GINN: Fast GPU-TEE Based Integrity for Neural Network Training

Aref Asvadishirehjini
axa159430@utdallas.edu
University of Texas at Dallas
USA

Murat Kantarcioglu
muratk@utdallas.edu
University of Texas at Dallas
USA

Bradley Malin
b.malin@vumc.org
Vanderbilt University Medical Center
USA

ABSTRACT

Machine learning models based on Deep Neural Networks (DNNs) are increasingly deployed in a wide variety of applications, ranging from self-driving cars to COVID-19 diagnosis. To support the computational power necessary to train a DNN, cloud environments with dedicated Graphical Processing Unit (GPU) hardware support have emerged as critical infrastructure. However, there are many integrity challenges associated with outsourcing the computation to use GPU power, due to its inherent lack of safeguards to ensure computational integrity. Various approaches have been developed to address these challenges, building on trusted execution environments (TEE). Yet, no existing approach scales up to support realistic integrity-preserving DNN model training for heavy workloads (e.g., deep architectures and millions of training examples) without sustaining a significant performance hit. To mitigate the running time difference between pure TEE (i.e., full integrity) and pure GPU (i.e., no integrity) , we combine random verification of selected computation steps with systematic adjustments of DNN hyperparameters (e.g., a narrow gradient clipping range), which limits the attacker's ability to shift the model parameters arbitrarily. Experimental analysis shows that the new approach can achieve a 2X to 20X performance improvement over a pure TEE-based solution while guaranteeing an extremely high probability of integrity (e.g., 0.999) with respect to state-of-the-art DNN backdoor attacks.

CCS CONCEPTS

• **Security and privacy** → *Hardware security implementation*; • **Computing methodologies** → *Classification and regression trees.*

KEYWORDS

Deep Learning, Trusted Exeution Environments, Intel SGX, Integrity Preserving Deep Learning Training

ACM Reference Format:
Aref Asvadishirehjini, Murat Kantarcioglu, and Bradley Malin. 2022. GINN: Fast GPU-TEE Based Integrity for Neural Network Training. In *Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy (CODASPY '22), April 24–27, 2022, Baltimore, MD, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3508398.3511503

CODASPY '22, April 24–27, 2022, Baltimore, MD, USA

ACM ISBN 978-1-4503-9220-4/22/04…$15.00
https://doi.org/10.1145/3508398.3511503

1 INTRODUCTION

Deep Learning (DL) is radically changing how machine learning interacts with, and supports, society. Numerous industries increasingly rely on DL models to make decisions, ranging from computer vision, natural language processing to high-quality synthetic data release [3]. One example is, Tesla that relies on DL for its *autopilot* mode of operation [7] and it seems that, in the near future, DL will be a routine technique for enabling autonomous vehicles to commute on roads. Another prominent application of DL is Covid-19 diagnosis [27], where wrong decisions can have irreversible consequences. The training process for these DL models requires a substantial quantity of computational resources (often in a distributed fashion) for training, which traditional CPUs are unable to fulfill. Hence, special hardware, with massive parallel computing capabilities, (e.g., GPUs, and TPUs) are often utilized. At the same time, the DL model training process is increasingly outsourced to the public cloud. This is natural, as applying cloud services (e.g., Amazon EC2, Microsoft Azure, or Google Cloud) for DL training can be more fiscally palatable for companies, while also enabling them to focus on the software product without worrying about the hardware maintenance and quality service level agreements of the hardware. Nevertheless, such outsourcing raises numerous concerns with respect to the privacy and integrity of the learned models. In recognition of the privacy and integrity concerns around DL (and machine learning (ML) in general), a considerable amount of research has been dedicated to applied cryptography, in three general areas: 1) *Multi-Party Computation (MPC)* (e.g., [25]), 2) *Homomorphic Encryption (HE)* (e.g., [10]), and 3) *Trusted Execution Environment (TEE)* (e.g., [15, 16]). However, the majority of these investigations are limited in that: 1) they are only applicable to simple shallow network models, 2) they are evaluated with datasets that have a few records (such as MNIST [21] and CIFAR10 [19]), and 3) they incur an amount of overhead that is unacceptable for real-life DL training workloads.

In an effort to mitigate some of these problems, and securely move from CPUs to GPUs, the *Slalom* [37] system was developed to focus on the computational integrity at the *test* phase while depending on the application context. It can also support enhanced data privacy. However, at a much greater performance cost. Similarly, *Goten* [26] introduced a privacy-preserving training and inference solution based on TEEs and GPUs. However, the empirical analysis suggested it is still not feasible to both enable privacy, and integrity with an acceptable performance comparable to the non-private setting (using GPU).

To address some of these limitations, we introduce **GINN** (See Figure 1); a framework for *integrity-preserving* learning as a service that provides integrity guarantees in outsourced DL model training

in TEEs. We assume that only the TEE running in the cloud is to be trusted, while all the other resources such as GPUs can be controlled by an attacker to launch an attack (e.g., insert a trojan). In this context, our goal is to support realistic DL training workloads while *ensuring data and model integrity*. To achieve this goal, we focus on the settings where maintaining the learning process's integrity is critical, while the training data may not contain privacy-sensitive information. For example, for a traffic sign detection model on public traffic sign images, it is desirable to prevent malicious behaviors that can put the pedestrians or the driver in harm's way. This is because safety is of paramount importance in the above scenario and trainers must ensure the integrity of the training process to avoid unintended consequences.

The trivial approach of executing the entire learning process inside a TEE is not scalable. This is mainly due to the fact that TEEs based on CPU extensions are substantially slower compared to GPUs. It should be noted that performance improvement techniques, such as random matrix verification [37]), have been proposed, but the gains achieved are insufficient to scale up to large DL model learning settings. To significantly improve the performance of pure TEE based approach (i.e., running entire DNN training inside the TEE), we introduce an approach that incorporates *randomized verification* into the DNN training process. This strategy is based on the observation that it is unnecessary to verify all computation steps of the GPU during the DNN training. Rather, we only need to occasionally verify to ensure a very high likelihood of catching any deviation.

Unfortunately, naive randomized verification of the DNN training steps may not be enough. Because, if the DNN training steps is not adjusted properly, even an attack in one step could have a devastating impact. For example, DNN training usually require stochastic gradient descent (SGD) based updates using the current batch of the data. If there are no limits on the SGD update step, the attacker can arbitrarily modify the model even in a single update. Given that randomized verification may itself be insufficient, we further show how parts of the DNN hyperparameter setting process, such as *clipping rate* should be modified to prevent single step attacks, and require a larger number of malicious updates by an attacker that controls the GPU. In other words, **GINN** limits the amount of change an adversary can inflict on a model through a single SGD update step. As a consequence, the adversary is forced to keep attacking while, randomly, being verified by the TEE. Using state-of-the-art backdoor attacks, we illustrate that a randomized verification technique can detect attacks with a high probability (e.g., 0.999) while enabling 2x-20x performance gains compared to pure TEE based solutions.

The specific contributions of this paper are as follows:

- We introduce the first approach to support integrity-preserving DNN training by randomized verification of stochastic gradient (SGD) steps inside TEE. This approach has an extremely high probability of ensuring the integrity the DNN training.
- We illustrate how gradient clipping can be used as a defensive measure against single (or infrequent) step attack in combination with randomized verification.
- We show the effectiveness of our TEE randomized verification and gradient clipping through extensive experimentation on DNN backdoor attacks.

2 BACKGROUND AND RELATED WORKS

Our system combines DNN training on specialized fast hardware such as GPUs with TEEs based on Intel Software Guard Extensions (SGX) to ensure the produced model's integrity. Appendix A provides a legend of the notation used in this paper.

2.1 Intel SGX

SGX [6] is an example of a common TEE that is available in many modern-day computers and *existing cloud infrastructure such as Microsoft Confidential Computing Cloud* [1]. As outlined in Table 3, it provides a secluded hardware reserved area, namely, processor reserved memory (PRM), that is kept private (i.e., it is not readable in plaintext) from the host, or any privileged processes, and is free from *direct* undetected tampering. It also supports *remote attestation*, such that users can attest the running code within the enclave before provisioning their secrets to a remote server. Calls from routines that should transition to/from enclave are handled through predefined entry points that are called Ecall/Ocall that must be defined in advance, before building the enclave image. While it provides security and privacy for applications, directly running unmodified applications inside SGX can induce a significant hit on performance because the memory and computational capacity are limited.

