# Towards Efficient and Reliable Training Assurance of Untrusted Federated Learning Participants Under Hardware Non-Determinism

Cheng Lyu, Xiaoli Zhang, Jiaqing Cheng, Wenmao Liu, Xiaohu Ye, Ke Xu, *Fellow, IEEE*, Qi Li, *Senior Member, IEEE*, and Xu-Cheng Yin, *Senior Member, IEEE*

***Abstract*—Federated learning (FL) is a popular privacy-preserving machine learning paradigm, enabling collaborative training across participants without exposing local data. Since FL loses direct control over participants' training executions, a fundamental requirement is to verify that participants faithfully perform the assigned training tasks. In this paper, we present TrustFL+, an efficient, scalable, and reliable verification scheme that ensures the training correctness of federated learning participants by leveraging both Trusted Execution Environments (TEEs) and GPUs. Essentially, it pushes all local training on high-performance but untrusted GPUs, while the TEE replicates the random parts for tunable levels of assurance. A key challenge is that hardware non-determinism can cause the same floating-point operations to yield different results between GPUs and TEEs, leading to false positives when participants behave honestly. TrustFL+ builds on deterministic training by recording rounding directions of intermediate operations during GPU-side model training and reusing them in TEE-based verification. It especially introduces adaptive rounding precisions to practically control non-determinism while maintaining global model performance in federated learning systems with lots of heterogeneous GPUs and iterative training. We prototype TrustFL+ using a range of NVIDIA GPUs covering multiple hardware architectures, along with Intel SGX, and evaluate its performance across convolutional neural networks and transformer-based networks. The experimental results demonstrate that TrustFL+ delivers up to an order of magnitude speedup compared to naive SGX-based training. Furthermore, all models trained with TrustFL+ on different GPU architectures successfully pass verification within SGX, resulting in 0 false positives.**

***Index Terms*—Federated learning (FL), hardware non-determinism, trusted execution environments, verifiable training.**

## I. Introduction

Federated learning (FL) [1], [2], as a popular privacy-preserving machine learning framework, supports applications in healthcare, finance, and large-scale personalized AI services [3], [4], where direct data sharing is infeasible due to privacy regulations like the EU GDPR [5]. It enables multiple data owners (aka participants) to compute local model updates based on their private data and then aggregates the updates into a global model. However, since FL lacks visibility into participants' local training processes, malicious or lazy participants may deviate from the assigned computation: forging model updates, skipping or tampering with training iterations. Such execution misbehavior can degrade model performance, hinder convergence, facilitate stealthy backdoor insertion [6], [7], [8], or even lead to reward fraud in FL-based marketplaces where model updates are monetized [9].

Therefore, it is imperative to enable verifiable training of participants in federated learning systems. To achieve this, the design must satisfy four key objectives: 1) *Security guarantee*, which ensures participants faithfully execute the assigned training task; 2) *Efficiency*, which requires that participant-side overhead remains comparable to native (insecure) GPU-based training; 3) *Scalability*, which demands that the server-side cost (e.g., for proof verification) does not grow significantly with the number of participants, model size, or data volume; 4) *Reliability*, which ensures the verification results are trustworthy and free from false positives when participants behave honestly.

Existing studies on verifiable training mainly fall into three classes, but none of them simultaneously satisfy all the above objectives, as summarized in Table I. First, cryptographic primitives [10], [11], [12] are impractical for verifying modern deep neural network models due to their prohibitive computational cost. For instance, state-of-the-art zero-knowledge proof designs require over eight minutes to generate proofs for predicting just 20 images [10], [11], [12]. Second, Trusted Execution Environments (TEEs) can enforce training integrity [13], [14], yet are typically one to two orders of magnitude slower than GPU accelerators commonly used for training deep neural networks [15]. Third, the reproduction-based verification, in which a trusted party (e.g., the FL server or an external auditor) re-executes the

Received 31 July 2025; revised 11 March 2026; accepted 12 March 2026. Date of publication 25 March 2026; date of current version 9 July 2026. This work was supported in part by NSFC under Grant 62302452, Grant 62132011, and Grant 62425201, in part by Beijing Natural Science Foundation under Grant 4262028, in part by the Open Research Fund of State Key Laboratory of Internet Architecture under Grant HLW2025ZD02, in part by China Postdoctoral Science Foundation under Grant 2025M771583, in part by the Young Elite Scientists Sponsorship Program of the Beijing High Innovation Plan under Grant 20250835, in part by the Fundamental Research Funds for the Central Universities under Grant FRF-TP-25-037, in part by the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China under Grant JYB2025XDXM114, in part by Beijing-Tianjin-Hebei Natural Science Foundation Cooperation Project under Grant 25JJJJC0003, and in part by CCF-NSFOCUS 'Kunpeng' Research Fund. *(Corresponding authors: Xiaoli Zhang; Qi Li.)*

Cheng Lyu, Xiaoli Zhang, Jiaqing Cheng, and Xu-Cheng Yin are with the School of Computer & Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China (e-mail: xiaoli.z@ustb.edu.cn; lc@xs.ustb.edu.cn; qing@xs.ustb.edu.cn; xuchengyin@ustb.edu.cn).

Wenmao Liu and Xiaohu Ye are with NSFOCUS Technologies, Inc., Beijing 100089, China (e-mail: liuwenmao@nsfocus.com; yexiaohu@nsfocus.com).

Ke Xu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: xuke@tsinghua.edu.cn).

Qi Li is with the Institute of Network Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: qli01@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TDSC.2026.3677485

1545-5971 © 2026 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF PRIOR DEFENSE SOLUTIONS AGAINST UNTRUSTED PARTICIPANTS

| Method | | Security Guarantee | Efficiency | Scalability | Reliability |
|---|---|---|---|---|---|
| Cryptographic Primitives [10]–[12] | | ✓ | × | ✓ | ✓ |
| Full TEE Execution [13], [14] | | ✓ | × | ✓ | ✓ |
| Reproduction-based Verification | Threshold-based [16] | ×[1] | ✓ | × | × |
| | Rounding-based [17] | ✓ | ✓ | × | ✓ |
| TrustFL [21] | | × | ✓ | ✓ | × |
| **TrustFL+** | | ✓ | ✓ | ✓ | ✓ |

[1] PoL [16] has been shown to be vulnerable to an attack that produces a verifiable yet fake training trajectory with lower cost than real training [22], [23].

participants' training process for verification [16], [17], does not scale well to federated learning systems with many participants, complex models, and large datasets. Moreover, these approaches suffer from a reliability limitation: even with identical data and random seeds, exact reproduction of training is difficult, as hardware non-determinism, e.g., parallel computation on GPUs, can introduce accumulation errors in floating-point operations like matrix-vector multiplication [18], [19], [20].

To advance the state-of-the-art, we propose TrustFL+, a GPU-TEE co-designed decentralized verification scheme that enables efficient, scalable, and reliable training assurance against untrusted participants in federated learning systems. Specifically, each participant conducts the entire training process on an untrusted but high-performance GPU, while its co-located TEE adopts a reproduction-based verification strategy that re-executes a small fraction of the training process to validate the integrity of local training. The TEE then generates a signed attestation report, which encapsulates evidence of correct execution and is subsequently verified by the FL server before model aggregation. This design ensures efficiency, as it avoids the prohibitive overhead of full training execution inside the TEE [13], [14] or cryptographic proof generation [10], [11], [12]. By limiting the server's role to lightweight signature verification, the design eliminates centralized bottlenecks and scales to a large number of participants.

However, pushing reproduction-based verification processes into TEEs that are controlled by untrusted participants is challenging in terms of security and reliability. First, in the GPU-TEE interactive verification procedure, malicious participants may selectively execute the training segments that are subject to TEE verification while skipping the rest. Second, since the TEE's scheduling and I/O are controlled by the host participant [24], an adversary may reuse results of prior training epochs[1] for verification without being detected. Third, as mentioned earlier, naive reproduction of training would produce unreliable verification results (i.e., false positives) due to hardware non-determinism.

Correspondingly, we devise three techniques to address these challenges. The first is *secure sampling*, which adopts a commit-and-prove paradigm to ensure that the entire GPU-side training process is completed before the TEE reveals the sampling decisions for verification. The second technique is *dynamic-yet-deterministic data selection*, which introduces controlled randomness into the training inputs so that each epoch and each step produce distinguishable results. These dynamics can be determined by the TEE during verification, ensuring that the training results are fresh and not reused. The third is *a novel adaptive rounding strategy* that builds on the prior deterministic training method [17]. It allows the GPU to perform model training in higher precision (e.g., FP64), periodically rounds to a target precision (e.g., FP32), and records the rounding directions for the TEE to replay. More importantly, unlike prior work [17], which requires access to GPUs of the same type as each participant's device to determine its rounding precision, our approach introduces the global minimum rounding precision issued by the server. This precision is determined by empirically balancing the error tolerance of heterogeneous GPU architectures with the global model performance during interactive training, and is then shared with all participants for reliable verification. We note that this design addresses a limitation in our prior conference version (TrustFL [21]), which did not account for hardware non-determinism across diverse GPU architectures and neural network models.

We implement TrustFL+ using SGX library OS-Occlum [25] and open-source it at Github[2]. Extensive experiments are conducted across a range of neural network models (VGG16, VGG19, ResNet50, GPT2) and GPU architectures (Hopper, Ada Lovelace, Ampere, Turing, Volta). The results show that, compared with TrustFL [21], TrustFL+ reduces false positive rates from over 39% to 0 across diverse GPUs and models. Besides, TrustFL+ introduces a moderate 20.4%–99.5% increase in participant-side processing time compared to native GPU training, while delivering up to 10× speedup over pure TEE-based training. TrustFL+ also maintains server-side verification time at only 0.5% of recent reproduction-based verification method [17] as the number of participants increases from 5 to 20, demonstrating strong scalability to large-scale FL systems.

*Contributions:* In summary, our contributions are:

- We propose a practical scheme, TrustFL+, that combines TEE and GPU to efficiently verify the integrity of training at the participants' side with high confidence in federated learning systems.
- Our scheme employs secure sampling and dynamic-yet-deterministic data selection techniques to defend against adversarial manipulations in GPU-TEE interactive verification procedure.
- Compared to our prior conference version (TrustFL [21]), TrustFL+ introduces an adaptive rounding strategy that allows the TEE to replay training computations from diverse GPU architectures deterministically, enabling reliable verifiable training without requiring hardware-specific introspection.
- The theoretical analysis demonstrates that our scheme can achieve high-confidence training assurance with sparse

[1] In federated learning, we use "training epoch" to refer to a participant's full local training phase between two global model updates. We divide the local training process into discrete "training segments", recording intermediate model weights as checkpoints at the end of each segment. A "training step" refers to performing one complete forward pass and one backward pass on a single batch of data during local training.

