropy loss between labels and model predictions from the batch of augmented labeled examples with squared  $L_2$  loss between guessed labels and model predictions from the batch of augmented unlabeled examples.

Jagielski *et al.* [31] leverage MixMatch as an extraction technique on the SVHN [41] and CIFAR-10 [36] datasets. For both datasets, inputs are color images of  $32 \times 32$  pixels belonging to one out of ten classes. The victim model is a WideResNet-28-2 architecture that achieves 97.36% and 95.75% accuracy on SVHN and CIFAR-10, respectively. The adversary is given access to the same training set but without knowledge of the labels. The results of the MixMatch attack show that the adversary needs to query the victim model on a small subset of the training points to extract a model whose accuracy on the task is comparable to the victim model’s. MixMatch is able to exploit a few labels because of the prior it is able to build using the unlabeled data. This results in improved test set accuracy and fidelity.