2.2 Deep Learning Training

Over the past decade, Deep Neural Networks (DNN) have become popular for solving problems related to computer vision and natural language processing [14, 20, 31, 35, 36]. In practice, these networks are stacks of layers, each of which perform a transformation $\mathcal{F}^l_{\mathcal{W}}(\cdot)\ \forall l \in |L|$ where $\mathcal{X}^{l+1} = \mathcal{F}^l_{\mathcal{W}}(\mathcal{X}^l)$ and $|L|$ is the number of layers. The training task is to learn the correct parameters (point-estimates) $\mathcal{W}^*$ that optimizes (commonly minimizes) a task-specific (e.g., classification) loss function $\mathcal{L}$. The most common approach for training a DNN is mini-batch Stochastic Gradient Descent (SGD) [29]. A randomly selected mini-batch of a dataset is fed to the DNN and the value of objective function $\mathcal{L}$ is calculated. This is usually referred to as the *forward* pass. Next, to derive the partial gradients of $\mathcal{L}$ with respect to $\mathcal{W}$ ($\nabla^{\mathcal{L}}_{\mathcal{W}}$), a *backward* pass is performed [12]. Finally, the parameters are then updated according to Equation 1, where $0 < \alpha < 1$ is referred to as the learning rate. Depending on the complexity of the dataset and the task, this process might require hundreds of passes (called *epoch*) over the input dataset to achieve convergence.

$$\mathcal{W}^{t+1} = \mathcal{W}^t - \alpha \nabla^{\mathcal{L}^t}_{\mathcal{W}^t} \tag{1}$$

2.3 Gradient Clipping

Gradient Clipping (GC) is a method that has been shown to help mitigate the problem of exploding gradients during training [11]. Simply, GC forces the gradients into a narrow interval to prevent very large updates during the SGD step. There have been some efforts to analyze GC with respect to convergence. For instance,

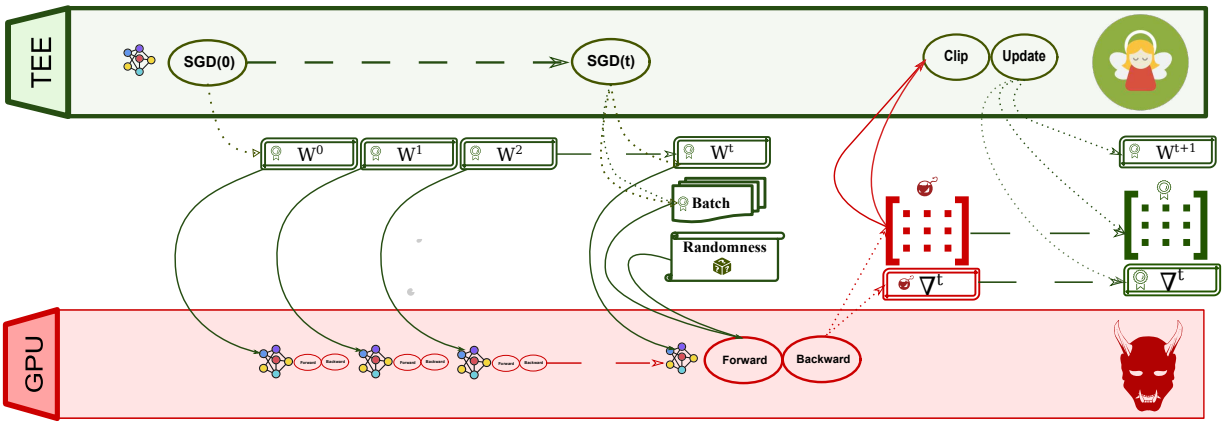


Figure 1: The main architecture of GINN. The TEE handles mini-batch selection, layer-specific randomness, and parameter initialization. The GPU performs forward and backward passes over the mini-batch (items selected by SGX provided seed) and reports the computed gradients to the TEE. TEE then clips the gradients and performs the weight update. Also, TEE preserves the MAC-authenticated intermediate gradient reports. During verification, TEE performs the forward and backward passes with the batch items along with layer-specific randomness (regenerated) and compares the gradients with the GPU's report.

Zhang and colleagues [39] prove (assuming a fixed step size) that training with GC can be faster than training without it. Moreover, their theoretical analysis suggests that too small clipping values can reduce the training performance (i.e., require more steps for convergence). However, in practice, this performance reduction is rarely observed. [5] has an interesting theoretical analysis coupled with empirical evidence (symmetry of gradients distribution with respect to the SGD trajectory) that answers the gap between previous theoretical and practical observations. These results suggest that GC could be leveraged in practical DNN training without significant performance issues.

2.4 Attacks on DNN Models in Training Phase

Attacks on DNN models can be realized during both *training* or *test* phases. However, GINN is concerned with integrity issues during the training phase of DNN models, such that attacks related to testing are out of the scope of this paper since test time attacks have been addressed before (e.g., *Slalom* [37]). In the literature, particularly in the computer vision community, targeted trojan attacks on DNN classification models have become a real concern as deep learning has grown in its adoption. These attacks tend to alter the prediction of models when a specific condition in the input is met. These conditions may be *feature-based* [13, 23] or *instance-based* [30]. Recently, trojan attacks were extended to Reinforcement Learning (RL) and text classification models [18, 34].

In practice, these attacks are implemented by manipulating samples during training through data poisoning. This could be achieved, for instance, by stamping images with a certain pattern and modifying the label of the image (e.g., changing "dog" to "cat"). Notably, these models provide similar competitive classification test accuracy when compared to clean models (i.e., models that have not been attacked). As a consequence, it is non-trivial to distinguish trojaned models from non-trojaned models based on model accuracy alone. To make matters worse, even if the model owner was aware of examples of the trojan trigger pattern, the owner would need to patch the model through a retraining process to dampen the efficacy of the trojan trigger pattern. Retraining does not always guarantee complete removal of the trojan behavior from the model. Various techniques have been proposed to diagnose and mitigate the effect of trojaned models. However, all approaches developed to date are either 1) based on unrealistic assumptions or 2) are excessively costly. For instance, Neural Cleanse [38] requires access to a sizable sample of clean inputs to reverse-engineer the backdoor and has shown to be successful only for trigger patterns with a relatively small size. ABS [22] improves upon Neural Cleanse in that requires a significantly smaller number of samples; however, it assumes that the neurons responsible for the trojan can activate trojan behavior independently of each other, which is unlikely to be true in practice.

2.5 Integrity Protection for DNN Training

[37] took the first step towards achieving both *fast* and *reliable* execution in the *test* phase, but neglected the training phase. The training phase is far more computationally demanding than the test phase, such that verification of all steps in training requires a substantially longer time because 1) parameters continuously changing, and 2) the backward pass involves computing gradients for both the inputs and the parameters, which requires a larger amount of time than the forward pass alone. Moreover, attacking the training pipeline to inject a trojan in the final model is quite simple (i.e., replace the actual model with the desired attack model by modifying the last SGD update) and, thus, is likely the most desirable form of attack for real world adversaries to launch. Altogether, throughout this work, we mainly focus on showing the effectiveness of our method at preventing this type of attack from being successful. The main objective of GINN is to enable a high-integrity training pipeline so that the users are assured that the model 1) is built on the correct dataset and 2) uses the correct parameters

without modification. If, at any point during the training, GINN detects a deviation from the specified execution, it will not approve the final model to ascertain its validity.

GINN relies upon the *proactive* training as opposed to the post-training or deployment-time methods to protect the health of a DNN model. It should be noted that our approach is *independent of the attack strategy* and is sufficiently *generic* to catch any continuous attack during the training of a DNN model.

GINN limits the amount of change an adversary can inflict on a model through a single SGD step. As a consequence, the adversary is forced to keep attacking while being verified at random by the TEE.

3 THREAT MODEL

Attacks on the integrity of DNNs can be orchestrated at different stages of the model learning pipeline (e.g., data collection or training). We assume the TEE node in GINN is trusted and the bytes stored on the *Processor Reserved Memory* (PRM) are always encrypted and authenticated before they are fetched inside the CPU. We assume that the data sent to GINN is provided by honest users via a secure/authenticated channel and is devoid of malicious samples.[1] For the training phase, we assume that the adversary has complete knowledge about the network structure, learning algorithm, and inputs (after TEE performs an initial pre-processing) to the model. In our threat model, the adversary is in complete control of the host system's software stack, and hardware (unprotected RAM, GPU), except for the CPU package and its internals. Therefore, the code that runs inside the enclave is free from tampering, and the data that are accessed inside the cache-lines or registers are not accessible to the adversary. For the inputs supplied to DNN tasks, the adversary is capable of performing insertion, modification, and deletion to influence the final model towards her advantage. Finally, the adversary controls the communication between TEE and the user, but cannot impose denial of service attacks. As a result, an attacker may report falsified gradients at any time. Nonetheless, to enable protection against rollback attacks, readers can refer to [24].