[2] https://github.com/Blue-zenith-buzhan/TrustFL_Plus.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

sampling. Experimental results show TrustFL+ achieves nearly an order of magnitude improvement in efficiency compared to pure TEE-based training and obtains 0 false positives.

## II. Problem Statement

### A. Preliminaries

*Federated Learning:* Federated learning is a form of distributed and collaborative machine learning which has many different implementations regarding various data distribution and participant scale [1], [2], [26]. Our work aims at assuring DNN training of participants and is compatible with arbitrary FL implementations. In this paper, we take a concrete implementation introduced by Google as an example to elaborate on our scheme.

Assume that there are participants $\mathcal{P}$ with sensitive data $D$, and one server $\mathcal{S}$ that is responsible for coordinating distributed learning tasks. To obtain a converged global model $\theta$, the server performs multi-iteration training. At each epoch, e.g., $t$, the server selects a random subset of participants $\mathcal{P}^t \subseteq \mathcal{P}$ and sends them the latest global model $\theta^t$. Each participant $p$ generates the local model $\theta_p^{t+1}$ over local training data $D_p^t$, and uploads the local model updates which are $\mathcal{L}_p^t = \theta_p^{t+1} - \theta^t$. With all received updates, the server derives the new global model by performing a weighted average as:

$$\theta^{t+1} = \theta^t + \eta \Sigma_{p \in \mathcal{P}^t} \frac{n_p}{n} \mathcal{L}_p^t, \tag{1}$$

where $\eta$ is the global learning rate that instructs how much the model updates at every epoch, $n$ is the total data size and equals to $\Sigma_{p \in \mathcal{P}^t} |D_p^t|$. Note that, the number of training epochs is determined by the time when the model converges, e.g., achieving a target test-set accuracy [26]. **SGX.** Intel's Software Guard Extension (SGX) [27], [28] only requires trust in the processors and provides integrity and confidentiality guarantee for the execution of user-level code. Specifically, SGX is a set of new instructions supported by Intel Skylake CPUs that creates a protected physical memory known as an enclave. The enclave forbids any other process to access enclave memory (called Enclave Page Cache, i.e., EPC). Yet, it can read and write untrusted memory outside the EPC.

SGX supports remote attestation to prove that the software is running correctly in the enclave. When an enclave is launched with the program code, the processor generates a *measurement* which is a hash digest of the loaded binary. Then, the program in the enclave produces a report including the measurement and supplementary data (e.g., output), and signs it as a *quote* with the private attestation key fused in the processor. The quote as an attestation can be verified by anyone possessing a trustworthy measurement of the binary and accessing Intel's Attestation Service (IAS) [29]. For the sake of simplicity, we follow the formal model of SGX presented in [30] where each enclave instance produces an attestation key pair $(sk_{\mathsf{TEE}}, pk_{\mathsf{TEE}})$ and $\Pi$ is the existentially unforgeable signature scheme. An enclave can generate an attestation by calculating a digital signature $\sigma = \Pi_{Sig}(sk_{\mathsf{TEE}}, \Phi, out)$, as a proof that the output $out$ is produced by the program $\Phi$ running in the enclave. The attestation can be further verified by checking whether $\Pi_{Ver}(pk_{\mathsf{TEE}}, \sigma, \Phi, out)$ outputs 1.

*Hardware Non-determinism:* Hardware non-determinism is a well-known challenge in modern machine learning systems.

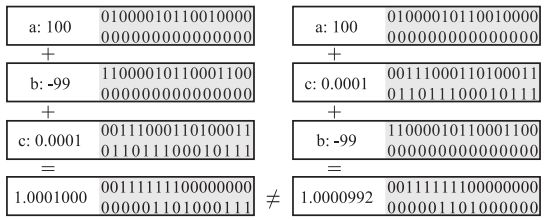



Fig. 1. Example of numerical inconsistency in floating-point arithmetic caused by reordering operations (FP32 Case).

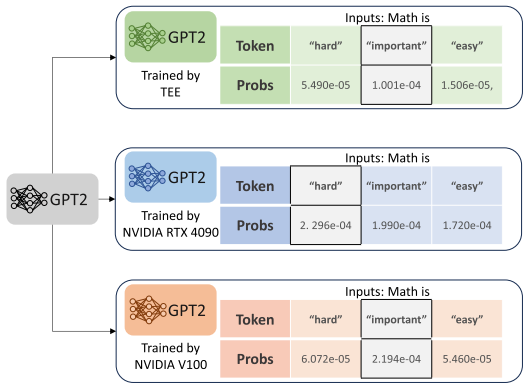



Fig. 2. After being trained for 2500 steps with a batch size of 8 using the same code, dataset, and random seed, the GPT2 models on SGX-based TEE (CPU), NVIDIA RTX 4090, and NVIDIA V100 yield markedly different outputs for identical input.

Even with the same code, data, and random seed, training on different GPU architectures can yield models with similar overall performance (e.g., accuracy) but noticeably different predictions on individual inputs (see Figs. 1 and 2). This divergence is even more pronounced when reproducing GPU-based training on CPU-based TEEs as reported by [31], due to their fundamentally different execution models and floating-point arithmetic behaviors. The sources of such non-determinism are diverse. Beyond floating-point approximation errors and differences in accumulation order [32], [33], parallel execution schedules, memory access patterns, cache behaviors, warp scheduling, and instruction scheduling can all affect the execution order of operations on modern GPUs [34], [35], [36], [37]. While these factors arise at different layers of the hardware and execution stack, their effects ultimately manifest as small perturbations in the lower bits of floating-point mantissas, which accumulate over training and prevent exact reproduction of model execution. TrustFL+ leverages this observation by applying controlled rounding at a predefined mantissa bit position during both GPU training and TEE-side re-execution. Thereby, it achieves bitwise-consistent reproduction and supports reliable verification of honest training execution.

### B. Threat Model

We consider a scenario where a task owner desires to obtain a high-quality DNN model yet without substantial training data. He publishes his requirements in an open data marketplace platform, e.g., building upon blockchain [9], [38], and initiates a crowd-sourced federated learning task with bounty. Participants

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

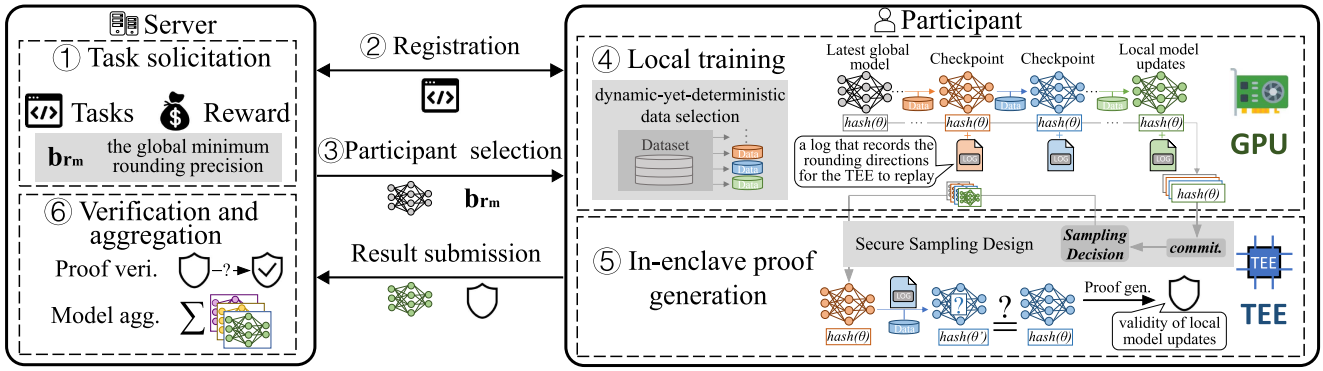



Fig. 3. High-level workflow of TrustFL+. Participants leverage TEE and GPU to fast prove the local training integrity. The task owner verifies the proofs before aggregating their results and distributing rewards.

with appropriate data can join the task to realize the economic value of local data and computation resources while preserving data privacy.

- Task owner is honest-but-curious. He faithfully coordinates the overall FL procedure without any deliberate disruption, as he wants to obtain the aggregated DNN model trained by participants. Yet, he is curious about sensitive data of individuals, e.g., for the purpose of abusing data for profit. While federated learning avoids direct access to raw training data, it is still vulnerable to gradient/updates-based reconstruction attacks [39]. Addressing such privacy threats is orthogonal to the goal of this work and is typically handled by complementary mechanisms such as differential privacy [40] or secure aggregation [41], which are discussed in Section VI.
- Task participants may be dishonest and attempt to conduct *fraudulent training behaviors* in order to gain unfair benefits or compromise the global model. Specifically, such behaviors include forging updates, reusing stale model states, skipping training steps, tampering with the prescribed training iterations, and launching sybil-based cheating, i.e., using one training effort to claim multiple rewards. Our security objective is to build the assurance of training integrity at the participants' side with high confidence. We mitigate sybil attacks by diminishing the gains of adversaries with increased computing efforts. Note that, beyond fraudulent training behaviors, the participants might also use unexpected training data, for example, arbitrary data that deviates from the intended training objective, or poisoned data that degrades the global model performance or injects backdoors [42], [43]. This attack surface concerns the quality of local training data rather than the training integrity, and thereby lies outside the scope of our work. Such threats can be mitigated by existing server-side defenses [44], [45], [46], [47], which are discussed in Section VI.

Attacks in which a participant submits a benign model for verification while deploying a different malicious model in its own private services fall outside the commonly adopted federated learning threat model and, accordingly, outside the scope of our work, as the malicious model used for private services does not influence federated training or global aggregation. In addition, although side-channel vulnerabilities against Intel SGX have been documented [48], they do not affect the trust base of TrustFL+, as our design relies primarily on the integrity guarantees of TEEs rather than their confidentiality guarantees.

