A Generative Framework for Low-Cost Result Validation of Machine Learning-as-a-Service Inference

Abhinav Kumar
Saint Louis University
St. Louis, U.S.
abhinav.kumar@slu.edu

Miguel A. G. Aguilera
New Mexico State University
Las Cruces, U.S.
guirao@nmsu.edu

Reza Tourani
Saint Louis University
St. Louis, U.S.
reza.tourani@slu.edu

Satyajayant Misra
New Mexico State University
Las Cruces, U.S.
misra@nmsu.edu

ABSTRACT

The growing popularity of Machine Learning (ML) has led to its deployment in various sensitive domains, which has resulted in significant research focused on ML security and privacy. However, in some applications, such as Augmented/Virtual Reality, integrity verification of the outsourced ML tasks is more critical–a facet that has not received much attention. Existing solutions, such as multi-party computation and proof-based systems, impose significant computation overhead, which makes them unfit for real-time applications. We propose *Fides*, a novel framework for real-time integrity validation of ML-as-a-Service (MLaaS) inference. Fides features a novel and efficient distillation technique–Greedy Distillation Transfer Learning–that dynamically distills and fine-tunes a space and compute-efficient verification model for verifying the corresponding service model while running inside a trusted execution environment. Fides features a client-side attack detection model that uses statistical analysis and divergence measurements to identify, with a high likelihood, if the service model is under attack. Fides also offers a re-classification functionality that predicts the original class whenever an attack is identified. We devised a generative adversarial network framework for training the attack detection and re-classification models. The evaluation shows that Fides achieves an accuracy of up to 98% for attack detection and 94% for re-classification.

CCS CONCEPTS

• **Security and privacy** → **Domain-specific security and privacy architectures**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

Verifiable computing, result verification, trusted execution environment, machine learning as a service, edge computing.

ACM Reference Format:
Abhinav Kumar, Miguel A. G. Aguilera, Reza Tourani, and Satyajayant Misra. 2024. A Generative Framework for Low-Cost Result Validation of Machine Learning-as-a-Service Inference. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3634737.3657015

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07…$15.00
https://doi.org/10.1145/3634737.3657015

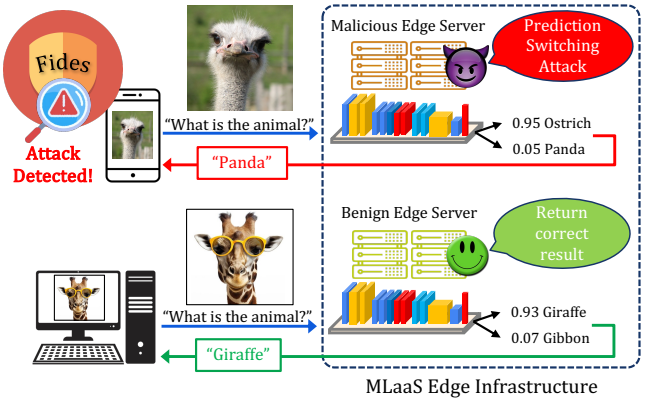


Figure 1: We consider the integrity verification of Machine Learning-as-a-Service inference, where clients send their data to the edge servers for ML inference tasks. In our proposed framework, Fides, we aim to detect any malicious misclassification caused by a malicious edge server when running the clients' inference tasks.

1 INTRODUCTION

The increasing popularity of machine learning (ML) applications and the plethora of data generated by smart and connected devices, such as smartphones, Internet of Things devices, and video cameras, has led to the development of sophisticated and complex ML applications, such as autonomous driving and cognitive assistance [74]. However, the challenges of processing such a high volume of data to support real-time applications have led to the development of several edge-computing solutions for autonomous driving [37], healthcare [55], and Augmented/Virtual Reality applications [59].

This motivates inference-based ML-as-a-Service (*i.e.,* MLaaS), in which trusted ML providers deploy their services on third-party edge servers as query-based APIs. These APIs allow the clients to share their data with the edge servers, which are running ML applications, to provide the processed result. However, the edge computing trust model differs from the Cloud, primarily due to its distributed and multi-stakeholder nature [64]. Given that any entity may host a third-party server, outsourcing ML applications to the edge ecosystem raises potential privacy and security issues (Figure 1), including model/data inference and Trojan attacks [6, 35, 36, 46, 62, 66, 75]. Two of the most pertinent attacks against the integrity of MLaaS are Trojan attacks [6, 75] and adversarial example [51, 63]. In Trojan attacks, the attacker partially modifies

the neural network using a poisoned dataset to include a set of samples with a pattern trigger, aiming to cause misclassifications when the model observes the pattern trigger. Backdoor attacks can be devised either during model training or on a pre-trained model [6, 62, 71, 75]. An adversarial attack (*i.e.,* adversarial example) exploits the inherent vulnerabilities of neural networks for post-training misclassifications. These attacks are input-specific in that each input data should be individually perturbed to cause a misclassification.

The existing techniques for preserving the integrity of outsourced computation use cryptosystems, such as multi-party computation [26, 29, 60, 72], proof-based systems [16, 34, 40], and homomorphic encryption [40, 47, 49, 69]. A few initiatives have built frameworks using a Trusted Execution Environment (TEE) to offload parts of the ML applications [18, 65]. Despite their merits, these solutions are often computationally expensive, impose significant latency overhead, or do not scale with the complexity of the evolving ML models. Thus, making them impractical for real-time and latency-sensitive applications. Some solutions utilize multiple models in the form of redundant computing or majority voting [21, 45]. Redundant computing techniques, however, require higher degrees of redundancy to achieve stricter integrity guarantees, which leads to significantly higher computational overhead. Moreover, redundant computing techniques are built on the presumption that the only relevant part of a redundant model's output is the class with the highest probability (predicted class), which is untrue. *The model's posterior vector provides information about the learned knowledge representation through the probabilities assigned to the incorrect classes.* This insight is used in knowledge distillation for training higher accuracy and fidelity compressed versions of cumbersome models [23]. We believe the same insight can be used to study the impact of an adversary on two models trained on identical distributions. Figure 2 shows the density distribution of Kullback–Leibler (KL) divergence between two models (ResNet50 and ResNet152) and the drastic change in the distribution when an adversary manipulates the prediction generated by one of the models (ResNet152). We will analyze the impact of an adversary on the divergence trends (Section 2) to create a result validation framework.

Our Framework: To cope with the attacks that target MLaaS inference integrity, we propose Fides. In a nutshell, Fides validates the integrity of MLaaS inference by securely running a special-purpose *verification model*. This model is constructed through knowledge distillation, guaranteeing that the knowledge representation of the verification model closely approximates that of the service model. Fides comprises two primary components. The first one is a resource-efficient model distillation technique–Greedy Distillation Transfer Learning (GDTL)–which generates a customized verification model for a given service model. GDTL distills the knowledge of the service model into the verification model by incrementally unfreezing and fine-tuning the last layers of the model. This leads to only a few layers being distilled at a time, which makes it suitable for resource-constraint and TEE-based private training paradigms [21, 43] The second component is a novel generative framework for training a client-side attack detection model that uses posterior vectors of the service and verification models, alongside their divergences, on a given input to detect a potential attack. Our framework also trains an attack re-classification model,

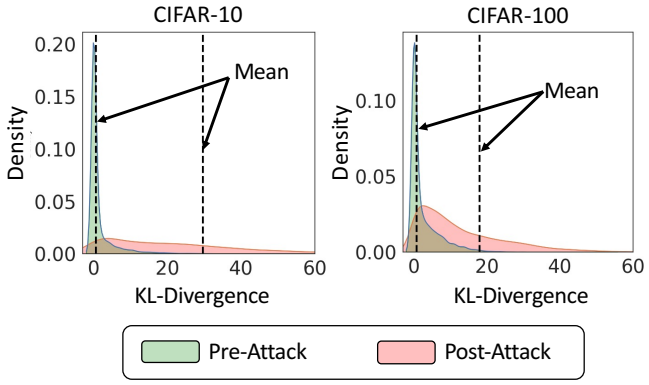


Figure 2: The KL divergence density distribution between two models (ResNet50 and ResNet152) before (green) and after (red) attack on one of the models (ResNet152). Both models are trained on Cifar10 and Cifar100 datasets. The KL divergence between the benign models' posterior vectors belongs to a distribution with low mean and variance (green distribution). Performing a prediction-switching attack against one of the models (ResNet152) leads to a significant increase in the distribution's mean and variance (red distribution). We use this identifiable behavior in designing Fides. (distributions' tails are truncated for better visualization).

aiming to predict the correct result of the outsourced inference task by learning a typical adversary's goal and behavior in the system under attack.

We evaluate Fides' performance using three datasets, including CIFAR-10, CIFAR-100, and ImageNet, across three neural network architectures and also assess its computational overhead on multiple constrained devices. We compare Fides with Slalom [65] and Chiron [27], and show that Fides achieves 4.8×–26.4× and 1.7×–25.7× speed-up to Slalom and Chiron, respectively.

The novel **contributions** of Fides are as follows:
(i) We propose Fides–a framework for result validation of MLaaS inference using an efficient verification model running on a TEE to corroborate the results of the service model. This makes Fides a perfect candidate for real-time applications, such as autonomous driving, and medical imaging.
(ii) In Fides, we propose a greedy distillation transfer learning (GDTL) technique for training the verification model, aiming to reduce the distillation's overhead by incrementally unfreezing layers of the verification model for fine-tuning to the desired accuracy.
(iii) We introduce two client-side shallow neural networks for detecting and resolving adversarial attacks, with negligible computation overhead. We propose a generative framework for effective training of these models.
(iv) We systematically assess Fides using major benchmark datasets of varying complexity and number of detection classes on three architecturally different deep neural networks: namely ResNet [22], DenseNet [25], and EfficientNet [61]. Our results show that Fides outperforms existing solutions in terms of computation complexity while achieving high attack detection and remediation accuracy.

We share the observation that motivated our design in Section 2. Section 3 presents the background. In Section 4, we elaborate on system model, threat model and assumptions, and the detail of the attack modeling. Section 5, we present the detail of our design.

Section 6 presents our experimental insights. Section 7 includes the related work. Section 8 shares our conclusion.