4 SYSTEM DESIGN

GINN offers integrity for the training phase of a DNN model while inducing limited computational overhead. An overview of GINN is illustrated in Figure 1.

Training Setup Before the training phase initiates, the training dataset is decrypted and validated inside the TEE. We assume an honest and authenticated user will send her data encryption key K_{client} (after remote-attestation [4]) to the TEE. Next, the TEE decrypts/verifies the initial encrypted dataset using the K_{client} and supplies the trainer (GPU) the plain-text of the training set. Lastly, the TEE allocates the necessary resources for the model and initializes the parameters with random values for the GPU to initiate the first SGD step.

Training with GINN 1) At the beginning of mini-batch SGD step $\mathbf{i}$ (nominal), the TEE supplies the untrusted GPU with the pseudorandom number generator (PRNG) seeds for the mini-batch selection and the per-layer PRNG seeds (e.g., Dropout [32]) that are derived within the TEE and supplied to the GPU. As a result, the adversary will have to use the provided PRNGs for random choices. TEE generated randomness is applied to populate the data buffers of the GPU with the correct batch. Depending on the number of layers, other PRNG seeds will be generated for each layer to generate random values for operations of each layer. 2) After completion of the forward and backward passes over the mini-batch, the computed gradients are sent back to the TEE. In our design, the GPU always performs the forward and the backward passes and reports the computed gradients to the TEE. 3) GINN always clips the reported gradients and ensures that they are within a narrow range before performing the update *so that evolving the model towards the attacker's intended model requires a prolonged malicious intervention by the attacker.* 4) GINN updates the parameters ($O(N)$ complexity) with the clipped gradients and saves authenticated snapshots of the state outside the TEE. 5) If the computation at this step is randomly selected for verification, then the faulty behavior, if any exists, can be detected. Otherwise, the chance that the model has evolved towards the attacker's desired optima likely requires multiple rounds, which provides ample opportunities for detection. Additionally, the verification is performed randomly to prevent the attacker from guessing which step is likely to be verified. In the end, if the TEE does not detect any violation, it will certify the final model (Appendix B) by digitally signing the model hash.

Probabilistic Verification with GINN The TEE randomly decides whether or not to verify the computation over each mini-batch. If the mini-batch is selected for verification, then the intermediate results are saved and the verification task is pushed into a verification queue. *Verification by the TEE can take place asynchronously* and it does not halt the computation for future iterations on the GPU. The authenticity of snapshots is always verified with a key that is derived from a combination of the TEE's session key, $SK_{SGX}^{session}$ and the corresponding iteration. When the TEE verifies the step $\mathbf{i}$, it populates the network parameters with the snapshot it created for the step $\mathbf{i-1}$. It then regenerates the randomness designated to step i to obtain the batch indices and correctly sets up the per-layer randomness. Given that the TEE's goal is to verify that the reported gradients for step i are correctly computed, GINN does not keep track of the activation results. Rather, it only requires the computed gradients, batch mean/std (for BatchNorm layer), and the matrix multiplication(MM) outcomes (in case random MM verification is chosen).

Randomized Matrix Multiplication Verification with GINN Matrix Multiplications (MMs) take up the bulk of the resource-heavy computations in DNNs. In modern DNN frameworks, convolutional and connected layers computation are implemented in the form of a matrix multiplication in both of the forward and backward passes. Table 4 in Appendix C depicts the computations in the forward pass and backward gradient with respect to the weights and previous layers' outputs in the form of a rank 2 tensor multiplication. Fortunately, there exists an efficient verification algorithm (i.e., Freivalds's randomized MM verification algorithm) for matrix multiplication([9]) when the elements of matrices belong to a field. In this work, we leverage the Freivalds's randomized MM verification algorithms as well.

[1] Detecting malicious samples is beyond the scope of this work.

5 INTEGRITY ANALYSIS

To achieve our integrity goal p_i (i.e., the probability that an attacker can modify the result without being detected is less than $1 - p_i$), we need to derive the p_v (i.e., the probability that TEE verifies an SGD step).

5.1 Random Mini-Batch Verification

We define the total number of DNN training steps as B. For each step, report R_b ($\forall b \in [1, B] \wedge R_b \in \{0, 1\}$) has a probability p_c for being corrupted (i.e., $R_b = 1$) and the overall integrity probability goal of p_i (for example $p_i = 0.999$).

We define $V_b = 1$ if a batch is chosen by the TEE for verification, and the verification result indicates a malicious SGD step. If the verification passes, then $V_b = 0$.

We define random variable $X = \sum_{b=1}^{\lceil B \times p_v \rceil} V_b$ to be the total number of random verifications performed that resulted in catching a malicious SGD step. It should be recognized that the TEE needs to catch at least one deviation (i.e., $X \geq 1$) with probability greater than p_i to invalidate the overall model learning.

Theorem 1. *Given a total of B steps during SGD training, a P_c probability of an SGD step being malicious, and P_i probability of detecting at least one malicious SGD step, the required probability of choosing a step to verify (p_v) should be greater than* $B^{-1}(\frac{\log(1-p_i)}{\log(1-p_c)} - 1)$.

Proof. We need to ensure that $P(X \geq 1)$ (i.e., the probability that at least one deviation is caught) is greater or equal to p_i.

$$\begin{aligned} P(X \geq 1) = 1 - P(X = 0) &\geq p_i \\ 1 - p_i &\geq P(X = 0) \\ 1 - p_i &\geq \binom{\lceil B \times p_v \rceil}{0} p_c^0 (1 - p_c)^{\lceil B \times p_v \rceil} \\ 1 - p_i &\geq (1 - p_c)^{\lceil B \times p_v \rceil} \\ \log(1 - p_i) &\geq \lceil B \times p_v \rceil \log(1 - p_c) \\ p_v &> B^{-1}(\frac{\log(1 - p_i)}{\log(1 - p_c)} - 1) \end{aligned} \quad (2)$$

□

As shown in Figure 8 in Appendix D, we only need to verify a small subset of the batch computations inside the TEE to ensure a high probability of correct computation. For example, for large datasets such as Imagenet [8], when corruption probability is %0.5, we need to randomly verify %1 of computation to achieve *0.9999* correctness probability on the computation outsourced to the GPU .

5.2 Random Mini-Batch Verification with Randomized Matrix Multiplication

Since Freivalds's randomized matrix multiplication verification scheme [9] is a randomized algorithm, it is possible that the scheme will falsely attest to the validity of the MM operations performed by the GPU. Thus, in addition to the previous configuration, the verification can be replicated k times to decrease the chance of encountering a false negative. The scheme has no false positive, so if the matrix multiplication is correct then the verification succeeds. Given that each SGD step contains m independent MM operations,

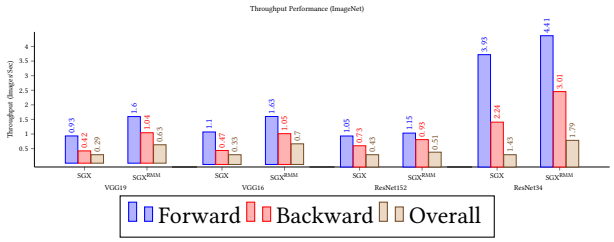


Figure 2: Throughput of the SGD training step for VGG19,VGG16, ResNet152, and Resnet34 on ImageNet dataset with respect to forward and backward passes. "SGX$^{\text{RMM}}$" refers to Freivalds' MM verification scheme, while "SGX" refers to the baseline case of fully computing the MM operation. RMM can lead to verification that is twice as fast as full MM verification in case of a VGG architecture.

which are to be repeated k times (independently), and random values are uniformly sampled from a field of size $|S|$, the probability of error (attesting to the validity of a malicious MM operation) is less than $\alpha = \frac{1}{|S|^{mk}}$. We define the random variable $V'_b = 1$ if $R_b = 1$ (i.e., corrupt report of a malicious step) and $MM\_verify(b) = 0$ (Freivalds's verification rejects the equality), otherwise $V'_b = 0$

Also, define $X = \sum_{b=1}^{\lceil B \times p_v \rceil} V'_b$. We need to detect at least one deviation with probability greater than p_i while conducting random matrix multiplication verification.