TABLE II
NOTATIONS

| Notation | Description |
|---|---|
| $(I, d, H)$ | (index, data content, HMAC) for one data |
| $D_i$ | data used for training segment $i$ |
| $\lambda$ | security parameter |
| $\theta_0^t$ | global model at the training epoch $t$ |
| $\theta_i^t$, $h_i^t$ | model parameter and its hash value at checkpoint $i$ |
| $R$ | the number of training segments at each epoch |
| $l$ | the number of sampled training segments |
| $v_i$ | random number in $[1,R]$ |
| $B$ | the number of training data in a batch |
| $N$ | the number of the total training data at one participant |
| $T$ | random nonce of enclave as the participant identifier |
| $\Phi$ | a program running in the enclave |
| hash() | one-way cryptographic hash function |
| SNG() | SGX random number generator |
| PRF() | pseudo-random function |
| DNN() | DNN training algorithm |
| $\Pi$ | existentially unforgeable signature scheme |
| log | the log recording rounding directions of local training |
| $b_{r_m}$ | the rounding precision |

## III. Design

This sections introduces the design overview and details. Table II lists all the main notations used in this paper.

### A. Design Overview

We initiate TrustFL+ in the context of a realistic scenario: a task owner seeks to obtain a high-quality DNN model but lacks sufficient local training data. He publishes a training task on an open data marketplace, offering a bounty to incentivize participation in a crowdsourced federated learning process. The end-to-end workflow includes six stages as shown in Fig. 3.

1) *Task Solicitation:* The task owner initiates a crowdsourced learning task on an open platform, calling for contributions from the public. It specifies the rewards and other instructions like setup tutorials. In this paper,

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

we assume the task owner maintains a central server $\mathcal{S}$ to coordinate the overall FL procedure. Notably, other decentralized alternatives like using blockchain [9], can adopt our scheme. In addition, similar to [49], we assume the training algorithm can be reviewed and checked by participants, so as to convince them that the training code would not steal their data.

2) *Participant Registration:* An interested participant $\mathcal{P}$ with appropriate data and an SGX-enabled platform can register itself for the task with necessary information like data description and identifier. Then, $\mathcal{P}$ reviews and downloads the program, launches an enclave with the program, and generates an attestation report to demonstrate the correct program setup. $\mathcal{S}$ verifies the correctness of the attestation and builds bidirectional connections to orchestrate successive processes.
3) *Participant Selection:* After the registration, an iterative learning procedure proceeds. At each epoch (e.g., epoch $t$), $\mathcal{S}$ selects a subset of participants to perform training tasks. In our system, we adhere to the strategy of original FL [50], where participants are randomly selected with a sampling probability.
4) *Local Training:* $\mathcal{P}$ receives the latest global model $\theta_0^t$, performs local training with the program $\mathsf{DNN}()$ and local data $D$ on a GPU. The training process yields local updates $\mathcal{L}_{\mathcal{P}}^t$ for the current epoch. During training, $\mathcal{P}$ periodically records intermediate model weights as checkpoints, which are later used to prove the integrity of the local training process and the validity of the resulting local updates.
5) *In-enclave Proof Generation:* $\mathcal{P}$ invokes the proof generation program inside the TEE. The enclave samples multiple pairs of recorded checkpoints and, for each pair, re-executes the gradient descent steps between them, starting from the earlier checkpoint. It then checks whether the reproduced model parameters match the later checkpoint. Only if all sampled segments pass the check, the enclave generates a signed attestation using $\Pi_{Sig}$, certifying the validity of GPU-generated local updates.
6) *Proof Verification and Model Aggregation:* Upon receiving local training results, the server $\mathcal{S}$ invokes the verification function $\Pi_{Ver}$ to determine whether participant $\mathcal{P}$ has performed the assigned training task honestly, with high confidence. If the verification passes, $\mathcal{S}$ distributes rewards to corresponding participants and aggregates these updates for the new global model as shown in (1). If the global model converges, $\mathcal{S}$ ends the learning task, otherwise, go back to the third stage.

In federated learning systems with lots of adversarial participants and diverse GPU architectures, TrustFL+ faces three critical vulnerabilities that prevent it from fully achieving all design goals described in Section I: 1) verification evasion due to sampling leakage, 2) freshness vulnerability under host-controlled TEE, and 3) verification fragility due to GPU-TEE float-point numerical divergence. We detail each of these vulnerabilities and present our countermeasures in the following subsections.

### B. Preventing Verification Evasion Due to Sampling Leakage

*Vul. #1. Verification Evasion Due to Sampling Leakage:* To verify whether a participant has correctly executed local training on the GPU, the TEE selectively re-executes segments of the training process between a pair of recorded checkpoints and compares the results against expected computations (i.e., Stage 5 in Fig. 3). This sampling-based verification offers a practical balance between efficiency and verification confidence. However, if sampling decisions are exposed to the participant beforehand, a malicious participant can exploit this by executing only the segments likely to be verified, while skipping or forging the rest. This vulnerability—referred to as sampling decision leakage—undermines the validity of the proof generated by the TEE, as the verified training segments may no longer represent the participant's full training behavior.

*CM. #1. Secure Sampling Design:* To mitigate sampling decision leakage, we adopt a commit-and-prove mechanism that requires participants to complete all local training segments and submit a cryptographic commitment to the TEE before any sampling decisions are made. This commitment acts as a binding snapshot of the training process and must satisfy two properties: 1) it must represent the results of all training segments in order, ensuring that the participant's behavior is finalized and cannot be modified post hoc; 2) it must support efficient verification of any selected segment by the TEE. As a result, the TEE can later sample and re-execute arbitrary training segments based on a fixed and tamper-proof execution trace, effectively preventing selective execution attacks.

In detail, we assume each training epoch is divided into $R$ segments, and the program outside the enclave (called $\mathsf{Prog_o}$) generates a commitment. Later, the enclave process called $\mathsf{Prog_i}$ verifies $k$ randomly selected segments to check training integrity. Especially, we offer two alternative commitment approaches, allowing participants to flexibly choose the appropriate commitment strategy based on their device capabilities (e.g., EPC size) and server-side verification requirements (e.g., number of segments to be verified).

- In the Sequential Hash-based (SH-based) approach, $\mathsf{Prog_o}$ computes a sequence of hash digests $\{h_0, h_1, \ldots, h_R\}$, where $h_i$ is the hash of the model parameters at the $i$-th checkpoint. This sequence is transmitted to $\mathsf{Prog_i}$ via a single Ecall during the commitment phase. During verification, $\mathsf{Prog_i}$ selects $k$ segments. For each selected segment $x$, $\mathsf{Prog_i}$ requests from $\mathsf{Prog_o}$ the necessary raw input, typically the model parameters at the beginning of segment $x$ (i.e., the checkpoint at $x-1\,(x>0)$). It then re-executes the same training computation as performed by $\mathsf{Prog_o}$ for that segment, producing an output model state. $\mathsf{Prog_i}$ hashes this output and compares it to the committed digest $h_x$. If all comparisons succeed, the segment is considered correctly executed.
- In the Merkle Tree-based (MT-based) method, $\mathsf{Prog_o}$ first computes a sequence of hash digests $\{h_0, h_1, \ldots, h_R\}$, where each $h_i$ corresponds to the hash of model parameters at the $i$-th checkpoint. These digests serve as the leaf nodes of a binary Merkle tree. The tree is constructed bottom-up: each internal node is computed by hashing the concatenation of its two child nodes. Padding is applied if the number of leaves is not a power of two. The root node $r$ serves as the final commitment and is transmitted to $\mathsf{Prog_i}$ via a single ECALL. During verification, $\mathsf{Prog_i}$ selects $k$ training segments. For each selected segment $x$, it requests from $\mathsf{Prog_o}$: the model parameters at the checkpoint $x-1, (x>0)$ as the input to the training segment; the Merkle proof corresponding to $h_x$ (a list of sibling hashes along the path to the root). $\mathsf{Prog_i}$ then re-executes the training segment $x$, hashes the output parameters $h'_x$,

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

reconstructs the Merkle root $r$ using $h'_x$ and the Merkle proof, compares the reconstructed root with the committed root. If all comparisons succeed, the segment is accepted.

Both the SH-based and MT-based commitment approaches offer the same level of security guarantee, grounded in the cryptographic properties of the hash function used, specifically, its collision resistance and second preimage resistance. The difference lies in how the commitments are structured and how verification is carried out. The SH-based method stores all $O(R)$ hash digests of checkpoint states in the enclave, allowing verification to proceed entirely within the trusted environment without additional runtime interaction. This makes it well-suited for the environments where enclave memory is sufficient and verification demands are high, such as when a large number of segments $k$ are selected for verification. In contrast, the MT-based method significantly reduces enclave memory usage by committing only the Merkle root, and defers proof transmission to the verification phase, incurring $O(k \log R)$ $\mathsf{Prog_o} - \mathsf{Prog_i}$ interaction. This strategy is more scalable with respect to the total number of segments $R$ and better suited for resource-constrained devices or scenarios with sparse verification needs (i.e., $k$ is small).

### C. Guaranteeing the Freshness of Training Results

*Vul. #2. Freshness Issues Under Host-Controlled TEE:* Since TEE's scheduling and I/O rely on the untrusted host, our scheme is hard to guarantee the freshness of a single training segment and final results. Specifically, the $\mathsf{Prog_o}$ may perform a small portion of training segments (e.g., $r$ segments) and replicate these to camouflage all training segments. The detection probability for such misbehavior is small and equals to $(1 - \binom{R-\lceil(\frac{R}{r})\rceil+1}{l} / \binom{R}{l})$ where $l$ is the number of random-selected segments. For example, if $R$ is 100, $r$ is 10, and $l$ is 5, the detection probability is about 38.23% which is quite low. Meanwhile, in the overall iterative FL procedure, the $\mathsf{Prog_o}$ may not invoke $\mathsf{Prog_i}$ to generate proofs, yet return previous training results and the corresponding proof to the task owner to defraud rewards. In addition, the participant may create dummy parties and mount sybil attacks for the purpose of over-claiming rewards.

*CM. #2. Dynamic-Yet-Deterministic Data Selection:* To ensure the freshness of training results, we leverage the dynamics of input training data, which further makes the results of any two segments or model updates of two epochs differentiable. In detail, we slightly modify the original data selection method in DNN algorithms ("shuffle then per-batch selection") [2] and use dynamic-yet-deterministic data selection in each training segment. To realize this, $\mathsf{Prog_i}$ should negotiate with $\mathsf{Prog_o}$ about data organization in advance. One naive method is to load all training data into the enclave, similar to existing TEE-protected machine learning works [49], [51], [52]. With the same data, $\mathsf{Prog_i}$ and $\mathsf{Prog_o}$ use the same selection function to determine the training data in each segment. However, this method may trigger frequent paging when fetching data to train from the entire dataset, and further degrade the performance of subsequent execution inside the SGX due to the shrunk memory space.