2 OBSERVATION

In this section, we will discuss our findings regarding the disparities between two models, which were trained on datasets with similar distributions, where one is subjected to an attack that causes unintended mispredictions. The insights drawn from these observations, coupled with the evident trend we have identified, lay the foundation for the design of Fides.

Consider two neural networks, F_θ^1 and F_θ^2, where they are trained on the data distribution $\mathcal{D}$. F_θ^1 is a larger primary model, and F_θ^2 is a significantly smaller model trained on the same primary task. *We speculate that the posterior vector of the smaller model (F_θ^2) can be used to detect the impact of a malicious actor on the bigger model (F_θ^1)*, even if they disagree on the prediction. This is because the decision boundarys of the F_θ^2 significantly overlaps with the decision boundary of F_θ^1. An adversary will either modify the decision boundary or the sample distance from the decision boundary in F_θ^1 to conduct the attack. Hence, the posterior vector of F_θ^2 can be used to detect these modifications. To test our speculation we orchestrate prediction-switching attacks, *i.e.,* adversarial attacks in which malicious actors modify the posterior vectors of F_θ^1 during inference. We observe the divergence between the posterior vector of F_θ^1 and F_θ^2 and assess if we can establish some trends that can be utilized to detect the presence of a malicious actor. As such, we analyze the divergence density distribution of F_θ^1 and F_θ^2–both before and after conducting a prediction-switching attack against F_θ^1. In comparing the divergence trends, *i.e.,* $D(F_\theta^1(x), F_\theta^2(x))$, we consider the following cases:

- Natural Agreement **(Case A)**: happens if $argmax(F_\theta^1(x)) = argmax(F_\theta^2(x))$ prior to any adversarial modifications to the prediction.
- Natural Disagreement **(Case B)** : happens if $argmax(F_\theta^1(x)) \neq argmax(F_\theta^2(x))$ prior to any adversarial modifications to the prediction.

Using these two cases, we aim to recognize identifiable patterns between naturally occurring disagreements and those resulting from malicious actions. In our analysis, we employ three datasets: CIFAR-10, CIFAR-100, and ImageNet, and three architectures: ResNet, DenseNet, and EfficientNet (more details in Section 6). For assessing divergence, we use Jeffreys' Divergence and Wasserstein-1 measures. Figure 3 shows the JD distributions between the output of the two models across all datasets and architectures. We identified evident trends between pre-attack and post-attack JD distributions:

Trend 1: Consider Case A, where both models naturally agree on the predictions before the attack. When the two models agree with each other, and neither of them is under attack, the distribution exhibits small standard deviation and mean values, as depicted by the green curve in Case A. In contrast, when one of the models is subjected to an attack, the JD distribution undergoes a significant transformation, with a sharp increase in both the mean and standard deviation values (red curve in Case A).

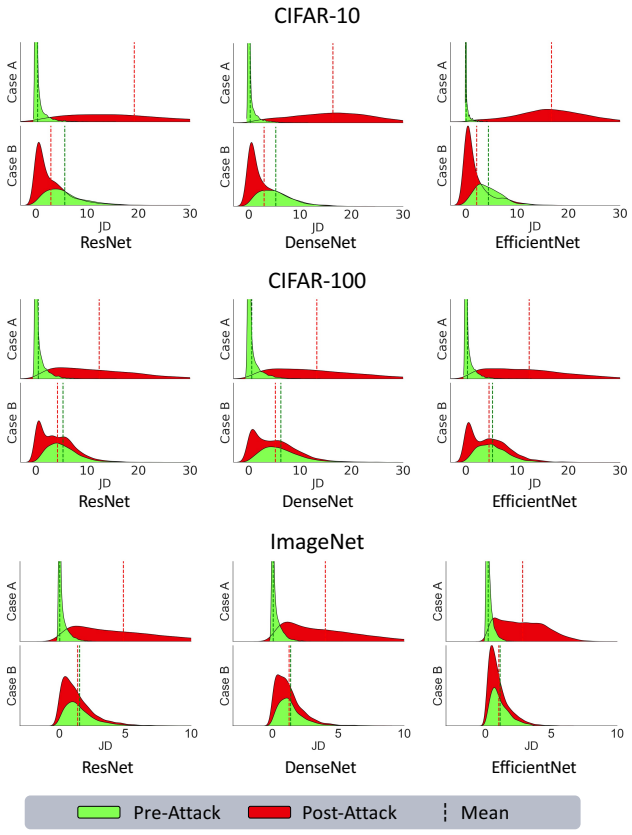


Figure 3: The JD measurements of two models. For Case A, the attack increases the divergence value, whereas in Case B (disagreement), the attack decreases the divergence value. Thus validating our reasoning for divergence's role in attack detection.

Trend 2: Consider Case B, where the two models predict different classes before the attack. We notice a contrasting transformation once one of the models is attacked. Specifically, when the system is not under attack, the differences between the two models' outputs result in a wide JD distribution with a relatively larger standard deviation and mean values, as shown by the green curves in Case B. In contrast, the attack leads to a narrow distribution with smaller standard deviations and mean values (red curves in Case B).

The trends we have identified, which remained consistent across various architectures and datasets, and the observations from Figure 3 are the key supporting elements that empirically validate our intuition behind the impact of an attack on the similarity of two probability distributions. In particular, the results confirm that *if the two models agree with each other, an attack will lead to an increase in the mean and standard deviation of the divergence, resulting in a wide distribution. In contrast, when the two models have natural disagreements, the attack decreases the mean and standard deviation of the divergence distribution, leading to a narrow distribution.* We also measured the similarity of the two models using the Wasserstein-1 metric to show independence from any particular similarity measure. Our analysis shows similar trends in different value ranges (refer to Table 3 and Table 4 in Appendix A). The observations shown above, however, pose a major challenge. There are overlaps between the pre-attack and post-attack distributions across

all cases. Moreover, increasing the complexity of the dataset and the dimensionality of the output vector leads to an increase in the overlap. This growing overlap and complexity serve as indicators of a more complex decision boundary for the attack discrimination task, which can potentially be realized through a neural network. We will address this challenge in our design (Section 5.2).

3 BACKGROUND

In this section, we introduce the concepts relevant to our work, including a trusted execution environment, ML compression techniques, and divergence-based similarity measures.

3.1 Trusted Execution Environments

A Trusted Execution Environment (TEE) is an isolated processing environment in which each application is allocated a region of memory, called an enclave, which is protected even from processes running at higher privilege levels [2, 57]. The TEE guarantees the integrity of the run-time states, the confidentiality of the code and data stored on persistent memory, and the authenticity of the executed code and its correct execution via remote attestation between the data owner and the secure enclave. The two common TEE implementations are Intel SGX [8] and ARM TrustZone [1].

3.2 Model Compression

Model Quantization [20] is a strategy for model compression and inference process acceleration. Quantization reduces the precision of the model's weight, gradient, or activation values by converting floating point numbers to lower precision integer numbers [77]. Thus, reducing the storage needed for the model and speeding up the inference task [78]. The two primary quantization strategies include post-training quantization, where the weights of a trained model will be compressed (*e.g.,* 32-bit float to 8-bit integer), and quantization-aware training, in which the model's weight and gradients are quantized during the training process [7].

Knowledge distillation (KD) [23] is a machine learning technique for transferring knowledge from a complex neural network(s) (*i.e.,* teacher model(s)) to a single model (*i.e.,* student model), aimed at model compression, robustness, and performance enhancement [3, 14, 24]. The distillation process entails training the student model using the output feature maps of the teacher model, *i.e.,* soft labels, and their corresponding true labels. The distilled models are more robust against adversarial attacks when compared with their teacher models [41, 50]. Defensive distillation [52] is a technique that helps in reducing the effectiveness of the adversarial example attacks against DNNs.

3.3 Similarity Measures

Divergence-based similarity measures are statistical techniques to assess the similarity of a probability distribution (P) with respect to the reference probability distribution (Q). We review the Jeffreys' divergence (JD) and Wasserstein distance.

Jeffreys' Divergence. The JD is the bounded symmetrization of the Kullback–Leibler divergence (KLD), which is used to quantify the similarity between two probability distributions P and Q. Unlike KLD, the JD distance represents a normalized score that is symmetric, *i.e.,* $D_J(P \parallel Q) = D_J(Q \parallel P)$ and can be calculated as:

$$D_J(P \parallel Q) = \frac{1}{2} D_{KL}\Big(P \parallel Q\Big) + \frac{1}{2} D_{KL}\Big(Q \parallel P\Big),$$

where $D_{KL}(P \parallel Q)$ is the KLD between P and Q. JD provides a smoothed and normalized score compared to KLD.

Wasserstein-1 Metric. Wasserstein-1 distance is a measure of the distance between two probability distributions. Informally, the Wasserstein distance can be interpreted as the minimum energy cost of moving the mass from x to y to transform probability distribution P to Q. Wasserstein-1 metric (D^1_W) between cumulative distribution functions P and Q is defined as the infimum, taken over the set of all joint distribution $\Gamma(P, Q)$. Thus, the Wasserstein-1 distance between P and Q ($D^1_W(P, Q)$) is:

$$D^1_W(P, Q) = \inf_{\gamma \in \Gamma(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||].$$

4 MODELS AND ASSUMPTIONS

In this section, we first describe our system model and then formalize our threat model and discuss security assumptions. Our system comprises edge servers, ML service providers, and clients. Service providers offer a range of resource-intensive machine learning services, such as image annotation or video analytics, which require input data from the clients. Given the complex nature of the computation load and the need for real-time computation of these services (*e.g.,* autonomous driving, multi-player gaming, and traffic monitoring), clients use the services running on the edge server rather than the distant Cloud. We envision a subset of the servers to be equipped with low-end trusted hardware like Intel SGX or AMD TrustZone.

4.1 System Model

Our system comprises a computing ecosystem, service providers, and clients. We consider the quickly growing pervasive edge computing (PEC) environment [64] as our computing ecosystem. The PEC ecosystem emphasizes the inclusion of client-owned devices in the pool of computing resources, allowing them to carry out clients' requests. Thus, resulting in extensive heterogeneous resource pools in the proximity of clients. In the PEC ecosystem, a server is either a pre-deployed infrastructure server or a client's device, such as smartphones and laptops, which runs outsourced services, aiming to generate revenue. In contrast to the infrastructure servers that are static in nature, the client's resources can intermittently join and leave the pool of computing resources. In our system, service providers offer a range of services to clients. We consider resource-intensive machine learning services like image annotation or video analytics, which require input data from the clients. Due to the need for quick computation, the client utilizes edge servers instead of using a distant Cloud.