Theorem 2. *If random matrix multiplication verification is applied, then, given the configuration of Theorem 1, the required probability of choosing a step to verify (p_v) should be greater than* $B^{-1}(\frac{\log(1-p_i)}{\log((1+(\alpha-1)p_c)} - 1)$.

Proof. Again, we need to ensure that $P(X \geq 1)$ (i.e., the probability that at least one deviation is caught) is greater or equal to p_i.

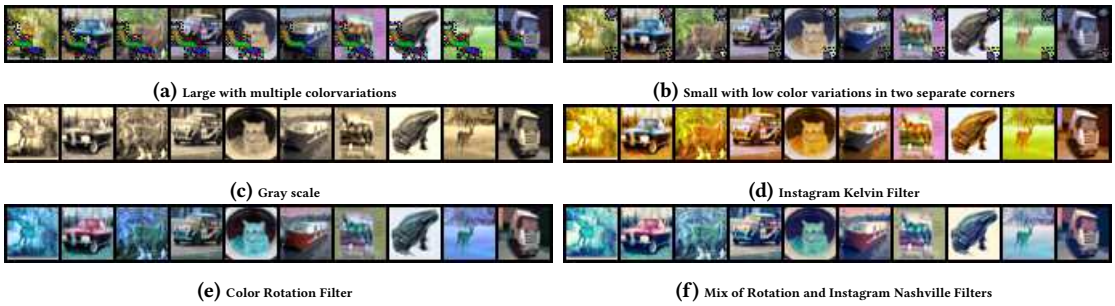
$$\begin{aligned} P(X \geq 1) &\geq p_i \\ 1 - P(X = 0) &\geq p_i \\ 1 - p_i &\geq \binom{\lceil B \times p_v \rceil}{0} (p_c(1-\alpha))^0 ((1 - p_c) + p_c\alpha)^{\lceil B \times p_v \rceil} \\ 1 - p_i &\geq ((1 - p_c) + p_c\alpha)^{\lceil B \times p_v \rceil} \\ \log(1 - p_i) &\geq \lceil B \times p_v \rceil \log((1 - p_c) + p_c\alpha) \\ p_v &> B^{-1}(\frac{\log(1 - p_i)}{\log((1 + (\alpha - 1)p_c)} - 1) \end{aligned} \quad (3)$$

□

The threshold in Theorem 2 yields approximately the same values as Theorem 1 when $\alpha \to 0$. However, the randomized MM verification requires a $O(N^2)$ operations for a $N \times N$ matrix, compared to an iterative algorithm that requires $O(N^3)$ operations.

6 EXPERIMENTAL EVALUATION

All of our experiments were run on a server with a Linux OS, Intel Xeon CPU E3-1275 v6@3.80GHz, 64GB of RAM and an NVIDIA Quadro P5000 GPU with 16GB of memory. Our attack code is implemented in python 3.6 using the PyTorch library. We use Intel



(a) Large with multiple colorvariations (b) Small with low color variations in two separate corners

(c) Gray scale (d) Instagram Kelvin Filter

(e) Color Rotation Filter (f) Mix of Rotation and Instagram Nashville Filters

Figure 3: All examples of triggers on CIFAR10 images

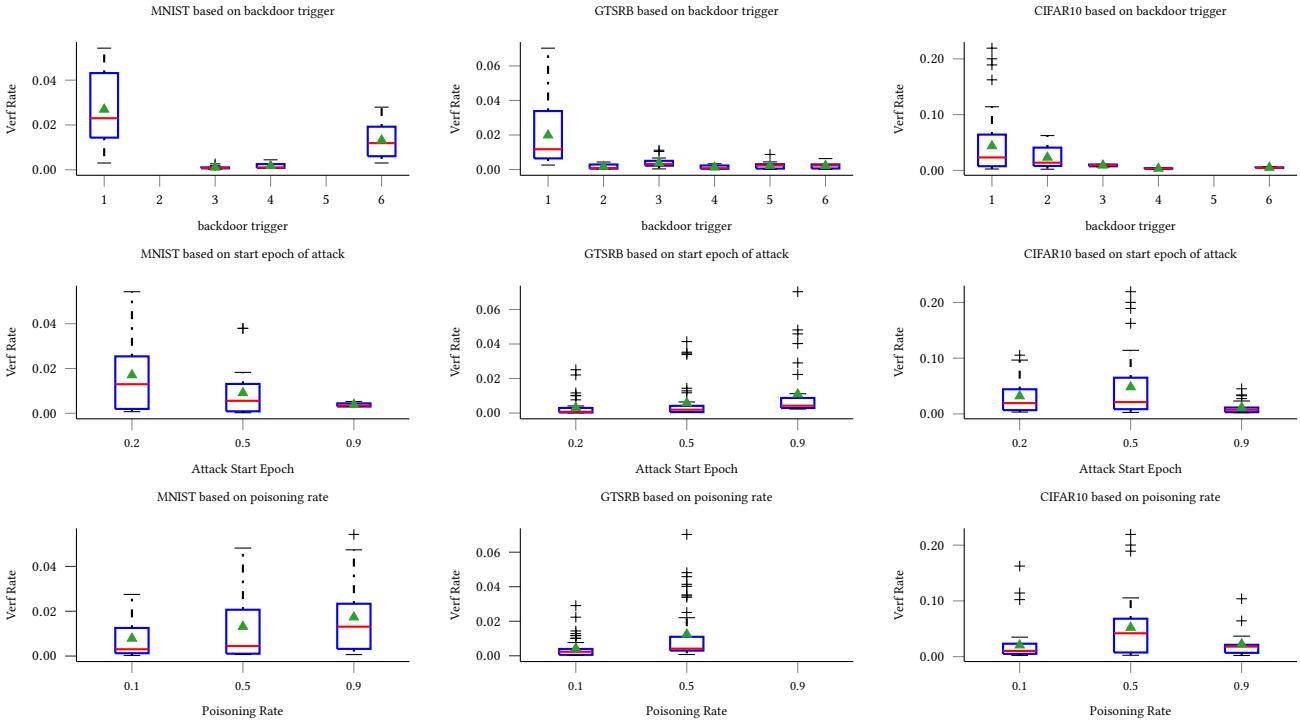


Figure 4: A boxplot representation of the verification rates required by TEE for detection failure $\leq 10^{-3}$. Based on the attack hyper-parameters: 1) backdoor trigger type (first row). 2) epoch that attack starts (second row, 20%×,50%×, and 90%×|epochs|). 3) backdoor poisoning ratio

Table 1: Trained models with restricted clipping and learning rate

Dataset	clip	lr	%clean	%attack	total
GTSRB	10^{-4}	10^{-4}	$\geq$ %95	$\geq$ %85	124
MNIST	10^{-4}	5×10^{-5}	$\geq$ %95	$\geq$ %85	49
CIFAR10	5×10^{-4}	10^{-1}	$\geq$ %90	$\geq$ %85	94

SGX for the TEE plaform. For SGX proof-of-concept implementation, we significantly modified the DarkNet [28] library to run the experiments.

Table 2: ImageNet (Re-)Training on ResNet34's Last Five Layers with *epochs* = 30, *epoch_attack* = 3, *pois_rate* = 0.5

#class targets	GC	clean accr	trojan accr	# steps	verf. rate
10	5	0.71	1.0	280	0.024
10	2	0.71	1.0	280	0.024
10	0.0005	0.57	0.99	2540	0.0027
50	5	0.71	0.93	181	0.038
50	2	0.71	0.93	181	0.038
50	0.0005	0.57	0.92	1380	0.005

Our SGX code has been tested with SDK 2.9 and the code runs inside a Docker container in hardware mode. Our experiments are

designed to investigate efficiency and effectiveness. First, we evaluate the degree to which integrating randomized matrix verification influences the computational efficiency of the process. Second, we analyze the effectiveness of gradient clipping in forcing the attacker to deviate from the honest protocol in higher number of mini-batch steps. Various attack hyper-parameters (e.g., poisoning rate) were evaluated in determining their importance towards a successful attack with minimal deviation. This is important because, if the attacker needs to deviate in more mini-batch steps, then the TEE can detect such deviations with a smaller number of random verification steps (i.e., p_c is higher in equation 2).