Our key observation is that, different from these prior arts [49], [52] protecting both confidentiality and integrity of data, we only require guaranteeing the data identifiability and integrity. Consequently, we retain all data on the outside of the enclave and perform efficient data preprocessing to meet our requirements. Specifically, $\mathsf{Prog_i}$ generates an index and HMAC for each

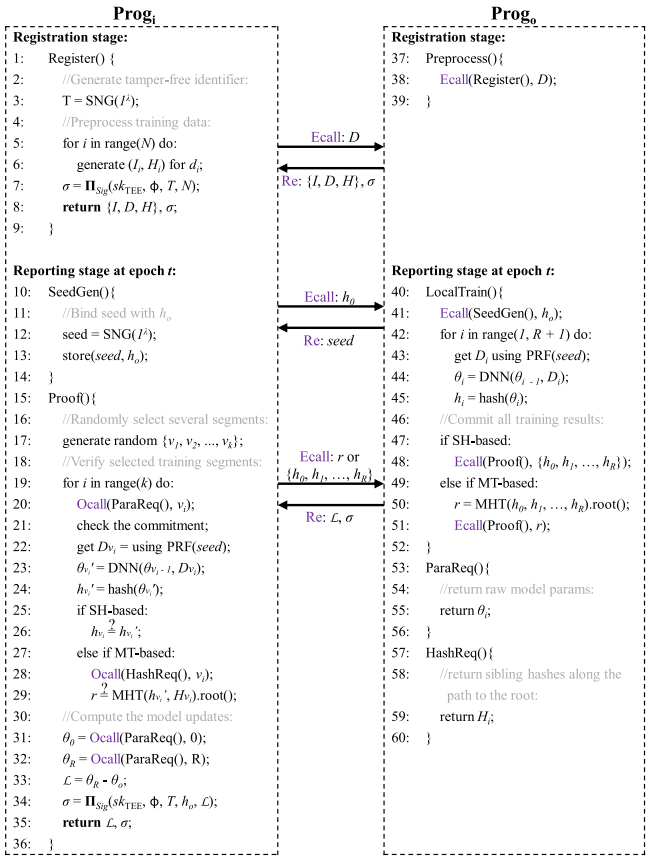



Fig. 4. The algorithm for assuring local training integrity with TEE. Local training is conducted by $\mathsf{Prog_o}$ at the host participant. Several randomly selected training segments are recomputed and checked by the $\mathsf{Prog_i}$.

training data which would be restored outside the enclave before training. When conducting local training, $\mathsf{Prog_o}$ requests a seed from $\mathsf{Prog_i}$ and each training data at each segment (e.g., $i$) is selected as follows:

$$I_{ij} = \mathsf{PRF}(seed \times i + j) \bmod N, \tag{2}$$

Where $I_{ij}$ is the index of the $j$th training data at segment $i$, PRF is a pseudo-random function, and $N$ is the number of total training data. Therefore, training data is different at each segment, and any replay attack is infeasible. When checking the correctness of selected training segments, $\mathsf{Prog_i}$ derives which data should be used, checks the integrity of the outside data with HMAC, trains the DNN, and verifies whether the output parameters are equal to the hash value generated by the outside.

Unfortunately, merely leveraging $\mathsf{Prog_i}$ may not be sufficient to attest to the freshness of the final model updates, as it is oblivious to whether the used global model is the latest. Here, we bind the training results with the used global model, and resort to the server to check the freshness of the training results. Before generating the $seed$, the $\mathsf{Prog_i}$ requires receiving the digest of the current global model (denoted as $h_0^t$) from the $\mathsf{Prog_o}$ (see lines 10-14 in Fig. 4). $h_0^t$ will be contained in the proof of TEE (see line 34 in Fig. 4). Notably, these operations implicitly bind the $h_0^t$ with the seed, and consequently bind with the final model updates. Finally, by checking the signature and the model digest, the server can ensure whether the participant used the latest model and faithfully performed the local training.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

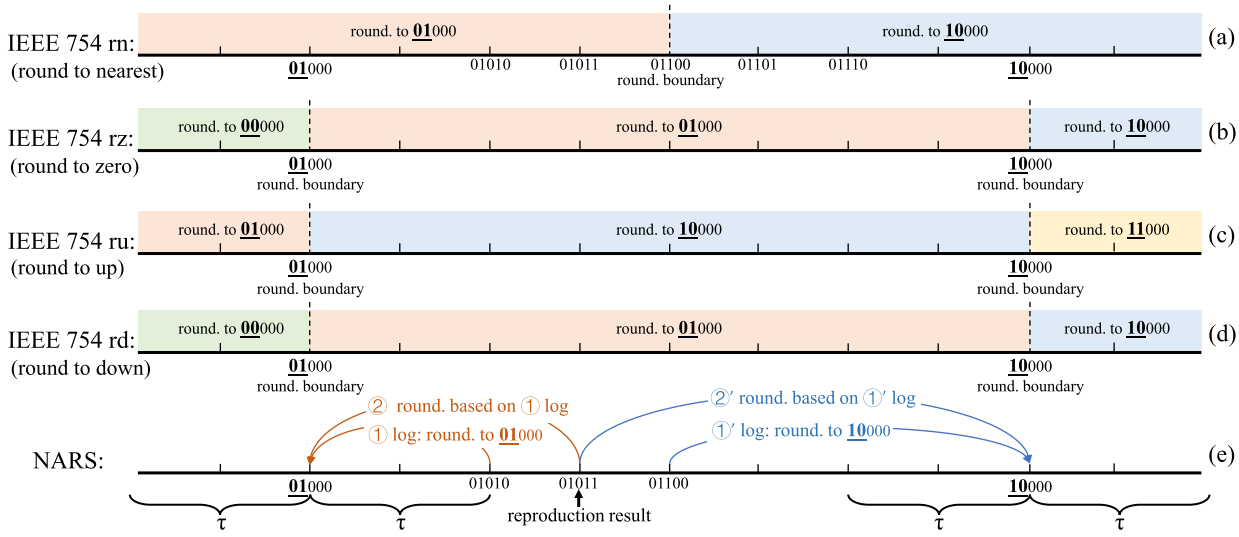



Fig. 5. Comparison between NARS and IEEE 754 rounding methods when rounding from 5 bits to 2 bits. IEEE 754 rounding methods always involve rounding boundaries, leading to potential discrepancies between devices. In contrast, NARS records the rounding direction of each computation, enabling bitwise-consistent reproduction under controlled rounding.

As for the sybil-based cheating where adversaries desire to use one training result to claim several-fold bounties at each epoch, we eliminate it by enforcing attackers to consume computing resources proportionally to the bounties. We easily use a unique random nonce $T$ produced by the enclave as the *tamper-free identifier* for the participant (this treatment is similar to [38]), as shown in line 3 of Fig. 4. In this way, even if an adversary registers multiple aliases, he must launch the same number of enclave instances to generate proofs for local training. If he wants to request multiple rewards using one set of training segments, he requires that all registered enclave instances produce valid proofs for this training set. That means, all enclave instances have to generate the same $seed$. This probability is extremely small which equals to $\frac{1}{2^{|seed|(m-1)}}$ where $m$ is the number of aliases. The overall procedure is shown in Fig. 4.

### D. Eliminating Impact of Hardware Non-Determinism

*Vul. #3. Verification Fragility Due to GPU-TEE Numerical Divergence:* In-enclave training results may differ from those produced on a GPU, even when using the same software version, the same random seeds, and the nondeterministic algorithms (via setting relevant library flags). That means, even if $\text{Prog}_\text{o}$ (performed on GPU) trains the model honestly, $\text{Prog}_\text{i}$ (performed inside the SGX) might generate a wrong proof suggesting that $\text{Prog}_\text{o}$ did not train the model faithfully, leading to non-negligible false positives and undermining the reliability of training assurance.

Prior work that handles hardware non-determinism in reproduction-based verification fails to ensure either security or scalability. As shown in Table I, one research line leverages a pre-defined tolerance threshold, i.e., checking whether distances between training results of $\text{Prog}_\text{i}$ and $\text{Prog}_\text{o}$ fall within an acceptable range, to tolerate reproduction errors caused by hardware non-determinism. TrustFL follows a similar strategy by converting model parameters into a lower-precision format and comparing their hash values, as detailed in Section V-B. Unfortunately, recent attacks [22], [23] demonstrate that such threshold-based verification can be exploited to forge training traces that fall within the threshold to the genuine training results, thereby bypassing verification and defrauding rewards. Specifically, the adversary first constructs a sequence of intermediate model checkpoints such that the parameter distance (e.g., Euclidean distance) between adjacent checkpoints remains within the verifier's tolerance threshold. For each checkpoint, the adversary keeps the model parameters fixed and adjusts the corresponding training data through adversarial optimization, so that the resulting training updates induce only negligible changes to the model parameters. As a result, when the verifier reproduces the training traces, the distance between the reproduced model and the forged model remains within the tolerance threshold, allowing the forged traces to pass verification. We note that the forged training trace can be constructed offline during local training, prior to any verification. Therefore, the short duration of each verification session in federated learning does not prevent these attacks.

Another line of research seeks to eliminate reproduction errors by rounding model parameters into lower precision representations. Traditional rounding modes specified in IEEE 754 [53], [54] are insufficient, as the same computation may yield slightly different results that straddle a rounding boundary and are rounded in opposite directions. For example, in Fig. 5, suppose device A outputs 01100 and device B reproduces 01011 after the same computation. If we round them from 5 bits to 2 bits, under the IEEE 754 rn mode, these two values lie on opposite sides of a rounding boundary: 01011 is rounded to 01000, whereas 01100 is rounded to 10000. This rounding discrepancy would lead to numerical divergence and subsequent failures in in-enclave proof generation. Similar problems exist for other rounding modes. To allow bitwise-consistent reproduction, one recent work (referred to as OVT in the paper) [17] proposes recording rounding directions of intermediate computations during training, then replaying the recorded rounding path in verification. However, this approach requires the FL server to determine appropriate rounding precisions by accessing GPUs of the same type as participants' GPUs. Therefore, it cannot scale well to large-scale federated learning systems.