We also consider an access control service such as the recently proposed APECS [12] for the servers and clients to mutually authenticate each other before outsourcing ML task. This construction prevents unauthorized users from accessing the services and also prevents unauthorized servers from accessing the clients' data. Considering the heterogeneity of the PEC ecosystem, we envision a subset of the servers to be equipped with low-end trusted hardware like Intel SGX or AMD TrustZone.

4.2 Threat Model

Attacks against computation outsourcing may target confidentiality, integrity, availability, and privacy. Among all, we focus on threats to integrity. Thus, the primary objective of Fides is to provide correctness verifiability for the output of the MLaaS inference task. More specifically, given an ML task and its input data, the goal is to enable the client to assess the reliability of the result and infer, with high probability, the trustworthiness of the executed service by the edge server.

We formalize the threat model by introducing a *security game*, in which the adversary $\mathcal{A}$ generates malicious predictions and deceives the challenger C into believing that the generated predictions belong to the universe $\mathcal{U}$ of benign input, prediction pairs.

Detection Game: The game utilizes a data universe $\mathcal{U} := \{(x_i, F_\theta(x_i))\}_{i=1}^N$, where $F_\theta : x_i \rightarrow [0,1]^n$. The function F_θ is realized through a machine learning model and trained using algorithm τ_F. While both the challenger C and Adversary $\mathcal{A}$ have complete knowledge of input distribution $\mathbb{D}$, trained model F_θ, training algorithm τ_F, and attack detection algorithm τ_F^D, the challenger has no knowledge of the attack algorithm τ_F^A used by adversary.

(1) The challenger samples the dataset $\mathcal{D} \subseteq \mathbb{D}$ and trains model $F_\theta \leftarrow \tau_F(D)$, where $F_\theta : x_{i=1}^N \rightarrow y_{i=1}^N$ *and* $(x_i, y_i)_{i=1}^N \in \mathcal{U}$.
(2) The challenger gives adversary a white-box access to F_θ.
(3) The adversary runs $\tau_F^A : (x, y) \rightarrow \tilde{y}$, such that $(x, \tilde{y}) \cap \mathcal{U} = \phi$, and returns $\{(x, y), (x, \tilde{y})\}$.
(4) The challenger generates a guess $z \in \{y, \tilde{y}\}$ by running: $\tau_F^D(\{y, \tilde{y}\}; x, F_\theta)$.
(5) The adversary wins if $(x, z) \notin \mathcal{U}$.

The adversary can implement τ_F^A in several different ways, including orchestrating Trojan attacks on image or text classification data [6, 75], performing attacks on the input image in the form of adversarial perturbation [51, 63], or leveraging their access to the model and directly modifying the model weights or the prediction vector.

We do not restrict the adversary's access to the service model, which is running in the insecure region of the general purpose processor. The adversary has complete knowledge of the architecture and parameters of the service model. We assume the adversary knows the deployed defense mechanism or its components within a trusted enclave but does not have access to any computations taking place inside the enclave. So, they can use the knowledge of the defense algorithm τ_F^D to orchestrate targeted attacks.

4.3 Attack Modeling

The primary outcome of the attack we consider in this work is the integrity violation of the MLaaS inference task. To achieve this goal, the adversary can use different techniques. In what follows, we discuss three prominent attack orchestration techniques and implement them in our evaluation. All the attacks take place at the edge server, with the attacker deciding which of the different attacks to execute. The attacker's goal is to have the service model classify an input with a wrong label. We note that while the adversary can completely control the execution of the service model, the adversary cannot interrupt the verification model as it runs in the secure enclave upon the client's request. We implemented all these attacks and tested Fides against them (refer to Section 6).

Prediction Switching Attacks. In the following attack methodology, the adversary only targets the posterior vector of the deployed machine learning model and modifies it directly, aiming to cause an incorrect prediction. We call this attack *naive prediction switching attack* if the attacker modifies or generates the predictions arbitrarily, without any attempt to avoid the detection mechanism. With the knowledge of a detection mechanism, the adversary may deploy different strategies, such as averaging the two highest probability values p_1 and p_2 ($\mu = \frac{p_1+p_2}{2}$) to switch the prediction by assigning $\mu + \epsilon$ to the second class (*i.e.,* wrong prediction) and $\mu - \epsilon$ to the first class (*i.e.,* true prediction), using a small ϵ value, *e.g.,* 0.01% of μ. Alternatively, the adversary trains its own verification model and uses it to minimize the distance between the forged incorrect prediction and the actual prediction of the service model. We call these *advanced prediction switching attacks*.

Well-known Attacks. The adversary can use popular backdoor or adversarial sampling techniques to modify the predictions. For our evaluation, we consider the following attacks:

Fast Gradient Sign Method (FGSM) [17] perturbations are crafted by calculating the loss between the prediction and the true label. Using the calculated loss, FGSM creates a max-norm constrained (ϵ) perturbation. Given image x, the adversarial image x^{adv} can be calculated as $x^{adv} = x + \epsilon \times \text{sign}(\nabla_x J(\theta, x, y_{true}))$.

Projected Gradient Descent (PGD) [32] acts as iterative extension of FGSM. The adversarial image is crafted by repeatedly adding perturbation, guided using the loss between the prediction and the target class. Each step of adversarial image generation can be formulated as $x_{N+1}^{adv} = Clip_{X,\epsilon}(x_N^{adv} + \text{sign}(\nabla_x J(\theta, x, y_{target})))$, where $x_0^{adv} = x$.

Trojan Attacks: This attack is conducted using the service model in a white-box setting. In the context of our work, the adversary can train the service model with a poisoned dataset containing trigger embedded images [5] [19]. Alternatively, the adversary adds more layers to the service model as a Trojan module and trains it using the poisoned data [62], which gets triggered when the input image contains the embedded trigger.

5 DESIGN OF FIDES

In a nutshell (refer to Figure 4), the process of Fides starts with a service provider preparing the service package (§ 5.1), including the ML application, *i.e.,* service model, and its verification component, *i.e.,* verification model, for deployment on the edge servers (the server-side component). The service provider also builds an attack detection and re-classification pipeline, that includes two neural networks (§ 5.2), one for attack detection and a second for re-classification. In training the attack detection and re-classification models, the provider uses a generative adversarial network approach. The service deployment process involves loading the service and verification models to a server with which a trust relationship has already been established (§ 5.3). After deployment, the server accepts requests for verification from a client. On receiving a client's request, the server runs the verification model inside the secure enclave and the service model in the unprotected region of the processor and returns both outputs to the client (§ 5.4). Finally,

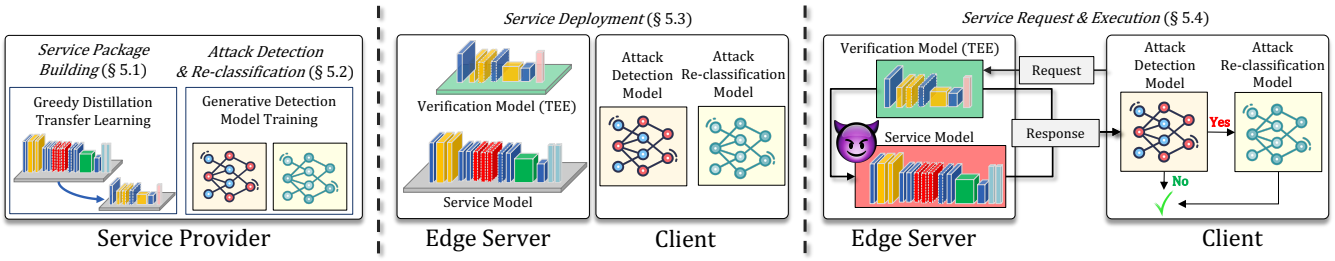


Figure 4: In Fides, the provider builds the service package (§ 5.1) and the attack detection and re-classification pipeline (§ 5.2) for deployment on the server (§ 5.3) and the client, respectively. The client then sends the service request to the edge server and verifies the result using the attack detection pipeline (§ 5.4).

the client runs Fides' detection and re-classification functionalities (the client-side component) to identify a potential attack and rectify the outcome of the MLaaS inference as needed. Without loss of generality, we use Intel SGX as the TEE in our implementation reference and explain details using SGX terminology.

5.1 Service Package Building

A service package includes two components–the application (*i.e.,* service model) and a verification tool (*i.e.,* verification model), which is a small ML model for the verification of its corresponding service model. As shown in Figure 5, service package development is a two-step process in which the provider first trains its service model, *e.g.,* image classification, and then dynamically re-train layers of the verification model using the fully trained service model.

As per Algorithm 1, the service building process takes an untrained service model (F_S), an independently pre-trained verification model (F_V), the privately owned training set ($\mathbb{D}_T^{Priv}$), and a set of hyper-parameters (λ and α). Upon completion, it returns the fully trained service (F_S) and verification (F_V) models. Training the service model follows the standard training process using $\mathbb{D}_T^{Priv}$ with the defined loss function for as many epochs as required (Lines 1-3). The verification model is a distilled and smaller version of the service model, so it renders minimal computation overhead when deployed into the edge server's enclave. To reduce the cost of knowledge distillation, we propose *Greedy Distillation Transfer Learning* (GDTL), a distillation technique that results in a time-efficient procedure for training the small verification model.

The GDTL process takes a pre-trained model that is significantly smaller as compared to the service model and adaptively unfreezes the layers that require re-training and fine-tuning. More specifically, GDTL first splits the verification model starting from its last layer (Line 8). Given the model split, GDTL runs F_S and F_V on $\mathbb{D}_T^{Priv}$ to obtain the soft labels of the service model and the partially trained verification model (Lines 8-10). In Line 10, GDTL calculates the knowledge distillation loss (*i.e.,* KDL) of these values and the one-hot encoding (τ) of the labels (Line 6). Note that the weight of the average (α) changes over time. GDTL starts with a larger α value, giving more weight to the teacher model, and then adaptively decreases it over time, giving higher weight to service true label. The rationale for adaptively changing α is to initially guide the model towards the service model and then decrease it for it to improve over samples the teacher model is providing incorrect prediction. GDTL then uses the loss value and the stochastic gradient descent method to update the trainable layers (Line 11) using $\mathbb{D}_T^{Priv}$.