The following is an enumeration of experiments in the order they are discussed. In section 6.1, we analyze the performance of our system for a large dataset. For this experiment we use ImageNet [8], which consists of RGB images of 1000 categories. Next in section 6.2 we analyze the performance for a much smaller dataset, CIFAR10 [19]. CIFAR10 dataset contains RGB images of 32 by 32 pixels in 10 categories of objects or animals. Then in section 6.3 we analyze the impact of the heap allocated to the enclave. For this experiment we used images from ImageNet dataset. In section 6.4 we seek to evaluate the impact of gradient clipping, in an attack scenario for three different datasets and contemplate about the potential verification rate necessary for a TEE to catch the malicious execution. CIFAR10, MNIST [21] (black-white digit images from 0 to 9), and GTSRB [33] (RGB images of traffic signs) are used to conduct this analysis. Afterwards, in section 6.5, we investigate whether training with gradient clippings can have an unusually high impact on the quality of the models, especially if the chance of the training being attacked is not high. Finally, in section 6.6, we conducted a similar analysis on ImageNet (an instance of a very large dataset) to observe the potential impact of gradient clipping on big datasets.

6.1 TEE Performance On ImageNet With Common Architectures

We tested TEE performance on popular DNN architectures such as VGG16, VGG19 ([31]), ResNet152, and ResNet34 ([14]) with the ImageNet ([8]) dataset. Figure 2 illustrates the throughput of the deep networks (i.e., the number of images processed per second during the DNN training). Usually, most of the computation takes place in the convolution and fully-connected layers of the network. However, the backward pass yields a smaller throughput, on average, because it involves one more MM (weight gradients and input gradients) than the forward pass, which only invokes MM once (i.e., output of convolution or fully-connected layers). The implementation is quite efficient in terms of MM operations, which uses both vectorized instructions along with multi-threading. We note that the baseline GPU, which lacks TEE support and integrity protection, significantly outperforms the pure TEE baseline. For instance, the overall throughput for ResNet34 is about 60 images per second, whereas the pure SGX baseline solution is about 1.5 images per second.

Both VGG networks gives better improvements ($\approx$ 2.2**X**) than ResNet ($\approx$ 1.2**X**). Considering that the TEE randomly decides to verify an SGD step with probability p_v, the overall improvement is approximately multiplied by $\frac{1}{p_v}$**X**. For instance, if we assume the attacker only needs to deviate with probability $7E{-}5$ (i.e., deviating only 70 steps out of 1M steps), then we can detect such deviation with $p_v \approx 0.1$ (See Figure 8d in Appendix). For VGG networks, this means that approximately 22**X** performance improvement in throughput compared to a pure TEE-based solution that verifies every step. Nonetheless, as our experiment results suggest, we believe that attacking deep models that are trained on very large datasets should require a significantly larger number of deviations. This, in return, may result in a much greater performance gain.

6.2 TEE Performance on CIFAR10

Figure 5 depicts the throughput performance for the CIFAR10 dataset and 9 different VGG architectures . We chose three popular VGG(11,13,16) architectures adapted for CIFAR10 image inputs with custom fully connected layers attached to its end. FC-1 is one layer (128×10), FC-2 is two layers (128×64, 64×10), and FC-3 is two layers (256×128, 128×10). For CIFAR10, our verification technique generally do not benefit from randomized matrix multiplication scheme as it did for ImageNet. This is mainly because most of the operations and network layers fit well within the hardware memory limit. Therefore, since the dimensions of MM operations are not too large, using randomized MM verification does not improve performance significantly.

6.3 Enclave Heap Size Impact on TEE Performance

Figure 6 depicts the impact of the heap size on the performance of DNNs for a single SGD step. It can be seen that increasing the heap size substantially beyond the available hardware limit (to around 92MB) can induce a negative impact on the performance. This is especially the case for the VGG architecture. This result is mainly due to two factors. First, it causes driver level paging, which must evict enclave pages that require an extra level of encryption/decryption. Second, there is a non-trivial amount of extra bookkeeping required for the evicted pages.

6.4 Combined Impact of Gradient Clipping and Learning Rate on Attack Success

As we discussed in Section 1, if the SGD updates are not bounded, the attacker can launch a successful attack by deviating from the correct training in a single update step. To prevent this, we require each update to be clipped using the gradient clipping approach. We conducted series of experiments to understand how gradient clipping impacts the attacker success in poisoning the model during training. As shown in Table 1, we selected the models that achieved high performance on both clean and poisoned test samples (267 out of 600+). [2] First, during the attack, attacker follows the correct protocol (mini-batch SGD) until $epoch_{attack}$. At this epoch, the attacker starts attacking by injecting a certain number of poisoned samples ($pois_{rate} \times batch\_size$) from every class into the training batch and labels it as the target label. The attacker continues to attack until they achieve a desired threshold in terms of success rate (correctly classifying backdoored samples as the attacker's target

[2] We also performed a series of experiment for the scenario where the attack is performed on steps chosen at random. However, this type of attack was not successful, such that we do not report those results here.

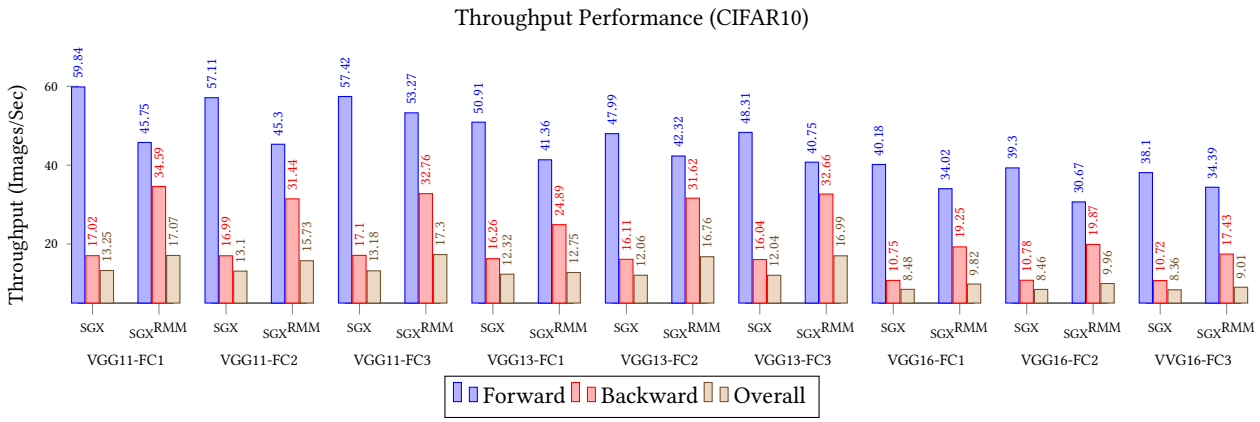
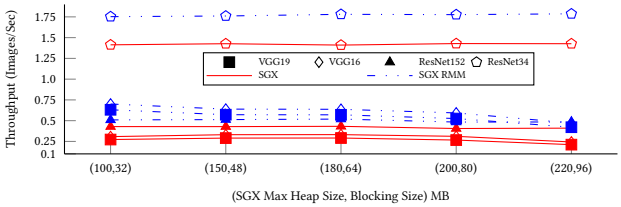
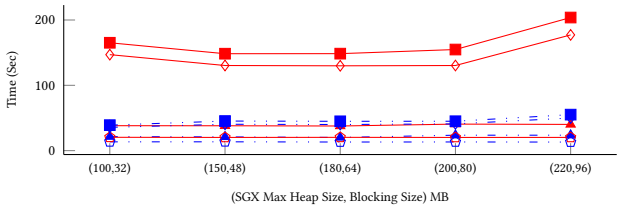


Figure 5: Throughput of SGD training step for VGG19,VGG16, ResNet152, and Resnet34 on CIFAR10 dataset. Randomized Matrix Multiplication (Freivalds' scheme) can make verification twice faster in case of VGG architecture.