*CM. #3. Novel Adaptive Rounding Strategy:* We propose a novel adaptive rounding strategy called NARS to ensure reliable

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

verification in large-scale federated learning across heterogeneous devices. Instead of requiring hardware-specific introspection, NARS enables the FL server to determine a rounding precision (denoted as $b_{r_m}$) by empirically balancing the floating-point error tolerance across diverse GPUs with the overall model precision. Given that FP32 is the typical training precision in real federated learning, each participant first converts model parameters from FP32 to FP64, then performs local training using a specific rounding precision while logging the rounding direction of each intermediate computation, and finally converts the trained parameters back from FP64 to FP32. The in-enclave proof is generated by replaying selected training segments based on the recorded logs, enabling bitwise-consistent reproduction under controlled rounding to confirm whether the participant has honestly executed the training. Note that, we adopt FP64 as the internal computation format because hardware non-determinism usually causes errors in several lowest bits of floating-point values. FP64 provides a much wider mantissa (52 bits) than FP32 (23 bits). After controlled rounding at a higher bit position and conversion back to FP32, the stored parameters exhibit no noticeable numerical drift.

The core of NARS lies in configuring the rounding precision $b_{r_m}$. $b_{r_m}$ defines the total number of high-order bits to be retained in the floating-point representation, including the 1-bit sign, 11-bit exponent, and the leading $(b_{r_m} - 12)$ bits of the mantissa. The remaining $(52 - (b_{r_m} - 12))$ low-order mantissa bits are zeroed out during rounding for each FP64 parameter. Note that we restrict $b_{r_m}$ to integer values in the range [12, 64], where the lower bound ensures full preservation of the sign and exponent fields, and the upper bound corresponds to retaining the full 64-bit FP64 representation without rounding.

Since $b_{r_m}$ should be low enough to eliminate hardware non-determinism and avoid false positives during in-enclave proof generation, yet high enough to preserve model performance (e.g., accuracy), it can be determined using the following objective:

$$b_{r_m} = \arg\max_{b_{r_m}} \{\alpha \mathsf{Tol}(b_{r_m}) + (1-\alpha)\mathsf{Per}(b_{r_m})\} \quad (3)$$

where $\mathsf{Tol}(b_{r_m})$ represents the reproduction error tolerance under the rounding precision $b_{r_m}$, and takes values in the range $[0, 1]$. A value of 1 indicates that the rounding precision fully eliminates floating-point discrepancies across GPU and TEE executions, allowing bitwise-consistent reproduction for all tested hardware configurations. Conversely, a value of 0 means that the precision fails to suppress hardware non-determinism, making accurate verification impossible even for honest participants. $\mathsf{Per}(b_{r_m})$ represents the model performance when training is conducted under the rounding precision $b_{r_m}$. $\alpha \in [0, 1]$ is a participant-defined weighting coefficient that balances hardware non-determinism elimination and model performance preservation. In practice, a larger $\alpha$ is suitable for models with complex structures or internal computations that may incur more hardware non-determinism errors, while a smaller $\alpha$ is preferred for models that are sensitive to numerical precision.

The two objectives $\mathsf{Tol}(b_{r_m})$ and $\mathsf{Per}(b_{r_m})$ are influenced by distinct factors. $\mathsf{Tol}(b_{r_m})$ primarily depends on the heterogeneity of GPU hardware: lower $b_{r_m}$ values (i.e., coarser precision) tend to suppress more sources of hardware non-determinism. In contrast, $\mathsf{Per}(b_{r_m})$ is closely related to the magnitude of gradient updates during training. If $b_{r_m}$ is set too low, i.e., retaining too few bits in the floating-point representation, small but meaningful gradient updates may be rounded to zero, especially when the model approaches convergence. For example, when the minimal step size enforced by rounding is $2^{-6}$ but most updates are on the order of $2^{-10}$, such updates are lost, effectively stalling model convergence. Regarding this, we observe that the magnitude of model updates typically decreases as training progresses. That is, large updates dominate in the early stages (called the initial training phase), while fine-grained adjustments become critical near convergence (called the convergence phase). Accordingly, NARS assigns a separate $b_{r_m}$ to each phase:

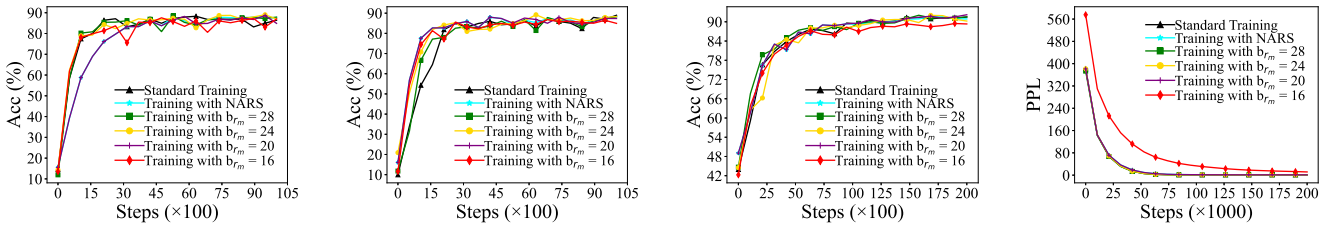
- *Initial training phase:* In this phase, $b_{r_m}$ is published before training. At this stage, we lack model updates with which to assess the relationship between $b_{r_m}$ and model performance, making it impossible to quantify $\mathsf{Per}(\mathsf{b_{r_m}})$ or solve for $b_{r_m}$ via (3). Fortunately, during the initial training phase, low precision has a negligible effect on performance, as mentioned in [17], [55], [56], meaning that $\mathsf{Per}(\mathsf{b_{r_m}})$ remains constant. Consequently, we can simply select a small $b_{r_m}$ at this stage, since $\mathsf{Tol}(\mathsf{b_{r_m}})$ grows as $b_{r_m}$ decreases. In practice, we found that, for the devices and models employed in this study, $b_{r_m} \leq 20$ (mantissa bits less than 8) is sufficient to eliminate the impact of hardware non-determinism during the initial training phase. In our experiments, the rounding was relatively conservative, while other related studies suggest that $b_{r_m} \leq 28$ (mantissa less than 16) is sufficient [17].
- *Convergence phase:* When training approaches convergence (e.g., after a predefined step threshold or when the loss falls below a target value), each participant locally selects a finer-grained rounding precision $b_{r_m}$ based on the magnitude of its unrounded recent parameter updates. The participant then continues training under this $b_{r_m}$ until convergence. The same $b_{r_m}$ is used by the co-located TEE when verifying sampled training segments in this phase. We highlight that although the participant may indeed be untrusted, they cannot manipulate $b_{r_m}$ to bypass TEE verification or gain an unfair advantage. This is because choosing an overly large $b_{r_m}$ makes rounding insufficient to eliminate hardware non-determinism errors and thus fails verification. Conversely, choosing an overly small $b_{r_m}$ increases error tolerance but degrades model performance as shown in Figs. 6(d) and 7(d). In practice, since FL rewards are typically performance-based [57], participants are incentivized to choose $b_{r_m}$ that preserves model performance while remaining verifiable. Finally, this design eliminates the need for the FL server to maintain hardware environments similar to those of participants in order to determine $b_{r_m}$, thereby reducing operational complexity and facilitating deployment in lightweight or cost-sensitive FL settings.

Specifically, we can set $\mathsf{Tol}(b_{r_m})$ and $\mathsf{Per}(\mathsf{b_{r_m}})$ as two sigmoid functions due to their smooth, S-shaped curve:

$$\begin{cases} \mathsf{Tol}(b_{r_m}) = \frac{1}{1+e^{(b_{r_m}-b_T)}}, \\ \mathsf{Per}(b_{r_m}) = \frac{1}{1+e^{-(b_{r_m}-b_P)}}, \end{cases} \quad (4)$$

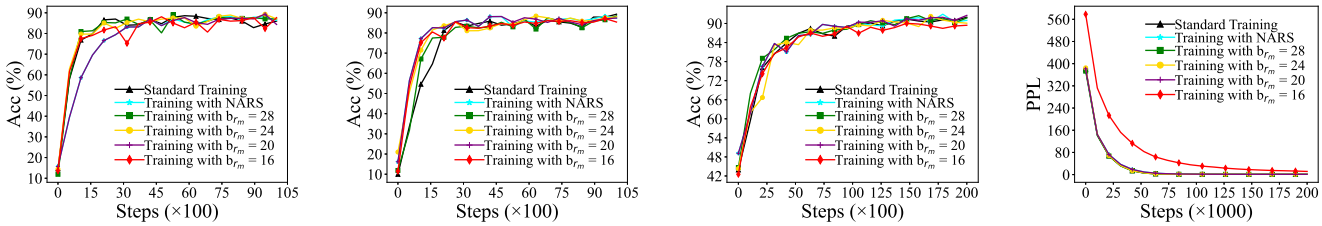
where $b_T$ represents the $b_{r_m}$ at which the error tolerance undergo significant changes while $b_P$ represents the $b_{r_m}$ at which the model performance begins to change notably. This choice is motivated by [17], [56], which demonstrates that in low-precision training, where fewer bits are used to represent model

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.




(a) VGG16 (b) VGG19 (c) ResNet50 (d) GPT2

Fig. 6. Evaluation curves of four models trained on NVIDIA RTX 4090. The classification models are evaluated using accuracy (Acc, %), while GPT2 is evaluated using perplexity (PPL). One step corresponds to a single batch update.




(a) VGG16 (b) VGG19 (c) ResNet50 (d) GPT2

Fig. 7. Evaluation curves of four models trained on NVIDIA A100. The classification models are evaluated using accuracy (Acc, %), while GPT2 is evaluated using perplexity (PPL). One step corresponds to a single batch update.

parameters, the gain in error tolerance from each additional bit initially increases and then decreases, while the impact on performance follows the opposite trend. Additionally, the boundedness of sigmoid functions makes them ideal for modeling these behaviors in a controlled manner.

Precisely locating the inflection points ($b_T$ and $b_P$) of the two sigmoid functions is critical. For $\mathsf{Tol}(b_{r_m})$, we can directly adopt $b_T = 28$. The setting is inspired by a recent work [17]. It shows that directly rounding FP64 trained model parameters to mantissa bits smaller than the 16th bit ($b_{r_m} \leq 28$) eliminates most hardware non-determinism errors, whereas rounding beyond the 16th bit ($b_{r_m} > 28$) introduces large errors that increase with the rounding position. Conversely, $b_P$ is data-driven. NARS selects the largest updates (top decile of absolute changes over multiple segments) and sets $b_P$ to the minimum precision required to represent them. This strategy of prioritizing major updates is grounded in the finding that revising only the largest 10% of gradient updates is sufficient to preserve model performance [58].