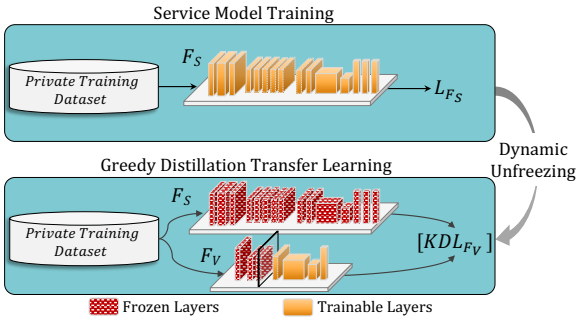


Figure 5: In service package building, the provider trains the service model and uses it to run the greedy distillation transfer learning process, which builds the customized compressed verification model.

Each training iteration, GDTL compares the accuracy and precision of the partially fine-tuned verification model with the service model. If the verification model's accuracy is higher than the λ factor of the service model's accuracy, the GDTL process stops training F_V (Lines 7). Otherwise, GDTL dynamically unfreezes another layer of the verification model for fine-tuning (Line 12). In essence, the GDTL process gradually splits the model from the last layer towards the first and continuously re-trains and fine-tunes the trainable layers. We note that λ is a hyper-parameter that the provider adjusts based on various factors like its available computing resources and the desired verification accuracy. A λ value closer to 1 results in a high-accuracy verification model but requires more fine-tuning steps. Our initial assessment (Figure 6) shows that GDTL outperforms standard distillation and fine-tuning in terms of accuracy and is comparable to fine-tuning in terms of per-epoch execution time. GDTL also reduces the difficulty of finding suitable regularization, which is the primary reason behind the lower accuracy of classical knowledge distillation [70].

For further compression, GDTL applies model quantization as a common technique for approximating a neural network that uses floating-point numbers using a neural network of lower bit-width numbers. Among all, in Fides, we used dynamic range quantization (Line 13), which is a post-training approach that does not need additional model re-training and fine-tuning. Dynamic range quantization converts 32-bit floating point numbers into 8-bit integers, with the resultant model running using floating point operations. At the end of this process, the service package is completed. It should be pointed out that the design of the distillation algorithm relies on an underlying assumption that pre-trained weights utilized are trusted. Violation of this assumption can lead to vulnerabilities [71], which are out of the scope of this work.

Algorithm 1: Service Package Development

Input: F_S (untrained), F_V (public pre-trained), $\mathbb{D}_T^{Priv}, \lambda \in [0,1], \alpha \in [0,1]$.

Output: $\langle F_S, F_V \rangle$.

Service Model Training:

for *number of the training epochs* **do**
  Use stochastic gradient descent to update F_S on: $\mathcal{L}_{F_S}(F_S(x_i), y_i), \forall (x_i, y_i) \in \mathbb{D}_T^{Priv}$
Store F_S ▷ *F_S is the fully-trained Service model.*
$\bullet_{j=1}^{n} F_V \leftarrow F_{V1} \bullet F_{V2} \bullet \cdots \bullet F_{Vn}$

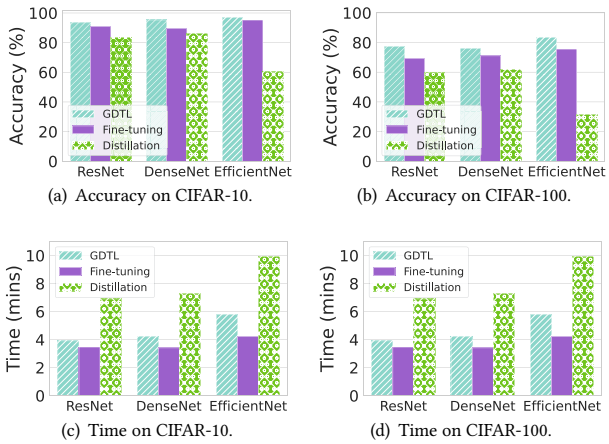
Distillation-based Fine-tuning:

$l \leftarrow n$ ▷ *Set the cut-layer to the last layer of F_V.*
$\tau \leftarrow$ One Hot Encoding of data label (y_i).
while $\left(l \geq 1 \;\&\&\; Acc(F_V) \leq \lambda * Acc(F_S)\right)$ **do**
  $\{\bullet_{j=1}^{l-1} F_{Vn}, \bullet_{j=l}^{n} F_{Vn}\} \longleftarrow$ ML-Split(F_V, l)
  for $(\forall (x_i, y_i) \in \mathbb{D}_T^{Priv})$ **do**
    $KDL\Big(F_S(x), F_V(x), \tau, T\Big) = -\alpha \mathbb{E}_{F_S(x)}\left[\log\left(\frac{F_V(x)}{T}\right)\right] - (1-\alpha)\mathbb{E}_{\tau}\left[\log\left(F_V(x)\right)\right]$
    Use stochastic gradient descent to update $\bullet_{j=l}^{n} F_V$ on: $KDL\Big(F_S(x), F_V(x), \tau\Big)$, where $gradients = gradients \times T^2$
  Unfreeze $(\bullet_{j=l}^{n} F_V)$ by adjusting the cut layer in F_V : $l \leftarrow l - 1$.
Quantize(F_V) ▷ *Dynamic-range quantization.*
Return $\langle F_S, F_V \rangle$ ▷ *Fully-trained Service model and fine-tuned distilled verification model.*

5.2 Generative Attack Detection and Re-classification Training

Based on the observations made in section 2 it is evident that the divergence between the service and verification models' probability distributions reveals statistical information, which is essential for attack detection. Thus, we utilize the differences in the relative divergence as features to assist attack detection.

For attack detection, a naive approach would be using the resultant divergence of the two probability distributions to set a fixed attack detection threshold. However, manually setting a threshold is not effective since *(i)* the threshold is model and data-specific, and *(ii)* the attack detection decision boundary is non-linear, for which a linear threshold will be ineffective. Supervised learning algorithms can effectively learn such non-linear decision boundaries and improve the detection accuracy. However, these algorithms will utilize known attack signatures and, hence, can become ineffective in detecting variations of known attacks or attacks outside the training dataset. As such, we used a GAN framework for training Fides' attack detection and re-classification models where no predefined attack signatures are used during the training phase, allowing the trained model to be more robust against unseen attack signatures.



(a) Accuracy on CIFAR-10. (b) Accuracy on CIFAR-100.
(c) Time on CIFAR-10. (d) Time on CIFAR-100.

Figure 6: The performance comparison between GDTL, distilling from scratch (Distillation), and fine-tuning (Transfer Learning) across multiple datasets and architectures. The results show the accuracy and per epoch time for all approaches showcasing how GDTL outperforms other approaches while being resource-efficient.

The correlation of the outputs of the service and verification models will result in five possible cases. Case *C1* is when both models agree with the ground truth. Case *C2* is when only the service model agrees with the ground truth while Case *C3* represents those instances where only the verification model agrees with the ground truth. For Case *C4* both models make two different incorrect predictions. In Case *C5*, both models make the same incorrect prediction, *i.e.,* disagree with the ground truth but happen to agree with each other. These cases are accounted for in our GAN framework to train the attack re-classification model. Note that the re-classification model is a five-class classifier, which returns one of these cases regardless of the number of classes in the service model.

GAN-based Attack Detection and Re-classification. We formalize the training of Fides' attack detection model ($\mathcal{D}$) and the attack generator model ($\mathcal{G}$) as a min-max game. The generator is an attack crafting neural network, which takes samples from the private dataset ($\mathbb{D}_T^{Priv}$) and it returns outputs that are different from the service model. The goal of the detection model, *i.e.,* the discriminator, is to differentiate between the output of the service model and the generator model's crafted outputs. The generative training process simultaneously trains a re-classification model ($\mathcal{R}$), which aims at correcting the attack's outcome by reclassifying the output of the service model when under attack.

In training the generator and detection models, we define the objective function of the min-max game as:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{x \sim p_{\text{data}}(x)}\left[\log\left(\mathcal{D}\big(F_S(x), F_V(x)\big)\right)\right] + \mathbb{E}_{x \sim p_{\text{data}}(x)}\left[\log\left(1 - \mathcal{D}\big(\mathcal{G}(x), F_V(x)\big)\right)\right] - \mathbb{E}_{x \sim p_{\text{data}}(x)}\left[\log\left(1 - \mathcal{G}(x)\right)\right].$$

The first and second terms are the cross-entropy between the output of the attack detection model and its true label $\mathcal{Y}_{\mathcal{D}}$ ($\in \{0,1\}$), where 1 signifies no attack while 0 signifies attack. The third term is the cross-entropy between the generator model's output and the private data's true label. Our proposed objective function is

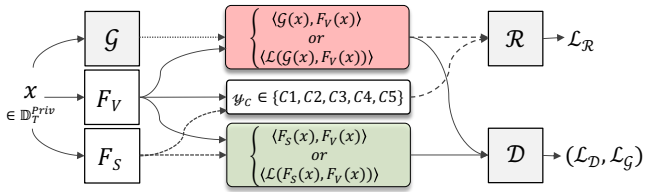


Figure 7: We formalized the attack detection and subsequent re-classification models as a GAN, in which the attack generator ($\mathcal{G}$) crafts various attacks to train the attack detection ($\mathcal{D}$) model in a two-player min-max game. In the process of training $\mathcal{G}$ and $\mathcal{D}$, the framework simultaneously trains a re-classification model ($\mathcal{R}$), aiming to correct the outcome of the attack.

different from the conventional generative adversarial networks in two aspects. First, the addition of the third term penalizes the generator model every time its prediction is the same as the true label. Thus, preventing the generator model from converging with the service model. Second, the generator model does not sample any input noise. Instead, it uses samples from the private dataset as input and crafts a malicious output with a wrong label, while mimicking the decision boundary of the service model.