(a) Available heap with respect to throughput



(b) Available heap with respect to time spent on matrix-matrix(vector) multiplication

Figure 6: The impact of increasing TEE heap size on (a) overall throughput and (b) the time spent in matrix multiplication routine. VGG shows significant reduction in performance as opposed to ResNet.

label). Once it passes the threshold, the attacker halts the attack and returns to the honest protocol, while observing the decay in attack success rate. If the success rate falls below a desired lower threshold, the attacker transitions back to attack mode and repeats the aforementioned strategy. The CIFAR10 ([19]), GTSRB ([33]) and MNIST ([21]) datasets were used to analyze the impact of multiple factors imposed by the attacker. For MNIST, and GTSRB, the *Adam* [17] optimizer (which requires a smaller initial learning rate), and for the CIFAR10 dataset, *SGD* with momentum are used. Additionally, for MNIST, and GTSRB 10% of the training set was chosen for the validation set to help adjust the learning rate based on the validation set loss. Same for the CIFAR10 dataset, the learning rate was set to decay (by tenfold) at fixed epochs (40, 70, 100).

6.4.1 Backdoor Trigger Pattern. We applied 6 different backdoor triggers (Figure 3). The MNIST dataset only has single channel images, we converted it to a three channel image to apply the triggers 3 to 6. As shown in Figure 4 (first row) the trigger pattern can significantly influence the effectiveness of the attack. The red lines show the median, while the green dots correspond to the mean. In all of the datasets, the first trigger pattern (Figure 3a) was the most effective. This pattern covers a wider range of pixels compared to the second trigger type (Figure 3b). As a consequence, it is more likely that the model can remember the trigger pattern across longer periods of SGD steps. Moreover, since photo filters (e.g. *Instagram*) have become popular, we investigated the potential for conducting attacks using some of the filters (or transformations) as the trigger pattern. However, covering a very wide range of pixels does not lead to a stealthy attack, as illustrated by the last four patterns. These patterns cover the whole input space and transform it to a new one that they share a lot of spacial similarities while only different in tone or scale (e.g. Figure 3d). Learning to distinguish inputs that are similar, and only different in their tone, is demanding in terms of continuity of the attack. In this case, both of the images (that is, with and without the trigger) are influencing most of the parameters and filters in a contradictory manner (different classification label). Thus, it takes a significant number of steps for the network to learn to distinguish them when gradient clipping is applied.

6.4.2 Attack Start Epoch. Another major factor influencing the evasiveness of the attack is when the attacker initiates the attack. For instance, early in the training phase the learning rate will be high, such that a savvy attacker might believe that they can avoid low clipping values by initiating their attack. However, if the attack begins too early, then it is unlikely that the model has yet converged. As a result, the attack may need to commit a substantially higher number of (unnecessarily) poisoned batches, which, in turn, would raise the probability of detection. Yet even if the attacker was successful, once they halt the attack, the model will likely evolve

the parameters back to a clean state relatively quickly, such that, once again, the attacker would need to re-initiate their attack. Additionally, because of a low clipping value, if the attacker waits until later in the training process, the attack is again unlikely to be effective. In this case, this would mainly be due to the considerably smaller learning rate. As shown in Figure 4, the best time to attack is when 1) the model has a relatively low loss on clean training inputs and 2) the combination of learning rate and clipping value (effective attainable update) allows the model to move toward attacker's desired optima. For MNIST, which is a trivial learning task, attacking early endows the attacker with a better chance to launch a stealthier attack. We believe that this is due, in part, to the faster convergence of the model. After a few epochs, the system quickly reaches a stable, low training loss for clean images. As a result, after reaching the desired attack success threshold, the attack success is generally preserved far longer than the other two datasets.

6.4.3 Mini-Batch Poisoning Ratio. Another critical factor is $pois_{rate}$, the ratio of the number of poisoned samples in the batch to the batch size. This is particularly the case when gradient clipping is applied. Setting $pois_{rate}$ appropriately can help the attacker by moving more parameters toward the desired optima. However, going beyond a ratio of 0.9 (i.e., $pois_{rate} > 0.9$) can impact the training negatively for both clean inputs and poisoned inputs. As depicted in Figure 4, our experiments suggest that filling more than half the batch with poisoned samples seems to be effective across all datasets. For the MNIST dataset, it appears that higher values can achieve slightly better performance, however, this finding is not replicated in more complex datasets. For example, for the GTSRB dataset, we did not observe a successful attack on the model where clean input accuracy is close to the clean input accuracy where there is no attack.

6.5 Impact of Gradient Clipping for Honest Trainers

One important question is whether the gradient clipping (applied to prevent the attacker from changing parameters in a given mini-batch update) can have a performance impact on training when there is no attack. We ran six experiment configurations, repeated 5 times, each with different randomness (initialization, batch order, etc.). Initial learning rates are set $\in (0.1, 0.01)$ and clipping thresholds are set $\in (nil, 0.001, 0.0005)$ (nil stands for no gradient clipping). In total, there were 30 ResNet56 (complex architecture with state-of-the-art performance) models trained on the CIFAR10 dataset (with no attack) for 200 epochs. Usually, for the *SGD* [29] optimizer with momentum, a learning rate value of 0.1 is chosen, (and for *Adam* optimizer a value less than 0.001). The reader can refer to [2] for an introductory background on deep learning optimization paradigms. For these experiments, we used the configuration with unbounded gradient updates as the main reference point. For learning decay schedule, we used a fixed decay by tenfold at epochs (50, 90, 130, 160).

Figure 7a shows the mean and standard deviation (dashed lines) of test accuracy taken for 5 runs at each epoch for the two learning rate configurations. As it can be seen, both models start to take giant leaps toward convergence at the first two learning decays enforced by the scheduler. Note that these reference runs have no

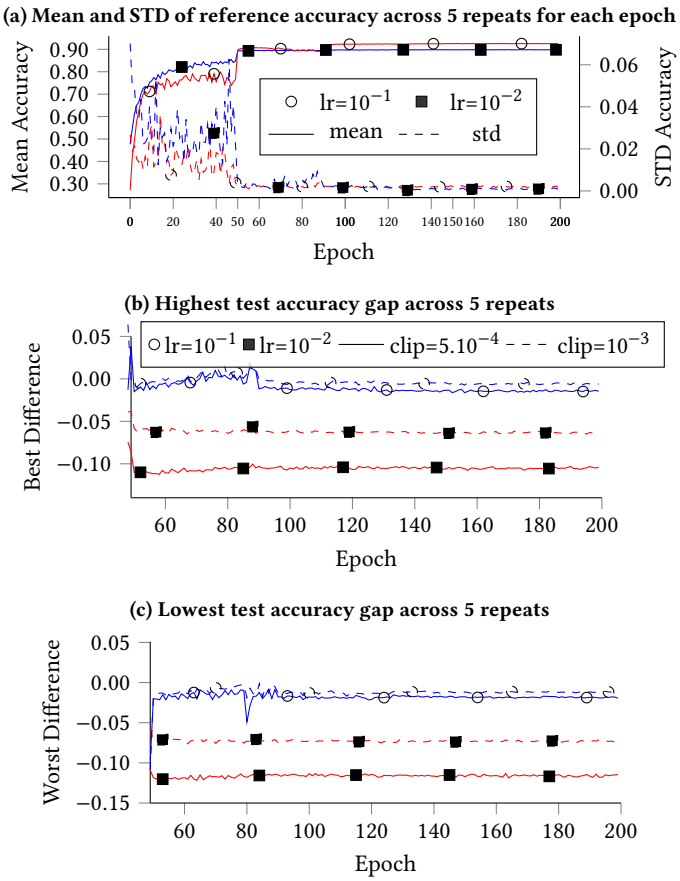


Figure 7: 7a Reference Models (no gradient clipping) mean/std on test accuracy of 5 repeats for two different learning rates. Each configuration had 5 repeats and a reference model (no attack and unbounded updates). 7b For each run configuration the test accuracy difference $diff_{lr,clip}$ is defined as $max\left(acc^{rep}_{lr,clip} - acc^{rep}_{ref}\right) \quad \forall rep \in [1,5]$. 7c $min\left(acc^{rep}_{lr,clip} - acc^{rep}_{ref}\right) \quad \forall rep \in [1,5]$

gradient clipping enforcement during the update step. Toward the end of training, the setting with the higher initial learning rate slightly outperforms in terms of accuracy ratio.

In Figure 7b, the largest difference (accuracy rate) with respect to the reference run is plotted for each combination of learning rate and clipping value. The plot shows that test accuracy is minimally influenced by the clipping value. Rather, the test accuracy is highly dependent on the learning rate value. When $lr = 0.1$, both clipping values can achieve values that are close to the reference runs that have no gradient clipping. In Figure 7c, the opposite - that is, the smallest difference - of the previous measure is plotted. Again, by the end of the training, the gaps are significantly reduced for the case where a better learning rate is chosen. Therefore, having a smaller clipping value has minimal impact on the performance,

which is notable because it is crucial to set a reasonable learning rate.