## IV. Theoretical Analysis

### A. Security Analysis

We prove the correctness and soundness of our scheme.

*Theorem 1 (Correctness):* The scheme is said to be correct, if for all key pairs $(sk_{\mathsf{TEE}}, pk_{\mathsf{TEE}})$ generated by enclaves, all training algorithm $\mathsf{DNN}()$, all dataset $D$, all global model $\theta_0$, all model updates $\mathcal{L}_{\mathcal{P}}$ produced by $\mathsf{DNN}(\theta_0, D)$ under the server-specified rounding precision $b_{r_m}$, we obtain $\mathsf{Verify}(pk_{\mathsf{TEE}}, \mathsf{DNN}(), \theta_0, \mathcal{L}_{\mathcal{P}}, \sigma_{\mathcal{P}}, b_{r_m}) \rightarrow 1$ with high probability, where $\sigma_{\mathcal{P}}$ is produced by $\mathsf{Prove}(sk_{\mathsf{TEE}}, \mathsf{DNN}(), D, \theta_0, \mathcal{L}_{\mathcal{P}}, b_{r_m})$.

*Proof:* The correctness of our scheme depends on the correctness of SGX and the reproduction error tolerance afforded by the selected $b_{r_m}$. If the participant honestly performs all training segments under the server-specified rounding precision $b_{r_m}$, then for any randomly selected training segments, $\mathsf{Prog}_{\mathsf{i}}$ in the TEE can reproduce model parameters that match those generated by $\mathsf{Prog}_{\mathsf{o}}$ with high probability. Consequently, their hash digests will also match. Once all sampled segments are successfully verified, the TEE attests to the integrity of the entire training process by producing a signature, which is then verified by the server using SGX's attestation functionality.

We frame the correctness guarantee probabilistically rather than deterministically, as $b_{r_m}$ may fail to fully eliminate hardware non-determinism in rare edge cases. In fact, $b_{r_m}$ can only be determined empirically, as it is infeasible to exhaustively test GPU–TEE reproduction errors for all possible models and hardware configurations. To ensure broad compatibility, NARS adopts a conservative strategy by selecting a $b_{r_m}$ value lower than those used in related works [17], thereby tolerating a wider range of floating-point deviations across heterogeneous GPUs. Importantly, our experiments show that this conservative precision maintains model performance. In Section VI, we further discuss how NARS handles rare cases where certain devices still exhibit reproduction errors beyond the selected $b_{r_m}$. ■

*Theorem 2 (Soundness):* The scheme is said to be sound, if for all key pairs $(sk_{\mathsf{TEE}}, pk_{\mathsf{TEE}})$ generated by enclaves, all training algorithm $\mathsf{DNN}()$, all dataset $D$, all global model $\theta_0$, any probabilistic polynomial-time adversary Adv, the following inequality holds,

$$\Pr\left(\begin{array}{c}\{\mathcal{L}^*_{\mathcal{P}}, \sigma_{\mathcal{P}}\} \leftarrow \mathsf{Adv}(pk_{\mathsf{TEE}}, \mathsf{DNN}(), D, \theta_0); \\ 1 \leftarrow \mathsf{Verify}(pk_{\mathsf{TEE}}, \mathsf{DNN}(), \theta_0, \mathcal{L}^*_{\mathcal{P}}, \sigma_{\mathcal{P}}); \\ \mathcal{L}^*_{\mathcal{P}} \neq \mathcal{L}_{\mathcal{P}}\end{array}\right) \leq \epsilon. \quad (5)$$

where $\mathcal{L}_{\mathcal{P}}$ is the correct result of $\mathsf{DNN}(\theta_0, D)$, $\epsilon$ is the soundness error and less than $(\frac{r}{R} + (1 - \frac{r}{R})q)^l$, $r$ is the number of honestly training segments of Adv, $l$ is the number of randomly selected

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

segments, $q$ is the probability that Adv can successfully give a qualified input model parameters without training.

*Proof:* In TrustFL+, verification is performed on randomly sampled training segments rather than on entire training runs. For each sampled segment, the verifier re-executes the computation and checks whether the result matches the original execution in a bitwise-identical manner. If an adversary wants to prove its honesty on all training segments, for each sampled training segment, he must give the correct input parameters to $\mathsf{Prog_i}$. It means that he either performs the selected segment or successfully guesses the qualified parameters before committing training results to $\mathsf{Prog_i}$. This property is held by the security of one-way hash functions used in MHT construction and commitment verification, as demonstrated in [59].

Following prior sampling-based verification works [16], [60], the detection probability of a bad training run can be characterized theoretically from the sampling process, without relying on empirical end-to-end run-level statistics.

Randomly sampling $l$ segments from the total $R$ training segments, there are $\binom{R}{l}$ choices. Given $(R-r)$ segments without training, the expectation that $\mathsf{Prog_o}$ can successfully cheat equals to $\sum_{i=0}^{R-r}\binom{r+i}{l}\binom{R-r}{i}q^n(1-q)^{R-r-i}$. Consequently, the cheating probability $P_c$ (i.e., the soundness error $\epsilon$) is:

$$\epsilon = \frac{\sum_{i=0\max(l-r,0)}^{R-r}\binom{r+i}{l}\binom{R-r}{i}q^i(1-q)^{R-r-i}}{\binom{R}{l}}. \tag{6}$$

The above $P_c$ is derived under the condition where $l$ samples selected from $R$ segments are mutually different, thus, it is less than that calculated where $l$ samples can be same in $[1, R]$, as follows:

$$\epsilon < \left(\frac{r}{R} + \left(1-\frac{r}{R}\right)q\right)^l. \tag{7}$$

■

Given a soundness error $\epsilon$, we can derive the lower bound of the number of samples:

$$l \geq \frac{\log\epsilon}{\log(\frac{r}{R} + (1-\frac{r}{R})q)}. \tag{8}$$

As an example, if $\epsilon$ is 1%, $\mathsf{Prog_o}$ faithfully executes 100 training segments from 1000 segments. Note that while one DNN model usually maintains tens of thousands to millions of parameters, we assume that $q$ is 0.1% which is larger than the actual value. We only need 2 samples. If $\mathsf{Prog_o}$ performs 90% of training segments, we require 43 samples to guarantee the soundness.

### B. Processing Time Analysis

We theoretically analyze the processing time of our scheme and compare it with training on GPU, CPU, and SGX. Assume the time of training one batch of data on GPU, CPU, and SGX is $t_{\mathsf{GPU}}$, $t_{\mathsf{CPU}}$, $t_{\mathsf{SGX}}$, respectively. If performing $R$ segments of training, the total time on GPU, CPU, and SGX is about $T_{\mathsf{GPU}} = R \cdot t_{\mathsf{GPU}}$, $T_{\mathsf{CPU}} = R \cdot t_{\mathsf{CPU}}$, and $T_{\mathsf{SGX}} = R \cdot t_{\mathsf{SGX}}$. In TrustFL, the total processing time is $T_{\mathsf{TFL}} = (R \cdot t_{\mathsf{GPU}} + l(t_{\mathsf{M}} + t_{\mathsf{SGX}} + t_C))$. In TrustFL+, the total processing time is $T_{\mathsf{TFL+}} = (R \cdot t'_{\mathsf{GPU}} + l(t_{\mathsf{M}} + t'_{\mathsf{SGX}} + t_C))$. $t_{\mathsf{M}}$ is the model initializing time in SGX, $t_C$ is the time of verifying the commitment. $t'_{\mathsf{GPU}}$ and $t'_{\mathsf{SGX}}$ represent the total training time on the GPU and SGX, respectively, including the additional cost introduced by NARS: $t'_{\mathsf{GPU}} = t_{\mathsf{GPU}} + t_{\mathsf{GPU_{NARS}}}$, $t'_{\mathsf{SGX}} = t_{\mathsf{SGX}} + t_{\mathsf{SGX_{NARS}}}$, where $t_{\mathsf{GPU}}$ and $t_{\mathsf{SGX}}$ are the normal training times without NARS, while $t_{\mathsf{GPU_{NARS}}}$ and $t_{\mathsf{SGX_{NARS}}}$ represent the overhead added by NARS on each platform.

$t_{\mathsf{GPU_{NARS}}}$ and $t_{\mathsf{SGX_{NARS}}}$ primarily consist of three components: the time cost of rounding forward-pass outputs ($t_{D_r}$), the time cost of rounding gradients produced during backpropagation ($t_{G_r}$), and the time cost of rounding the final model parameter updates ($t_{P_r}$). Since $t_{\mathsf{GPU_{NARS}}}$ and $t_{\mathsf{SGX_{NARS}}}$ consist of the same components, we focus our discussion on $t_{\mathsf{GPU_{NARS}}}$ for clarity. $t_{G_r}$ and $t_{P_r}$ are determined solely by the number of model parameters. Larger models require rounding more gradients and parameters, resulting in higher $t_{G_r}$ and $t_{P_r}$. Therefore, for a given model, $t_{G_r}$ and $t_{P_r}$ are constant. On the other hand, $t_{D_r}$ depends only on the batch size. A larger batch size produces more intermediate results during the forward pass, which increases the rounding overhead. Since $t_{D_r}$ is associated with rounding operations on each forward-pass output, it scales in proportion to the batch size, similar to the overall training time $t_{\mathsf{GPU}}$. As a result, $t_{D_r}$ remains a smaller multiple of $t_{\mathsf{GPU}}$. We can approximate: $t'_{\mathsf{GPU}} = t_{\mathsf{GPU}} + t_{\mathsf{GPU_{NARS}}} = t_{\mathsf{GPU}} + t_{D_r} + t_{G_r} + t_{P_r} \approx k \cdot t_{\mathsf{GPU}} + t_{G_r} + t_{P_r}$ ($k$ is a constant here). Since $t_{G_r}$ and $t_{P_r}$ are fixed for a given model, and $t_{\mathsf{GPU}}$ increases with larger batch size, using a larger batch size reduces the relative impact of the NARS-induced overhead, making $t'_{\mathsf{GPU}}$ approach $k \cdot t_{\mathsf{GPU}}$. As a result, $T'_{\mathsf{GPU}}$ approaches $k \cdot T_{\mathsf{GPU}}$, where $k$ remains relatively small. This indicates that NARS introduces only minimal overhead, especially when the batch size is large.