Recall that the generator model takes the samples from the private dataset as input and generates malicious outputs. We define a malicious output to be a prediction that is different from the true label but has a probability distribution similar to a naturally occurring misclassification. To achieve this behavior, we define the generator model's loss function as:

$$\mathcal{L}_\mathcal{G} = -\log\Big(\mathcal{D}\big(\mathcal{G}(x), F_V(x)\big)\Big) - \log\big(1 - \mathcal{G}(x)\big).$$

The first term in $\mathcal{L}_\mathcal{G}$ rewards the generator model every time the detector model classifies the output of the generator model as a valid output. The second term rewards the generator model for making predictions that are different from the true labels.

The input of the detection model comprises two tuples (Figure 7). The first tuple consists of the outputs of the generator model $\mathcal{G}(x)$ and verification model $\big(F_V(x)\big)$ alongside their cross-entropy loss $\Big(\mathcal{L}\big(\mathcal{G}(x), F_V(x)\big)\Big)$. The second tuple consists of the outputs of the service model $\big(F_S(x)\big)$ and verification model $\big(F_V(x)\big)$ alongside their cross-entropy loss $\big(\mathcal{L}(F_S(x), F_V(x))\big)$. We define the loss function of the detection model as the binary cross-entropy between the model's output and its true label:

$$\mathcal{L}_\mathcal{D} = -\log\Big(\mathcal{D}\big(F_S(x), F_V(x)\big)\Big) - (1 - \mathcal{Y}_\mathcal{D}) * \log\Big(1 - \mathcal{D}\big(\mathcal{G}(x), F_V(x)\big)\Big).$$

The primary challenge in training the detection model is differentiating the naturally occurring prediction disagreement between the service and verification models and the disagreement caused by the attack. As mentioned earlier the trend we observed in our analysis regarding the difference in the divergence ranges between a natural misclassification and an attack allows the detection model to differentiate between the two (refer to Figure 3 in Section 2).

Per Figure 7, during the training process, the re-classification model receives the output of the generator model $\big(\mathcal{G}(x)\big)$ and verification model $\big(F_V(x)\big)$ with their cross-entropy loss $\big(\mathcal{L}\big(\mathcal{G}(x), F_V(x)\big)\big)$.

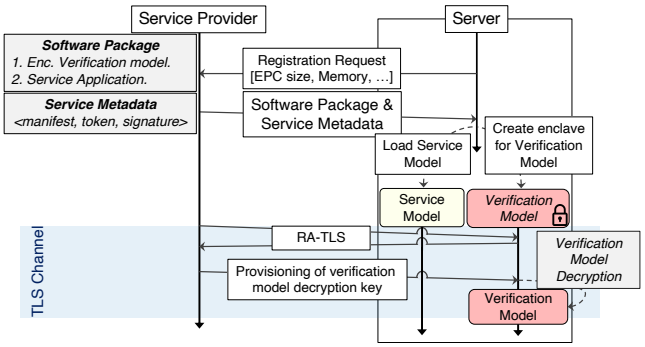


Figure 8: During the service deployment process, the service provider securely deploys the verification model on a secure enclave on the remote edge server and provisions the decryption key after remote attestation. Thus, guaranteeing the integrity of the verification model.

It also takes the compatibility label between the service and verification models, *i.e.,* $\mathcal{Y}_c$, which signifies the correctness of the outputs of these models and their similarity. Using these inputs, we defined the re-classification model's loss as the cross entropy between the output of the re-classification model and $\mathcal{Y}_c$ as:

$$\mathcal{L}_\mathcal{R} = -\sum_{c=1}^{5} \mathcal{Y}_c * \log\Big(\mathcal{R}\big(\mathcal{G}(x), F_V(x)\big)\Big),$$

where $\mathcal{Y}_c \in \{C1, C2, C3, C4, C5\}$.

After training the attack detection and re-classification models using the proposed GAN, the service provider shares them with the clients. Thus, allowing the clients to independently validate the results of the MLaaS inference task, executed by the edge server.

5.3 Service Deployment

Without loss of generality, we consider two possibilities for service deployment. In the first approach, the server initiates the deployment process by sending a request to the provider with requisite information about its resources, *e.g.,* EPC size, memory, bandwidth, and the service(s) it is willing to offer. The choice of the service depends on criteria like service demand or resource availability. Alternatively, the service provider initiates the deployment request to a potential server based on service demands in the server's locale. Per Figure 8, we adopt the first approach.

After the final agreement, the provider sends a complete application package to the server. The application package includes the service model, the encrypted verification model, and the service metadata. The service metadata contains application-specific configuration information, which is necessary for the integrity verification of the ML application. In particular, for Intel SGX, the metadata contains a `.manifest.sgx` file for configuring the secure application environment by the SGX SDK along with a `.token` and a `.sig` file, which SGX uses to verify the integrity of the enclave-loaded application and files. The server then initiates a secure enclave for loading the encrypted verification model. It then establishes an RA-TLS channel [28] with the service provider for provisioning the application decryption key into the secure enclave. Thus, allowing the enclave to decrypt the verification model.

The service provider shares requisite enclave information, such as ⟨`MR_ENCLAVE, MR_SIGNER, ISV_PROD, ISV_SVN`⟩ with the

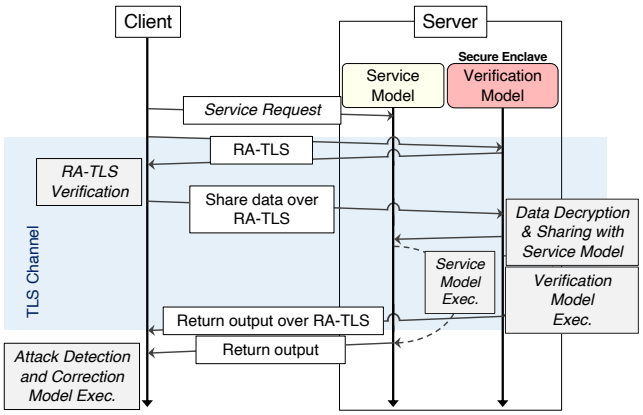


Figure 9: During service request, the client securely shares the data with the secure enclave running the verification model, which in turn shares it with the service model. Thus, ensuring that the verification model has access to trustworthy data.

clients, enabling them to verify the chain of trust during the remote attestation process (*i.e.,* RA-TLS) at the service outsourcing step. The provider generates these values while creating the SGX-supported application and shares them with the client when the application is downloaded. The RA-TLS protocol helps the client trust the output of the verification model running in the enclave.

5.4 Service Request and Execution

For service request, the client starts by discovering the servers that offer the service and have sufficient resources. If the service discovery fails, the client can either request the service from the provider or request a server to deploy the service. We do not discuss service discovery and assume the client finds a nearby server. Once the client has identified the server, it sends a service request to the server (Figure 9). Considering our threat model, a malicious server can change the data received from the client before sending it to the verification model for verification. Hence, the client first establishes an RA-TLS channel with the enclave to perform remote attestation. The remote attestation process on the client uses ⟨`MR_ENCLAVE, MR_SIGNER, ISV_PROD, ISV_SVN`⟩ information obtained from the provider to verify the integrity of the respective enclave in terms of its software and configuration. On successful remote attestation, *i.e.,* verifying the chain of trust and ensuring that the client is directly communicating with the secure enclave, the client securely shares the input data with the enclave. The application inside the enclave decrypts the client's data, shares it with the service model, and executes the verification model. Alternatively, the client can establish a TLS channel with the service model and an RA-TLS channel with the verification model to directly share the data with both. We use the first approach in our experiments.

After completing the ML inference task, the service and verification models return their outputs to the client. Note that the communication between the client and the secure enclave is protected through the established RA-TLS channel. The service model can either set up a secure connection to the client and securely share the result or use an insecure channel. The client then uses Fides' detection and re-classification pipeline on the outputs of the service and verification models to verify the correctness of executed ML inference task and detect potential attacks.

Table 1: Accuracy of service and verification models.

		ResNet	DenseNet	EfficientNet
CIFAR-10	Service	93.77%	94.93%	96.74%
	Verification	93.38%	95.30%	96.91%
CIFAR-100	Service	76.99%	78.69%	83.79%
	Verification	77.31%	76.13%	83.41%
ImageNet	Service	73.01%	73.32%	79.73%
	Verification	70.65%	69.74%	72.16%

6 EXPERIMENTS

In this Section, we review our benchmark datasets and models and elaborate on our system and experiment setup. We first analyze the similarity of the service and verification models pre- and post-attack and share our observations, which confirms the rationale behind our design. Finally, we assess the efficacy of Fides in detecting and re-classifying attacks, followed by system performance analysis.

6.1 Data and Models

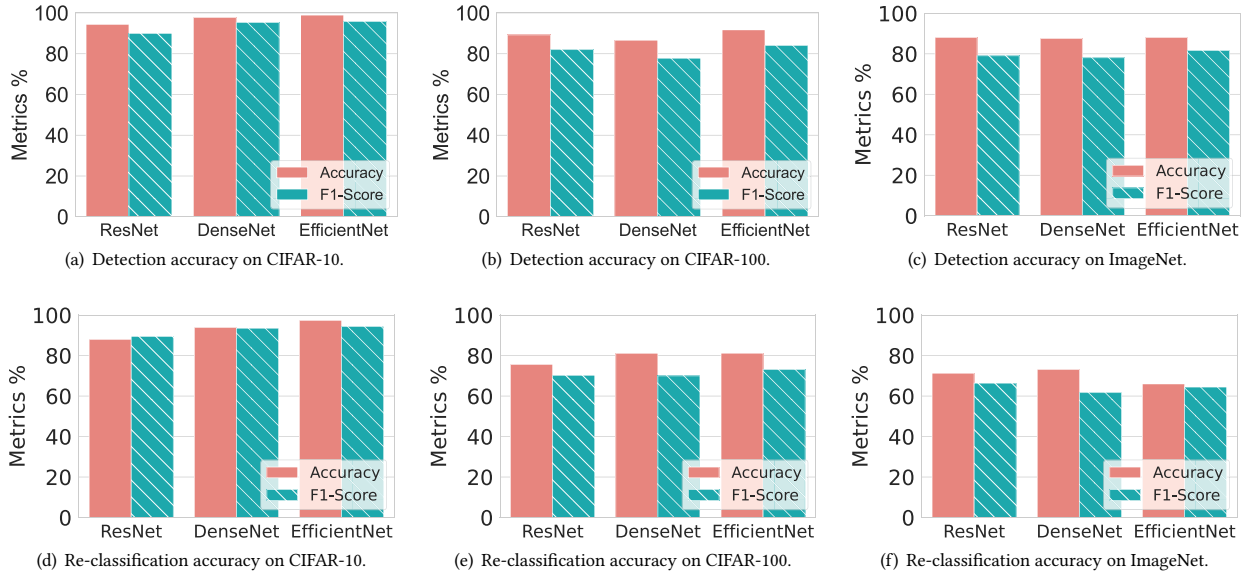
We evaluated Fides[1] using datasets of various complexities:

- CIFAR-10 consists of 60K color images (32×32 pixels) with 10 classes and 6K images per class. There are 50000 training and 10000 test images [30].
- CIFAR-100 features 100 classes with 600 images per class with pixels resolution of 32×32. There are 50000 training and 10000 test images [30].
- ImageNet-1K contains about 1.3M annotated images (about 1.28M training, 50K validation, and 100K test images), grouped into 1000 classes [10]. The resolution of images is 469×387 pixels; we cropped these images to 224×224 pixels.