Overall, Figure 7 shows that clipping has negligible impact on the learning task once an appropriate learning rate is chosen. It can be seen that, if the trainer chooses an acceptable learning rate for the task, small clipping values (e.g., 0.001 or 0.0005) do not impede the learning task. Once the model passes the first learning rate decay schedule, all of the configurations behaves the same in terms of their test performance compared to their reference model (no gradient clipping limit).

6.6 Gradient Clipping Impact Analysis On Large Dataset

We investigated our attack on the ImageNet dataset with the ResNet34 architecture. The state-of-the-art reference achieves an accuracy (rate) of 0.73. However, we only reinitialized and trained the fully-connected plus the last four convolutional layers (10M out of 21M parameters). We used an SGD optimizer for 30 epochs, and the initial learning rate was set to 0.05. Additionally, the learning rate was set to decay (by tenfold) at fixed epochs (4, 7, 13). Also, $pois_{rate}$ was 0.5, and the first trigger pattern (Figure 3a) was used as the only pattern (resized to 256x256). Finally, at the beginning of the third epoch (one epoch before the first decay) the attack started.

6.6.1 Gradient Clipping Impact on Poisoning Backdoor Attack Evasiveness. We conducted the attack for a subset of source labels (10, and 50 out of 1000 classes) to a single target class. Table 2 shows that small clippings (2 and 5) forces the attacker to continue for a longer period of time, which results in a small verification rate while supporting a reasonable performance loss (around 0.02). It is evident that the total number of malicious steps to launch successful attacks (achieved within 30 epochs of training) demands a very low verification rate, given the detection failure $< 10^{-3}$. It is common for models that are built on the ImageNet to perform more than 1**M** SGD steps. To require anything beyond 0.1 verification rate, the total number of steps that the attacker intervenes should be below 70, which is unlikely given the strict gradient clippings.

6.6.2 Gradient Clipping Impact On Training W.O Attack. To investigate if the optimal accuracy that can be achieved for the current setting (partial retraining) , we selected four clipping values: unbounded, mild (5.0), low (5.0), and tiny (0.0005). The training with unbounded clipping value was unstable and the model barely achieved an accuracy of 0.01. We believe that this is an example of how gradient clipping can help large DNN models converge faster.[3] However, the training with the low clipping values converged to an acceptable optima of ≈0.72 accuracy. It should be noted that we only trained the model for far fewer iterations than what is usual for ImageNet (1M-2M steps). Additionally, for the tiny clipping value, the model converged to an optimum accuracy rate of ≈0.61. We wish to highlight that, for large datasets (millions of inputs), if the trainer can choose a suitable learning rate, gradient clipping values that are not significantly smaller than the learning rate, are unlikely to incur an unacceptable performance overhead, and empirically, we observe that gradient clipping can also help the model converge faster.

[3]Training from scratch with unbounded clipping can potentially take at least a week using a single GPU of 16GB.

7 CONCLUSION

This paper introduced the GINN system, which provides integrity in outsourced DNN training using TEEs. As our experimental investigation illustrates, GINN scales up to realistic workloads by randomizing both mini-batch verification and matrix multiplication to achieve integrity guarantees with a high probability. We have further shown that random verification in combination with hyperparameter adjustment (e.g., setting low clipping rates), can achieve 2**X**-20**X** performance improvements in comparison to pure TEE-based solutions while catching potential integrity violations with a very a high probability.

8 ACKNOWLEDGEMENTS

The research reported herein was supported in part by NIH award 1R01HG006844, NSF awards, CNS-1837627, OAC-1828467, IIS-1939728, DMS-1925346, CNS-2029661 and ARO award W911NF-17-1-0356.

REFERENCES

[1] [n.d.]. Confidential computing on Azure. https://docs.microsoft.com/en-us/azure/confidential-computing/overview#introduction-to-confidential-computing
[2] [n.d.]. Intro to optimization in deep learning: Momentum, RMSProp and Adam. https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/. Accessed: 2010-09-30.
[3] Nazmiye Ceren Abay, Yan Zhou, Murat Kantarcioglu, Bhavani Thuraisingham, and Latanya Sweeney. 2019. Privacy Preserving Synthetic Data Release Using Deep Learning. In *Machine Learning and Knowledge Discovery in Databases*, Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim (Eds.). Springer International Publishing, Cham, 510–526.
[4] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13. ACM New York, NY, USA.
[5] Xiangyi Chen, Zhiwei Steven Wu, and Mingyi Hong. 2020. Understanding Gradient Clipping in Private SGD: A Geometric Perspective. arXiv:2006.15429 [cs.LG]
[6] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.
[7] Rov Csongor. [n.d.]. Tesla Raises the Bar for Self-Driving Carmakers. www.blogs.nvidia.com/blog/2019/04/23/tesla-self-driving/
[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.
[9] Rusins Freivalds. 1977. Probabilistic Machines Can Use Less Running Time.. In *IFIP congress*, Vol. 839. 842.
[10] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*. PMLR, 201–210.
[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. 10.11 Optimization for Long-Term Dependencies. *Deep Learning* (2016), 408–411.
[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. 6.5 Back-Propagation and Other Differentiation Algorithms. *Deep Learning* (2016), 200–220.
[13] 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).
[14] 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.
[15] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving Machine Learning as a Service. *CoRR* abs/1803.05961 (2018). arXiv:1803.05961 http://arxiv.org/abs/1803.05961
[16] Nick Hynes, Raymond Cheng, and Dawn Song. 2018. Efficient Deep Learning on Multi-Source Private Data. *CoRR* abs/1807.06689 (2018). arXiv:1807.06689 http://arxiv.org/abs/1807.06689
[17] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] Panagiota Kiourti, Kacper Wardega, Susmit Jha, and Wenchao Li. 2019. TrojDRL: Trojan Attacks on Deep Reinforcement Learning Agents. *CoRR* abs/1903.06638 (2019). arXiv:1903.06638 http://arxiv.org/abs/1903.06638

[19] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n.d.]. CIFAR-10 (Canadian Institute for Advanced Research). ([n. d.]). http://www.cs.toronto.edu/~kriz/cifar.html

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*.

[21] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/

[22] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. 2019. ABS: Scanning Neural Networks for Back-Doors by Artificial Brain Stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. https://doi.org/10.1145/3319535.3363216

[23] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2017. Trojaning attack on neural networks. (2017).

[24] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1289–1306. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic

[25] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. *IACR Cryptology ePrint Archive* 2017 (2017), 396. http://eprint.iacr.org/2017/396

[26] Lucien KL Ng, Sherman SM Chow, Anna PY Woo, Donald PH Wong, and Yongjun Zhao. 2021. Goten: GPU-Outsourcing Trusted Execution of Neural Network Training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 14876–14883.

[27] Harsh Panwar, P.K. Gupta, Mohammad Khubeb Siddiqui, Ruben Morales-Menendez, and Vaishnavi Singh. 2020. Application of deep learning for fast detection of COVID-19 in X-Rays using nCOVnet. *Chaos, Solitons & Fractals* 138 (2020), 109944. https://doi.org/10.1016/j.chaos.2020.109944

[28] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.

[29] Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951).

[30] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. 2018. Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks. In *Advances in Neural Information Processing Systems 31*. http://papers.nips.cc/paper/7849-poison-frogs-targeted-clean-label-poisoning-attacks-on-neural-networks.pdf

[31] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* (2014).

[33] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* (2012). https://doi.org/10.1016/j.neunet.2012.02.016

[34] Lichao Sun. 2020. Natural Backdoor Attack on Text Data. arXiv:2006.16176 [cs.CL]

[35] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

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

[38] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. 2019. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *2019 IEEE Symposium on Security and Privacy*.