$T_{\mathsf{TFL+}}$ includes two parts: $T'_{\mathsf{GPU}}$ (i.e., $R \cdot t'_{\mathsf{GPU}}$) and $T'_{\mathsf{SGX}}$ (i.e., $l(t_{\mathsf{M}} + t'_{\mathsf{SGX}} + t_C)$). Particularly, while $l$ is constant for a given soundness error and honest ratio (i.e., $\frac{r}{R}$) (see (8)), $T'_{\mathsf{SGX}}$ is constant for varied $R$ ($R > l$). Therefore, as $R$ grows, $T_{\mathsf{TFL+}}$ is first dominated by $T'_{\mathsf{SGX}}$, then by $T'_{\mathsf{GPU}}$. It means that, for a large $R$, TrustFL+ can achieve a processing time comparable to GPU-only training, which can be expressed as $T_{\mathsf{TFL+}} \approx T'_{\mathsf{GPU}}$. Similarly, for standard TrustFL without NARS, $T_{\mathsf{TFL}} \approx T_{\mathsf{GPU}}$. If training on a GPU is approximately two orders of magnitude faster than on SGX, then TrustFL can achieve up to 100× speedup over pure SGX execution. In the case of TrustFL+, since $T'_{\mathsf{GPU}} \approx k \cdot T_{\mathsf{GPU}}$, the overall speedup becomes approximately $\frac{100}{k}\times$, where k is a small constant introduced by NARS overhead. We demonstrate this in Section V-C.

## V. Implementation and Evaluation

In this section, we conduct extensive experiments to assess the effectiveness of TrustFL+ with the following evaluation goals:

- *Goal 1:* While TrustFL+ improves TrustFL [21] by introducing NARS, we first evaluate how NARS affects neural network model performance and whether it advances TrustFL in terms of handling false positives induced by hardware non-determinism under varied GPUs and neural network models (see Section V-B).
- *Goal 2:* We estimate the efficiency of participant-side proof-of-training, including in-enclave training time and the total time with TrustFL and TrustFL+ setup (see Section V-C).
- *Goal 3:* We evaluate the scalability of TrustFL+ throughout the iterative federated learning process with varied numbers of participants (see Section V-D).

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

### A. Implementations and Experimental Setup

*Implementation of TrustFL+:* We implement the DNN training algorithm outside the enclave called $\mathsf{Prog}_o$, with Python based on the Pytorch framework[3] and secure hash library.[4] We use Occlum [25] which is a memory-safe, multi-process library OS (LibOS) for Intel SGX. We deployed DNNs on Occlum.

In our prototype, we simplify the verification procedure by using the signature scheme built upon Crypto++,[5] instead of fully following the SGX remote attestation mechanism. The communication channels between the task owner and participants are built via TCP socket.

To implement NARS, we inject PyTorch hooks (custom functions that can be triggered during the forward and backward passes of a model) to round the results of each forward pass, the gradients of each backward pass, and the final model parameter updates, while recording the rounding logs. During reproduction, the injected hook examines the rounding logs to determine the appropriate rounding direction. All programs are developed using Python version 3.8 and PyTorch version 2.4.1.

*Experimental Setup:* In this experiment, we employed six different NVIDIA GPUs—RTX 4090, RTX 3090, RTX 2060, H20, A100, and V100. The TEE was hosted on Intel(R) Xeon(R) Platinum 8369B 2.70 GHz CPU with 64 GB RAM. In addition to assessing the effectiveness and performance of TrustFL+ on the aforementioned hardware, we simulated a server cluster with 20 clients and a single server to evaluate the scalability of our scheme.

We implemented three types of neural network models with varied architectures:

- *Image Processing Models:* We used VGG16 (138 M parameters), VGG19 (144 M parameters), and ResNet50 (25.6 M parameters). These convolutional networks are widely used for image classification and are trained on CIFAR-10 [61] (160 MB of 32×32-pixel images, which are upsampled to 224× 224 pixels for VGG16 and VGG19).
- *Natural Language Processing Model:* GPT2 (117 M parameters), a transformer-based model for tasks such as language modeling and text generation. It is trained on Wikitext2 [62], a widely adopted corpus.
- *Toy Model:* A lightweight model (0.06 M parameters) with two convolutional layers, one pooling layer, and three fully connected layers. It is trained on MINST [63].

### B. Effectiveness Evaluation of NARS

In this section, we assess the effectiveness of NARS. We first investigate the impact of different values of $b_{r_m}$ on neural network model performance, and then estimate NARS's ability to suppress false positives caused by hardware non-determinism.

*Impact of NARS on model performance:* To assess the impact of NARS on model performance, we trained VGG16, VGG19, ResNet50, and GPT2 to convergence under various $b_{r_m}$ settings, as well as under the NARS configuration (described below). The experiments are conducted on two representative GPUs: the consumer-class RTX 4090 and the data-center–class A100, and report the corresponding test-set performance trajectories in terms of accuracy (Acc) and perplexity (PPL). Regarding the NARS configuration, we set $\alpha$ in (3) to 0.3, which reflects a design choice that prioritizes model performance over aggressively eliminating hardware non-determinism errors (i.e., $\alpha < 0.5$). We initialize $b_{r_m}$ to 20in the initial training phase. These settings are chosen in accordance with the discussion in Section III-D. We define the training to enter the convergence phase after a predetermined number of steps, specifically, $5 \times 10^3$ steps for VGG16 and VGG19, $10^4$ steps for ResNet50, and $10^5$ steps for GPT2. The adaptively obtained $b_{r_m}$ values at the convergence phase for the four models are as follows: 22 (VGG16, RTX 4090), 22 (VGG16, A100), 22 (VGG19, RTX 4090), 22 (VGG19, A100), 23 (ResNet50, RTX 4090), 23 (ResNet50, A100), 25 (GPT2, RTX 4090), and 26 (GPT2, A100).

Figs. 6 and 7 depict test-set performance curves with training steps under different $b_{r_m}$ settings. Classification performance is measured by accuracy (Acc), while GPT2 is evaluated using perplexity (PPL). When training with fixed $b_{r_m}$, for $b_{r_m}$ values of 20 and above, all models attain good performance with similar convergence behavior. However, an excessively small mantissa width ($b_{r_m} = 16$) may lead to slower convergence and degraded final performance. Specifically, VGG16 and VGG19 remain largely unaffected, achieving final accuracies comparable to standard training. In contrast, ResNet50 shows a degradation in final accuracy. For GPT2, the impact of $b_{r_m} = 16$ is more pronounced, leading to substantially higher perplexity throughout training. Despite these differences, all models exhibit clear performance improvements during the early phases of training even when $b_{r_m} = 16$, indicating that training remains effective at low precision in the initial phase. Moreover, models trained with NARS consistently achieve performance comparable to standard training, with almost no observable degradation.

These results indicate that the impact of $b_{r_m}$ varies across models. In particular, models with more complex architectures, such as ResNet50 and GPT2, require larger $b_{r_m}$ values to match standard-training performance, whereas VGG16 and VGG19 show little performance degradation even at small $b_{r_m}$ values. This difference can be mainly attributed to two architectural factors: (1) deep gradient propagation paths make the differences between rounded and unrounded values caused by small $b_{r_m}$ more likely to accumulate and amplify across layers, especially in Transformer modules with self-attention and numerous softmax operations; (2) residual connections allow such differences to persist throughout the network, ultimately leading to performance degradation.

*Reliability evaluation of NARS:* To evaluate whether NARS introduced in TrustFL+ effectively addresses the false positive issue present in TrustFL, we reimplemented the original TrustFL framework. It is important to note that TrustFL converts parameter data types from high to low precision before hashing to mitigate the impact of hardware non-determinism. In our evaluation of TrustFL, models were trained using FP64 precision, and the final parameters were cast to FP32 and FP16 formats to assess false positive rates. We conducted experiments on a range of GPU devices (RTX 4090, RTX 3090, RTX 2060, H20, A100, and V100), training several models (toy model, VGG16, VGG19, ResNet50, and GPT2). A total of 200 checkpoints were saved and verified inside a TEE.

Table III reports the false positive rate (FPR) of all checkpoints under both methods, where FPR denotes the proportion of verification failures due to reproduction errors. Here, "verification failure" means that the reproduced model parameters are not bitwise identical to those obtained from the original

[3]http://www.pytorch.org/
[4]https://docs.python.org/3/library/hashlib.html
[5]https://www.cryptopp.com/

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

TABLE III
FALSE POSITIVE RATES OF TEE VERIFICATION FOR GPU TRAINING WHEN USING TRUSTFL AND TRUSTFL+

| Methods | Models | RTX 4090 | RTX 3090 | RTX 2060 | H20 | A100 | V100 |
|---|---|---|---|---|---|---|---|
| TrustFL[1] FP64 to FP64 | Toy Model | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | VGG16 [64] | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | VGG19 [64] | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | ResNet50 [65] | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | GPT2 [66] | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| TrustFL[2] FP64 to FP32 | Toy Model | 0 | 0 | 0 | 0 | 0 | 0 |
| | VGG16 [64] | 0.74 | 0.71 | 0.44 | 0.63 | 0.64 | 0.73 |
| | VGG19 [64] | 0.90 | 0.98 | 0.84 | 0.89 | 0.93 | 0.91 |
| | ResNet50 [65] | 0.39 | 0.45 | 0.67 | 0.62 | 0.64 | 0.48 |
| | GPT2 [66] | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| TrustFL[3] FP64 to FP16 | Toy Model | 0 | 0 | 0 | 0 | 0 | 0 |
| | VGG16 [64] | 0 | 0 | 0 | 0 | 0 | 0 |
| | VGG19 [64] | 0 | 0 | 0 | 0 | 0 | 0 |
| | ResNet50 [65] | 0 | 0 | 0 | 0 | 0 | 0 |
| | GPT2 [66] | 0 | 0 | 0 | 0 | 0 | 0 |
| TrustFL+ | Toy Model | 0 | 0 | 0 | 0 | 0 | 0 |
| | VGG16 [64] | 0 | 0 | 0 | 0 | 0 | 0 |
| | VGG19 [64] | 0 | 0 | 0 | 0 | 0 | 0 |
| | ResNet50 [65] | 0 | 0 | 0 | 0 | 0 | 0 |
| | GPT2 [66] | 0 | 0 | 0 | 0 | 0 | 0 |

[1] Reproduce in FP64 and compare the results with the training results without format converting.
[2] Reproduce in FP64 and compare the results with the training results after converting both to FP32.
[3] Reproduce in FP64 and compare the results with the training results after converting both to FP16.

training. As shown, under TrustFL with FP64 training and no data type conversion (FP64 to FP64), all models on all devices exhibit an FPR of 1.00. This result indicates that reproduction errors under FP64 format are pervasive. When FP64-trained parameters are converted to FP32, the average FPR is 0 for toy model, 0.65 for VGG16, 0.91 for VGG19, 0.54 for ResNet50, and 1.00 for GPT2. This indicates that reproduction errors tend to increase with model size and architectural complexity. When the FP64-trained parameters are converted to FP16, all models consistently pass verification across all devices, indicating that aggressively reducing numerical precision can absorb low-bit perturbations introduced by hardware non-determinism. In contrast, TrustFL+ effectively resolves this issue, enabling reliable verification of models trained on different hardware. Notably, the false positive rate is reduced to 0 for all models on all devices when TrustFL+ is applied. The above results demonstrate that with the introduction of NARS, TrustFL+ effectively eliminates hardware non-determinism errors and enables reliable verification across varied models and heterogeneous GPUs.