We used three DNN architectures, namely ResNet, DenseNet, and EfficientNet for the image classification application. More specifically, we used ResNet-152 (60.4M parameters), DenseNet-201 (20.2M parameters), and EfficientNet-B7 (66.7M parameters) as the set of service models and ResNet-50 (25.6M parameters), DenseNet-121 (8.1M parameters), and EfficientNet-B0 (5.3M parameters) as their corresponding verification models, respectively. The accuracy of the corresponding service and verification models on the given datasets are tabulated in Table 1. Note that the accuracy of the models we used may not reach the reported accuracy in their respective papers. However, our goal is to use them in developing a result validation mechanism for MLaaS inference. While we used a similar architecture for each service and verification model pair, it is possible to use different architectures.

For our GAN framework, we adopted the fully trained service model (architecture and parameter weights) as the generator model and further extended it by adding four extra fully connected layers. The additional layers take the learned decision boundary of the service model at the beginning of the training process and fine-tune the generator model, aiming to craft stronger attack outputs. For the attack detection and re-classification models, we used two

[1]Fides Code is available on https://github.com/akumar2709/Fides_AsiaCCS



(a) Detection accuracy on CIFAR-10. (b) Detection accuracy on CIFAR-100. (c) Detection accuracy on ImageNet.

(d) Re-classification accuracy on CIFAR-10. (e) Re-classification accuracy on CIFAR-100. (f) Re-classification accuracy on ImageNet.

Figure 10: Fides' attack detection and re-classification models' performance across all datasets and architecture combinations. The results suggest a direct correlation between the accuracy of the service model and the accuracy of the detection and re-classification models.

lightweight fully connected neural networks, each containing three hidden layers containing 128, 256, and 128 neurons per hidden layer.

6.2 System and Experiments Setup

We evaluated our framework on a server with an Intel Xeon Platinum 8352Y processor with a base clock speed of 2.20 GHz and 256 GB RAM. The server's processor is a 3rd generation Xeon processor (*i.e.,* Ice Lake series) with 64 GB EPC. The SGX drivers are in-kernel drivers and were implemented using Gramine OS v1.2. We used three classes of consumer processors, namely a mobile class device with a Snapdragon 765G processor, an IoT class device with an ARM Cortex-A72 processor, and an X86 consumer class device with an AMD Ryzen 5 processor.

6.3 Verification Efficacy Analysis

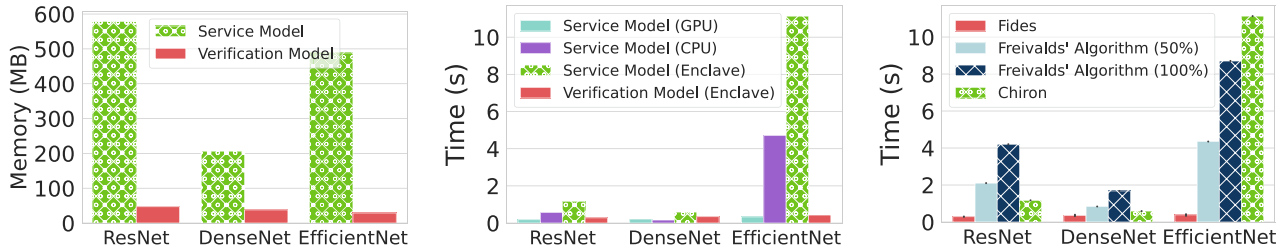
We evaluated the attack detection and re-classification model's performance in terms of accuracy and F1-score. In doing so, we built a test dataset by randomly selecting an average of 9000 samples from each dataset and applying the attacks we described in Section 4.3. We applied the attacks either on the service models' outputs of the 9000 samples (for naive and advanced attacks), input samples (for adversarial example attack), or the service model itself (for Trojan attack). We also included the output of the legitimate service models on the 9000 samples to the test dataset to represent the no-attack scenario. In our experiments, the accuracies we reported for the re-classification models are of the detected samples. Note that running the standalone re-classification models on all the generated attack samples (including the undetected ones) results in higher accuracies than those we discuss below.

For the attack detection and re-classification models, we devised two variants–one that uses the soft labels of the service and verification models as the input features and another one that uses the cross-entropy loss measurement as the input feature. Using the loss measurement enables the detection and re-classification models to utilize the divergence trends we discussed earlier while using soft labels allows the models to learn additional patterns, beyond the divergence trends.

Figure 10 represents the accuracy and F1-Score of the loss-based attack detection and re-classification models across all architectures and datasets. We measured the attack detection accuracy across all

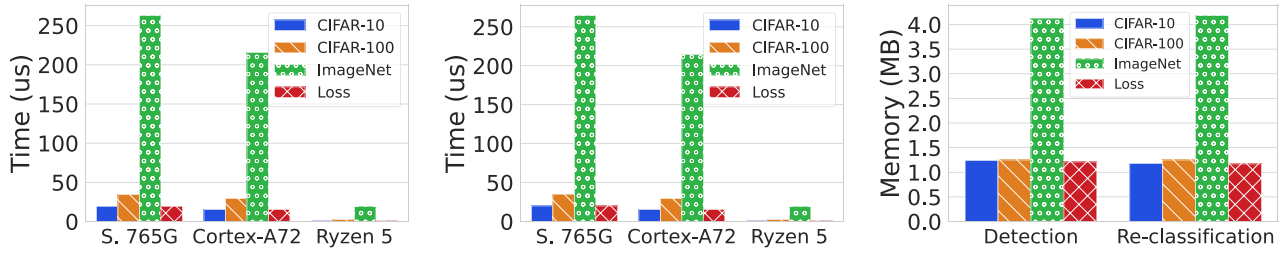
Table 2: Attack detection accuracy per attack type.

			ResNet	DenseNet	EfficientNet
CIFAR-10	Accuracy	Naive	95.07%	96.24%	97.36%
		Advanced	95.27%	96.36%	97.59%
		FGSM / PGD	89.25%	95.11%	95.89%
		Backdoor	95.23%	96.34%	97.57%
	F1-Score	Naive	94.84%	96.11%	97.30%
		Advanced	95.03%	96.22%	97.53%
		FGSM / PGD	89.39%	94.99%	95.86%
		Backdoor	94.99%	96.20%	97.53%
CIFAR-100	Accuracy	Naive	91.57%	89.87%	92.68%
		Advanced	86.62%	82.24%	90.18%
		FGSM / PGD	94.42%	95.63%	93.22%
		Backdoor	96.86%	97.14%	96.64%
	F1-Score	Naive	91.76%	90.32%	92.73%
		Advanced	87.52%	84.19%	90.48%
		FGSM / PGD	94.39%	95.58%	93.23%
		Backdoor	96.76%	97.07%	96.52%
ImageNet	Accuracy	Naive	84.36%	82.37%	83.56%
		Advanced	78.87%	76.08%	76.39%
		FGSM / PGD	87.94%	86.95%	85.31%
		Backdoor	89.42%	88.18%	88.22%
	F1-Score	Naive	83.60%	81.40%	82.50%
		Advanced	79.05%	76.34%	76.64%
		FGSM / PGD	75.53%	85.53%	84.06%
		Backdoor	88.29%	86.71%	81.25%



(a) Memory usage of service/verification models. (b) Execution time of service/verification models. (c) Execution time of verification techniques.

Figure 11: Evaluation of Fides' server-side performance in terms of memory and execution time on the server.



(a) Inference time of attack detection models. (b) Inference time of attack re-classification models. (c) Memory footprint of attack detection and re-classification models.

Figure 12: The number of input features for various datasets impacts the attack detection and re-classification model's performance. Using loss (cross-entropy) for attack detection and re-classification outperforms others due to the smaller feature size.

attack scenarios, *i.e.,* from the naive to advanced and well-known attacks. Table 2 tabulates the attack detection accuracy and F1-score for each attack type. These results show that the attack detection models on average achieve an accuracy of 91.15% across all architectures and datasets and the re-classification models achieve an average accuracy of 80%. More specifically, using EfficientNet for CIFAR-10 results in the best attack detection and re-classification performance (Figures 10(a) and 10(d)). However, increasing the dataset complexity, from CIFAR-10 to ImageNet, slightly reduced the attack detection accuracy with a more moderate impact on the re-classification accuracy (Figures 10(c) and 10(f)). Such behavior is expected as the accuracy of the service and verification models for ImageNet are lower than CIFAR-10 and CIFAR-100 accuracies across all architectures (Table 1).

For cases like ImageNet where the accuracy of the service and verification models are (on average) 75.24% and 70.85%, the attack detection accuracy is still higher than 87%, showing that the detection models were successfully identifying attacks where the service model and verification model disagree (the non-trivial scenarios). One can also observe that the performance of the detection and re-classification models are correlated to the service model's accuracy–the more accurate the service model, the more accurate the attack detection and re-classification models. Finally, we observed that both loss-based and soft labels-based variants of the attack detection and re-classification models achieve similar performance in terms of accuracy and F1-Score.