[39] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. 2019. Why gradient clipping accelerates training: A theoretical justification for adaptivity. arXiv:1905.11881 [math.OC]

A SYMBOLS

Table 3: Symbols and Acronyms Description

Category	Symbol	Description
TEE	$K_{SGX}^{session}$	TEE's session key for learning task
	Sig_{SGX}^{s}	SGX signature signing key
	K_{client}	client's encryption key
	Sig_{client}^{v}	clients public key
	PRM	Processor Reserved Memory
	EPC	Enclave Page Cache
Neural Network	RMM	Randomized Matrix Multiplication
	FV	Full Verification (No RMM)
	DNN	Deep Neural Network
	FC	Fully-Connected
General	$\mathcal{W}$	model parameters
	ds	training dataset
	v_num	model version number
	verf_rate	rate for the TEE to verify a training step

B MODEL SIGNING BY TEE

We assume, an honest and authenticated user will send her data encryption key K_{client} (after remote-attestation) to the TEE. Next, the TEE decrypts and verifies the initial encrypted dataset using the K_{client} and supplies the trainer (GPU) the plain-text of the training set. If the TEE fails to detect any violations of the protocol during training, it will sign the following message that certifies the final model where $\mathcal{W}$ is the model parameters, ds is training dataset, v_num is the model version number, $SHA256$ is Sha 256 bit cryptographic hash function,

$$SHA256(SHA256(\mathcal{W})||v\_num||SHA256(ds)||Sig_{client}^{v})$$

with signature key Sig_{SGX}^{s} of the enclave.

C MATRIX MULTIPLICATION OPS OF COMMON DNN LAYERS

Table. 4 shows common MM operations in DNNs. Connected and convolutional layers use MM routines to compute feed forward output, parameter gradients, and input gradients.

D VERIFICATION PROBABILITY GROWTH WITH RESPECT TO DETECTION PROBABILITY

Fig. 8 shows how verification probability changes with respect to the probability that a batch step is maliciously manipulated by the attacker. First row shows the verification probability for a dataset with 60K samples. Second row depicts the required for much bigger dataset (1M samples) over different mini-batch sizes. The smaller the mini-batch size is, there is a higher chance for detecting malicious behavior.

E GINN BLOCKING OF BIG MATRICES

By default, GINN allocates/releases resources on a per layer basis. In the event that even for a single sample, it is not possible to satisfy the memory requirements of network (either large network or large inputs), GINN breaks each layer even further.

For convolutional layers, the main memory bottleneck is *im2col*[4] which converts the layer's input (for each sample) of size $c_i \cdot w_i \cdot h_i$ to $[k^2 \cdot c_i] \times [w_o \cdot h_o]$ (k is kernel window size) matrix for a more efficient matrix multiplication. GINN divides the inputs across the channel dimension and processes the *im2col* on maximum possible channels that can be processed at once.

[4] extracts redundant patches from the input image and lays in columnar format

Table 4: Matrix Multiplication Operations

Layer Type	Pass	Computation	Verification	(Sub)Batched/ Precomp.
Fully Connected	Forward	$O_{[B][O]} = \mathcal{I}_{[B][I]} \times (\mathcal{W}_{[O][I]})^\intercal$	$\mathrm{Y}_{[I][1]} = (\mathcal{W}_{[O][I]})^\intercal \times \mathcal{R}_{[O][1]}$ $\mathcal{Z}_{[B][1]} = \mathcal{I}_{[B][I]} \times \mathrm{Y}_{[I][1]}$ $\mathcal{Z}'_{[B][1]} = O_{[B][O]} \times \mathcal{R}_{[O][1]}$	YES / YES
	Backward Parameters Gradient	$\nabla^{\mathcal{W}}_{[O][I]} = (\nabla^{O}_{[B][O]})^\intercal \times \mathcal{I}_{[B][I]}$	$\mathrm{Y}_{[B][1]} = \mathcal{I}_{[B][I]} \times \mathcal{R}_{[I][1]}$ $\mathcal{Z}_{[O][1]} = (\nabla^{O}_{[B][O]})^\intercal \times \mathrm{Y}_{[B][1]}$ $\mathcal{Z}'_{[O][1]} = \nabla^{\mathcal{W}}_{[O][I]} \times \mathcal{R}_{[I][1]}$	YES / NO
	Backward Inputs Gradient	$\nabla^{\mathcal{I}}_{[B][I]} = \nabla^{O}_{[B][O]} \times \mathcal{W}_{[O][I]}$	$\mathrm{Y}_{[O][1]} = \mathcal{W}_{[O][I]} \times \mathcal{R}_{[I][1]}$ $\mathcal{Z}_{[B][1]} = \nabla^{O}_{[B][O]} \times \mathrm{Y}_{[O][1]}$ $\mathcal{Z}'_{[B][1]} = \nabla^{\mathcal{I}}_{[B][I]} \times \mathcal{R}_{[I][1]}$	YES / YES
Convolutional	Forward	$O_{[f][w_o.h_o]} = \mathcal{W}_{[f][k^2.C_i]} \times \mathcal{I}_{[k^2.C_i][w_o.h_o]}$	$\mathrm{Y}_{[1][k^2.C_i]} = \mathcal{R}_{[1][f]} \times \mathcal{W}_{[f][k^2.C_i]}$ $\mathcal{Z}_{[1][w_o.h_o]} = \mathrm{Y}_{[1][k^2.C_i]} \times \mathcal{I}_{[k^2.C_i][w_o.h_o]}$ $\mathcal{Z}'_{[1][w_o.h_o]} = \mathcal{R}_{[1][f]} \times O_{[f][w_o.h_o]}$	NO / YES
	Backward Parameters Gradient	$\nabla^{\mathcal{W}}_{[f][k^2.C_i]} = \nabla^{O}_{[f][w_o.h_o]} \times (\mathcal{I}_{[k^2.C_i][w_o.h_o]})^\intercal$	$\mathrm{Y}_{[w_o.h_o][1]} = (\mathcal{I}_{[k^2.C_i][w_o.h_o]})^\intercal \times \mathcal{R}_{[k^2.C_i][1]}$ $\mathcal{Z}_{[f][1]} = \nabla^{O}_{[f][w_o.h_o]} \times \mathrm{Y}_{[w_o.h_o][1]}$ $\mathcal{Z}'_{[f][1]} = \nabla^{\mathcal{W}}_{[f][k^2.C_i]} \times \mathcal{R}_{[k^2.C_i][1]}$	NO / NO
	Backward Inputs Gradient	$\nabla^{\mathcal{I}}_{[k^2.C_i][w_o.h_o]} = (\mathcal{W}_{[f][k^2.C_i]})^\intercal \times \nabla^{O}_{[f][w_o.h_o]}$	$\mathrm{Y}_{[1][f]} = \mathcal{R}_{[1][k^2.C_i]} \times (\mathcal{W}_{[f][k^2.C_i]})^\intercal$ $\mathcal{Z}_{[1][w_o.h_o]} = \mathrm{Y}_{[1][f]} \times \nabla^{O}_{[f][w_o.h_o]}$ $\mathcal{Z}'_{[1][w_o.h_o]} = \mathcal{R}_{[1][k^2.C_i]} \times \nabla^{\mathcal{I}}_{[k^2.C_i][w_o.h_o]}$	NO / YES

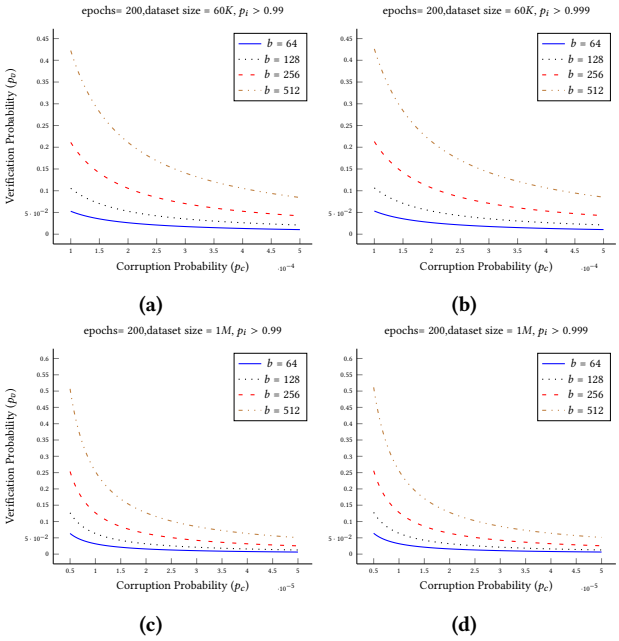


Figure 8: Required verification probability with respect to batch corruption probability and the desired integrity probability for a fixed 200 epochs and different SGD batch size.

For fully-connected layers, the main memory bottleneck is the parameters matrix $\mathcal{W}_{[O]\cdot[I]}$ that does not depend on the batch size. GINN divides the matrix across the first dimension (rows), and processes the outputs on the maximum possible size of rows that fits inside the TEE for the corresponding layer.