6.4 System Performance Analysis

We also ran a set of experiments to assess Fides's system performance in terms of execution time and memory (Figures 11 and 12). We used the ImageNet dataset in all architectures for assessing the memory consumption of the server-side components as ImageNet has the most number of classes–the worst-case scenario. Figure 11(a) shows that the GDTL process resulted in a smaller verification model with a significantly smaller memory footprint. This is crucial for deploying real applications on resource-constrained edge servers. We note that the compression ratio may vary across different models depending on the architecture. Figure 6 (Section 5.1) shows the accuracy and the time taken by GDTL when compared to fine tuning (transfer learning) and distillation from scratch. GDTL performed the best accounting for both parameters.

We compared the secure execution of the service and verification models with the insecure service model on CPU and GPU. Figure 11(b) demonstrates that the secure execution of the verification model for ResNet and EficientNet architectures takes far less than the secure and insecure execution of their service models. For DenseNet, the verification model only takes 0.178 seconds more than the insecure service model but it is still faster than the secure execution of the service model, which we attribute to the DenseNet smaller compression rate. The secure execution of the verification models is marginally slower than GPU executions (worst case 130 milliseconds for Densenet), which is expected.

We also compared the system performance of Fides with two other techniques–Slalom [65] and Chiron [27]. We selected Slalom as it outperforms other solutions in terms of validation of outsourced ML inference task and selected Chiron due to its similarity with Fides in running the entire model in a TEE. For Slalom, we only implemented its verification process, which uses Freivalds' algorithm to verify the matrix multiplication operations of each linear layer. Considering the probabilistic nature of the verification process in Slalom, we benchmark the verification of 50% and 100% of the service model's matrix multiplications per architecture. We

did not include the time of ECALL and OCALL operations (entering and exiting the enclave), which will result in a large latency overhead considering the frequency of these operations in Slalom. Per Figure 11(c), Fides outperforms other techniques in terms of validation time. It achieves a 1.73× to 25.7× speed-up compared to Chiron and a 4.8× to 26.4× speed-up compared to verification based on Freivalds' algorithm for 100% of the matrices. This is in part due to the sequential execution of the Freivalds' algorithm in our experiment. Note that the speed-up in both cases increases with the increase in the size of the verification model.

Given the constrained nature of clients' devices, we evaluated the performance of the attack detection and re-classification models on several consumer-grade device classes. Figures 12(a) and 12(b) show the elapsed time of running these models on an Internet of Things device (ARM Cortex-A72), a smartphone (Snapdragon 765G), and a personal desktop (AMD Ryzen 5). One can observe the negligible overhead of these models on clients; less than 260 microseconds in the worst case. Our analyses show that the performance of these models is highly correlated to the size of their input features when using soft labels as the input. For instance, detecting an attack on ImageNet with 1000 classes takes (on average) 13.47× longer than detecting the same attack on CIFAR-10 with 10 classes. The same behavior can be observed from attack re-classification models. However, the loss-based detection and re-classification models show significant improvement in the execution time–only 0.076% of the original latency in the case of ImageNet soft labels. This is primarily due to the independence of the model's input size to the number of the service model's classes when using cross-entropy loss value as the input feature. It is worth mentioning that the added overhead of loss calculation for the two input vectors is negligible. Figure 12(c) shows the memory footprint of different detection and re-classification models. The memory footprint falls within a similar range for CIFAR-10, CIFAR-100, and the loss-based solutions but increases significantly for ImageNet. It is primarily due to the smaller input layers for CIFAR-10, CIFAR-100, and loss-based solution, when compared with ImageNet's larger input layer.

7 RELATED WORK

The defenses against integrity violations in machine learning models during inference can be classified into three categories.

Cryptographic defences include solutions like multi-party computation [26, 29, 60, 72], proof-based systems [16, 34, 40], various constructions of homomorphic encryption [40, 47, 49, 69]. While the primary goal of multi-party computation is protecting data privacy, this technique offers a degree of verifiability. Similarly, solutions based on homomorphic encryption provide verifiability by virtue of protecting the data. For instance, one secure inference operation of the ResNet50 model using homomorphic encryption on customized hardware takes about 970 seconds compared to only 100 milliseconds for the same operation on non-encrypted data [56].

In contrast to these schemes, proof-based systems, *e.g.,* interactive and zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK), have been extensively used for verifiable ML [9, 16, 34, 38, 58, 68, 73]. The proof-based solutions, although theoretically more representative of verification, require significant computation by the prover [34] and do not scale for large ML models with many convolution layers [76]. For instance, generating proof for the VGG16 model with 16 layers and a decision tree with 23 levels take 88.3 seconds and 250 seconds, respectively [38, 73]. Moreover, the authors in [65] have shown that the best proof-based verifiable ML scheme is roughly 200 times slower when compared with running the entire model in a TEE.

TEE-based defenses has been used extensively in ML security and privacy, where confidentiality, privacy, and verifiability are paramount [11, 15, 21, 65]. In this domain, the main body of the literature aims at the efficient and secure execution of advanced models and large datasets in TEEs with small enclave page cache (EPC) [31, 43, 44]. These frameworks suggest partial outsourcing, in which a subset of the layers will be outsourced for secure execution. The authors in [31, 44] suggested running the last few layers in one or multiple parallel enclaves and the rest of the layers by the client. Alternative approaches proposed running the linear layers in a secure enclave and the remaining layers outside in insecure memory and verifying them using Freivalds' algorithm [65]

Attack Specific Defences could be designed to defend against backdoor attacks in the training phase [4, 48, 53, 67] by identifying and eliminating the updates or samples that are out-of-distribution. Since adversarial sample attacks take place in inference phase, the defenses against adversarial samples can take several different approaches, such as training discriminators to capture the difference between the training data and adversarial samples [13, 33, 79] or relying on invariance in the feature map and activation caused due to adversarial behavior [39, 42, 54].

Fides overcomes the shortcomings of these solutions by deploying a validation system, which relies on a significantly smaller verification model's posterior information to validate the prediction of service model. Fides aims to be highly parallelizable in deployment while being adaptable to already deployed MLaaS models. Fides also aims to make the defense attack methodology agnostic.

8 CONCLUSION

We introduced Fides–a framework for output verification of MLaaS inference. Fides features a Greedy model distillation technique. GDTL process gradually unfreezes the layers of an off-the-shelf model for distillation-based fine-tuning. Along with deploying the service model, the provider also securely deploys the verification model in a TEE on the server. The client then offloads the ML workload to the server by sharing its data with the service and verification models. Fides also features client-side neural networks for attack detection and re-classification, which are trained via our proposed GAN framework. Our rigorous evaluation using multiple datasets and neural network architectures shows Fides's superiority to the existing solutions in system performance–a 1.73× to 26.4× speed-up and achieves up to 98% attack detection and re-classification accuracy.

ACKNOWLEDGEMENTS

This research was partially funded by the US National Science Foundation under grants #1914635, #2148358, and #2133407, and the US Department of Energy grant #DE-SC0023392. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US federal agencies.

REFERENCES

[1] Tiago Alves. 2004. Trustzone: Integrated hardware and software security. *White paper* (2004).

[2] N Asokan, Jan-Erik Ekberg, Kari Kostiainen, Anand Rajan, Carlos Rozas, Ahmad-Reza Sadeghi, Steffen Schulz, and Christian Wachsmann. 2014. Mobile trusted computing. *Proc. IEEE* 102, 8 (2014), 1189–1206.

[3] Hessam Bagherinezhad, Maxwell Horton, Mohammad Rastegari, and Ali Farhadi. 2018. Label refinery: Improving imagenet classification through label progression. *arXiv preprint arXiv:1805.02641* (2018).

[4] Moses Charikar, Jacob Steinhardt, and Gregory Valiant. 2017. Learning from untrusted data. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. 47–60.

[5] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. 2017. Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning. arXiv:1712.05526 [cs.CR]

[6] Siyuan Cheng, Yingqi Liu, Shiqing Ma, and Xiangyu Zhang. 2021. Deep feature space trojan attack of neural networks by controlled detoxification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 1148–1156.

[7] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).

[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[9] Denise Demirel, Lucas Schabhüser, and Johannes Buchmann. 2017. Proof and Argument Based Verifiable Computing. In *Privately and Publicly Verifiable Computing Techniques*. Springer, 13–22.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *Conference on computer vision and pattern recognition*. IEEE, 248–255.

[11] Caiqin Dong, Jian Weng, Yao Tong, Jia-Nan Liu, Anjia Yang, Yudan Cheng, and Shun Hu. 2022. Fusion: Efficient and Secure Inference Resilient to Malicious Server and Curious Clients. *arXiv preprint arXiv:2205.03040* (2022).

[12] Sean Dougherty, Reza Tourani, Gaurav Panwar, Roopa Vishwanathan, Satyajayant Misra, and Srikathyayani Srikanteswara. 2021. APECS: A Distributed Access Control Framework for Pervasive Edge Computing Services. In *ACM SIGSAC Conference on Computer and Communications Security*. 1405–1420.

[13] Hasan Ferit Eniser, Maria Christakis, and Valentin Wüstholz. 2020. Raid: Randomized adversarial-input detection for neural networks. *arXiv preprint arXiv:2002.02776* (2020).

[14] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. 2018. Born again neural networks. In *International Conference on Machine Learning*. PMLR, 1607–1616.

[15] Akshay Gangal, Mengmei Ye, and Sheng Wei. 2020. Hybridtee: Secure mobile dnn execution using hybrid trusted execution environment. In *2020 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 1–6.

[16] Zahra Ghodsi, Tianyu Gu, and Siddharth Garg. 2017. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. *Advances in Neural Information Processing Systems* 30 (2017).

[17] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[18] Karan Grover, Shruti Tople, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. 2018. Privado: Practical and secure DNN inference with enclaves. *arXiv preprint arXiv:1810.00602* (2018).

[19] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[20] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[21] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. 2021. DarKnight: An accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 212–224.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[23] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).

[24] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv preprint arXiv:2010.02666* (2020).

[25] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[26] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. *IACR Cryptol. ePrint Arch.* 2022 (2022), 207.

[27] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961* (2018).

[28] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. 2018. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863* (2018).

[29] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* 34 (2021), 4961–4973.

[30] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[31] Abhinav Kumar, Reza Tourani, Mona Vij, and Srikathyayani Srikanteswara. 2022. SCLERA: A Framework for Privacy-Preserving MLaaS at the Pervasive Edge. In *International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events*. IEEE, 175–180.

[32] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. 2016. Adversarial examples in the physical world.

[33] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems* 31 (2018).

[34] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. 2020. vCNN: Verifiable convolutional neural network based on zk-SNARKs. *Cryptology ePrint Archive* (2020).

[35] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. 2021. When machine learning meets privacy: A survey and outlook. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–36.

[36] Qiang Liu, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor CM Leung. 2018. A survey on security threats and defensive techniques of machine learning: A data driven view. *IEEE access* 6 (2018), 12103–12117.

[37] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge computing for autonomous driving: Opportunities and challenges. *Proc. IEEE* 107, 8 (2019), 1697–1716.

[38] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. ZkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2968–2985.

[39] Jiajun Lu, Theerasit Issaranon, and David Forsyth. 2017. Safetynet: Detecting and rejecting adversarial examples robustly. In *Proceedings of the IEEE international conference on computer vision*. 446–454.

[40] Abbass Madi, Renaud Sirdey, and Oana Stan. 2020. Computing neural networks with homomorphic encryption and verifiable computing. In *International Conference on Applied Cryptography and Network Security*. Springer, 295–317.

[41] Javier Maroto, Guillermo Ortiz-Jiménez, and Pascal Frossard. 2022. On the benefits of knowledge distillation for adversarial robustness. *arXiv preprint arXiv:2203.07159* (2022).

[42] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. 2017. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267* (2017).

[43] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. 94–108.

[44] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. 161–174.

[45] Abdallah Moubayed, MohammadNoor Injadat, Abdallah Shami, and Hanan Lutfiyya. 2018. DNS Typo-Squatting Domain Detection: A Data Analytics and Machine Learning Based Approach. In *2018 IEEE Global Communications Conference (GLOBECOM)*. 1–7. https://doi.org/10.1109/GLOCOM.2018.8647679

[46] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 739–753.

[47] Deepika Natarajan, Wei Dai, and Ronald Dreslinski. 2021. CHEX-MIX: Combining Homomorphic Encryption with Trusted Execution Environments for Two-party Oblivious Inference in the Cloud. *Cryptology ePrint Archive* (2021).

[48] Thien Duc Nguyen, Phillip Rieger, Roberta De Viti, Huili Chen, Björn B Brandenburg, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, et al. 2022. {FLAME}: Taming backdoors in federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*. 1415–1432.

[49] Chaoyue Niu, Fan Wu, Shaojie Tang, Shuai Ma, and Guihai Chen. 2020. Toward verifiable and privacy preserving machine learning prediction. *IEEE Transactions on Dependable and Secure Computing* (2020).

[50] Guillermo Ortiz-Jiménez, Apostolos Modas, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. 2021. Optimism in the face of adversity: Understanding and improving deep learning through adversarial robustness. *Proc. IEEE* 109, 5 (2021),

635–659.

[51] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. 2016. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277* (2016).

[52] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2016. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 582–597.

[53] Neehar Peri, Neal Gupta, W Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P Dickerson. 2020. Deep k-nn defense against clean-label data poisoning attacks. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*. Springer, 55–70.

[54] Stefanos Pertigkiozoglou and Petros Maragos. 2018. Detecting adversarial examples in convolutional neural networks. *arXiv preprint arXiv:1812.03303* (2018).

[55] Partha Pratim Ray, Dinesh Dash, and Debashis De. 2019. Edge computing for Internet of Things: A survey, e-healthcare case study and future direction. *Journal of Network and Computer Applications* 140 (2019), 1–22.

[56] Brandon Reagen, Woo-Seok Choi, Yeongil Ko, Vincent T Lee, Hsien-Hsin S Lee, Gu-Yeon Wei, and David Brooks. 2021. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture*. IEEE, 26–39.

[57] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted Execution Environment: What It is, and What It is Not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. 57–64. https://doi.org/10.1109/Trustcom.2015.357

[58] Nitin Singh, Pankaj Dayama, and Vinayaka Pandit. 2021. Zero Knowledge Proofs towards Verifiable Decentralized AI Pipelines. *Cryptology ePrint Archive* (2021).

[59] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. 2021. A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects. *IEEE Communications Surveys & Tutorials* 23, 2 (2021), 1160–1192.

[60] Ekanut Sotthiwat, Liangli Zhen, Zengxiang Li, and Chi Zhang. 2021. Partially encrypted multi-party computation for federated learning. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 828–835.

[61] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[62] Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. 2020. An embarrassingly simple approach for trojan attack in deep neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 218–228.

[63] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. 2018. Attacks meet interpretability: Attribute-steered detection of adversarial samples. *Advances in Neural Information Processing Systems* 31 (2018).

[64] Reza Tourani, Srikathyayani Srikanteswara, Satyajayant Misra, Richard Chow, Lily Yang, Xiruo Liu, and Yi Zhang. 2020. Democratizing the Edge: A Pervasive Edge Computing Framework. *arXiv preprint arXiv:2007.00641* (2020).

[65] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287* (2018).

[66] Florian Tramèr, Reza Shokri, Ayrton San Joaquin, Hoang Le, Matthew Jagielski, Sanghyun Hong, and Nicholas Carlini. 2022. Truth serum: Poisoning machine learning models to reveal their secrets. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2779–2792.

[67] Brandon Tran, Jerry Li, and Aleksander Madry. 2018. Spectral signatures in backdoor attacks. *Advances in neural information processing systems* 31 (2018).

[68] Jiasi Weng, Jian Weng, Gui Tang, Anjia Yang, Ming Li, and Jia-Nan Liu. 2022. pvCNN: Privacy-Preserving and Verifiable Convolutional Neural Network Testing. *arXiv preprint arXiv:2201.09186* (2022).

[69] Guowen Xu, Hongwei Li, Hao Ren, Jianfei Sun, Shengmin Xu, Jianting Ning, Haomiao Yang, Kan Yang, and Robert H Deng. 2020. Secure and verifiable inference in deep neural networks. In *Annual Computer Security Applications Conference*. 784–797.

[70] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316.

[71] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. 2019. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2041–2055.

[72] Sen Yuan, Milan Shen, Ilya Mironov, and Anderson CA Nascimento. 2021. Practical, label private deep learning training based on secure multiparty computation and differential privacy. *Cryptology ePrint Archive* (2021).

[73] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. 2020. Zero knowledge proofs for decision tree predictions and accuracy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2039–2053.

[74] Qingyang Zhang, Quan Zhang, Weisong Shi, and Hong Zhong. 2018. Distributed collaborative execution on the edges and its application to amber alerts. *IEEE Internet of Things Journal* 5, 5 (2018), 3580–3593.

[75] Xinyang Zhang, Zheng Zhang, Shouling Ji, and Ting Wang. 2021. Trojaning language models for fun and profit. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 179–197.

[76] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. 2021. Veriml: Enabling integrity assurances and fair payments for machine learning as a service. *IEEE Transactions on Parallel and Distributed Systems* 32, 10 (2021), 2524–2540.

[77] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[78] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* (2016).

[79] Fei Zuo and Qiang Zeng. 2021. Exploiting the sensitivity of L2 adversarial examples to erase-and-restore. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. 40–51.

A COMPLEMENTARY RESULTS

Table 3: Entropy Analysis of Jeffreys Divergence (JD) and Wasserstein Metric (WM) for cases that service and verification models predictions are identical (Case A) post-attack.

			Case A		
			ResNet	DenseNet	EfficientNet
CIFAR-10	WM	Pre-attack	0.0038 ± 0.0037	0.0019 ± 0.0018	0.000013 ± 0.000012
		Post-attack	2.73 ± 0.76	2.99 ± 1.00	2.98 ± 0.99
	JD	Pre-attack	0.009 ± 0.009	0.004 ± 0.0001	0.000015 ± 0.000014
		Post-attack	13.09 ± 5.3	13.09 ± 4.37	14.08 ± 4.17
CIFAR-100	WM	Pre-attack	0.160 ± 0.157	0.153 ± 0.148	0.048 ± 0.047
		Post-attack	26.344 ± 13.344	25.99 ± 12.012	24.11 ± 12.12
	JD	Pre-attack	0.03 ± 0.0293	0.037 ± 0.035	0.0085 ± 0.008
		Post-attack	9.53 ± 3.98	10.25 ± 4.12	9.74 ± 4.07
ImageNet	WM	Pre-attack	1.25 ± 1.17	2.96 ± 2.64	38.79 ± 14.20
		Post-attack	51.96 ± 46.79	49.41 ± 43.26	85.13 ± 35.97
	JD	Pre-attack	0.035 ± 0.032	0.065 ± 0.06	0.216 ± 0.092
		Post-attack	3.96 ± 2.16	3.31 ± 1.82	2.70 ± 1.37

Table 4: Entropy Analysis of Jeffreys Divergence (JD) and Wasserstein Metric (WM) for cases that service and verification models predictions are different (Case B) post-attack.

			Case B		
			ResNet	DenseNet	EfficientNet
CIFAR-10	WM	Pre-attack	1.87 ± 0.63	2.039 ± 0.67	1.83 ± 0.62
		Post-attack	0.95 ± 0.57	0.99 ± 0.58	0.68 ± 0.47
	JD	Pre-attack	4.76 ± 1.99	4.83 ± 2.09	4.04 ± 1.55
		Post-attack	1.75 ± 1.36	1.64 ± 1.21	0.873 ± 0.66
CIFAR-100	WM	Pre-attack	23.89 ± 9.648	24.019 ± 9.49	22.53 ± 9.59
		Post-attack	17.11 ± 9.52	17.87 ± 9.62	16.54 ± 9.25
	JD	Pre-attack	4.77 ± 1.69	5.67 ± 0.035	4.87 ± 1.90
		Post-attack	3.67 ± 2.523	4.50 ± 2.99	4.12 ± 3.047
ImageNet	WM	Pre-attack	69.13 ± 34.87	66.13 ± 34.82	66.96 ± 26.09
		Post-attack	58.79 ± 33.12	55.05 ± 29.94	59.9 ± 22.93
	JD	Pre-attack	1.30 ± 0.56	1.20 ± 0.50	0.94 ± 0.35
		Post-attack	1.07 ± 0.58	1.03 ± 0.54	0.79 ± 0.34