*Sensitivity Analysis of weighting factor $\alpha$:* To study the sensitivity of TrustFL+ to $\alpha$, we vary the weighting factor $\alpha$ from 0.1 to 0.9 and train the models until convergence. For vision models (VGG16, VGG19, and ResNet50), we report the classification accuracy (ACC) as the model performance, while we report the perplexity (PPL) for GPT2. To measure the reproduction error tolerance under varied $\alpha$, we estimate the false positive rate (FPR) of the verification process by randomly sampling 200 training steps under each configuration. The results are summarized in Table IV.

From the results, we observe that varying $\alpha$ slightly affects model performance. Specifically, models that are more sensitive to parameter precision, such as ResNet50 and GPT2, show a slight improvement in performance as $\alpha$ decreases, from 0.89 to 0.91 for ResNet50. In contrast, VGG16 and VGG19 exhibit almost no observable variation in accuracy across the entire $\alpha$ range. More importantly, the verification process consistently achieves an FPR of 0 for all tested configurations. This indicates that TrustFL+ can reliably eliminate hardware non-determinism errors under varied $\alpha$.

TABLE IV
SENSITIVITY OF WEIGHTING FACTOR $\alpha$ ON MODEL PERFORMANCE AND VERIFICATION RELIABILITY

| Parameter | Range | Models | Acc or PPL (Range) | FPR (Range) |
|---|---|---|---|---|
| $\alpha$ | [0.1, 0.9] | VGG16 | [0.86, 0.86] | [0, 0] |
| | | VGG19 | [0.87, 0.88] | [0, 0] |
| | | ResNet50 | [0.89, 0.91] | [0, 0] |
| | | GPT2 | [8.6, 9.1] | [0, 0] |

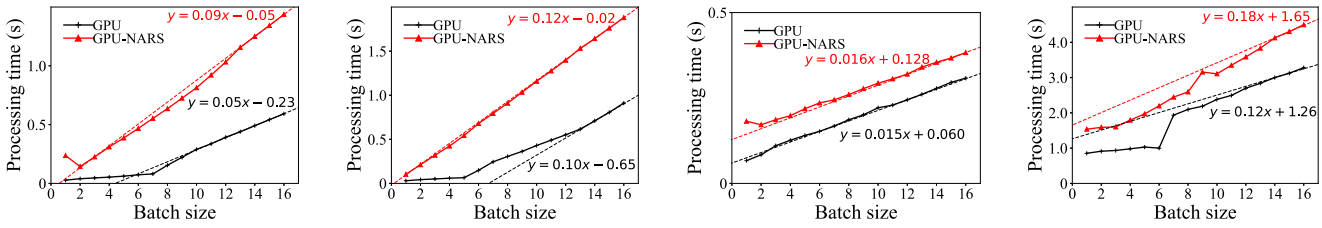
### C. Efficiency Evaluation of Participant-Side Proof-of-Training

We evaluate the processing time at participant side, including the runtime of GPU-side training, in-enclave training, and the overall time efficiency gains of TrustFL+.

*GPU-side training processing time:* To evaluate the training time of TrustFL+ on the GPU side, we compared the NARS-integrated training time with the original training time on RTX 4090 GPU. The evaluation was conducted using four models: VGG16, VGG19, ResNet50, and GPT2.

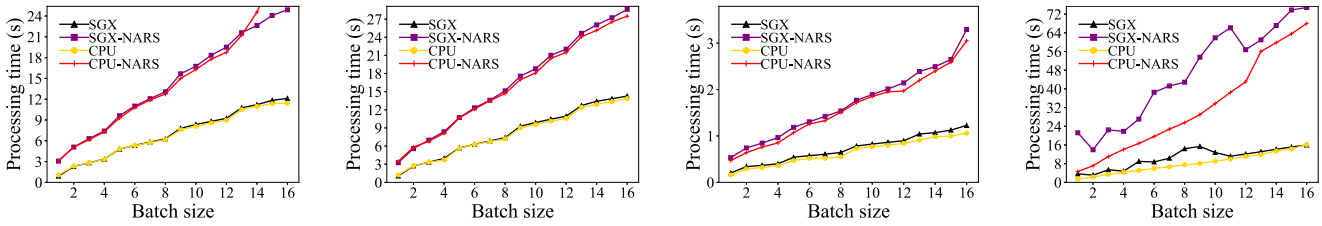
Fig. 8 compares the NARS-integrated training time and the original training time across different batch sizes. The introduction of NARS results in additional processing time, which

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.




(a) VGG16 (b) VGG19 (c) ResNet50 (d) GPT2

Fig. 8. One step training time under different batch sizes for four models, comparing the use of NARS versus no NARS on GPU.




(a) VGG16 (b) VGG19 (c) ResNet50 (d) GPT2

Fig. 9. One step training time under different batch sizes for four models, comparing execution inside and outside the enclave.

is reflected in the figure by the GPU-NARS curve consistently lying above the GPU curve. As the batch size increases, both curves begin to resemble linear functions, indicating that the relative overhead gradually decreases and eventually stabilizes at a fixed proportion. We observed that when the batch size is set to 1, the training time per step increases by 758%, 241%, 176%, and 81% for VGG16, VGG19, ResNet50, and GPT2, respectively. In contrast, with a batch size of 16, the overhead is reduced to 140%, 105%, 23%, and 36%, respectively. This observation aligns with our analysis in Section IV-B.

Furthermore, we measured the time required to construct a Merkle tree as described in Section III-B. For example, building a Merkle tree with $10^2$, $10^3$, and $10^4$ leaf nodes require $1.71 \times 10^{-3}$s, $1.81 \times 10^{-2}$s, and $2.34 \times 10^{-1}$s, respectively. Therefore, the time required to construct the Merkle tree is minimal.

*In-enclave training processing time:* To evaluate the processing time of executing DNN programs inside the enclave, we measure the time required for model initialization and DNN training across various models. Fig. 9 shows the comparison of training time between SGX and CPU across different batch sizes. The results show that, in our experiments, using SGX introduces minimal additional time overhead compared to running directly on a CPU, even when applying the rounding method used in NARS. Notably, the variation of GPT2's training time with respect to batch size within SGX shows some deviation compared to other models, although the overall trend remains similar, which may be due to Occlum's memory management mechanisms [25].

*Evaluation of overall participant-side processing time:* To evaluate the overall efficiency of TrustFL+ at participant sides in one training epoch, we compare its total processing time with that of training on an RTX 4090 GPU, CPU, SGX, and the original TrustFL. The comparison is conducted using VGG16, VGG19, ResNet50, and GPT2 with a batch size of 16, across varying numbers of training rounds (R = 50, 100, 1 k, 10 k, 100 k). We set the number of samples as 43, as it is sufficient to guarantee that the soundness error is less than 1% even for the adversaries who have completed 90% of training rounds.

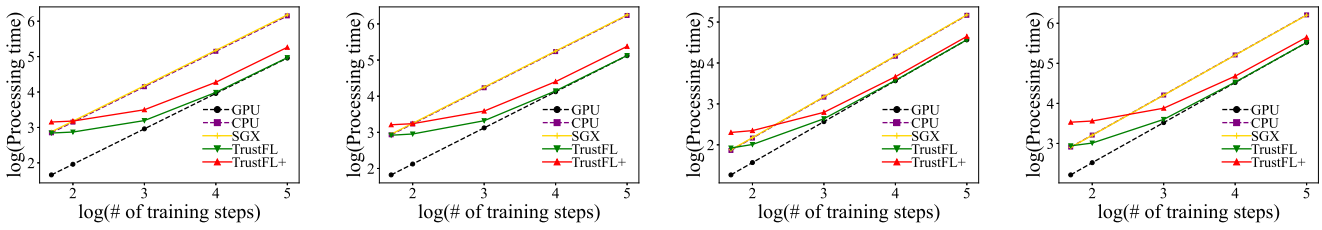
Fig. 10 show that, coincident with theoretical analysis in Section IV-B, as training steps grows, the processing time of TrustFL+ initially exceeds that of training on CPU and SGX, but gradually approaches the processing time of both TrustFL and GPU-based training. Meanwhile, the processing time of training on GPU, CPU, and SGX increases linearly with the training rounds. When set training steps to 50 or 100, the processing time of TrustFL+ is slightly higher than that of SGX with CPU. When set it to 100 k, TrustFL+ can achieve 9×, 8×, 4×, 8× speedups compared with naively using SGX, for VGG16, VGG19, Resnet50, and GPT2, respectively. Besides, for larger models, TrustFL+ can bring more speedups than that using SGX, without incurring significant overhead beyond that already present in TrustFL.

The above results show that although TrustFL+ introduces additional time overhead compared to TrustFL, the overhead is minimal and remains within an acceptable range. Moreover, it still achieves an efficiency improvement of nearly one order of magnitude compared to training directly on SGX.

### D. Scalability Evaluation

To evaluate the scalability of TrustFL+ in a federated learning setting, we simulated a server cluster consisting of 20 clients and a central server, each equipped with an RTX 4090 GPU. We compared TrustFL+ with OVT [17], one of the most recent and advanced methods that also incorporates a rounding-based strategy for verifiable training. For a fair comparison, we fix the final mantissa precision of the model parameters in OVT to 21 bits. This value is chosen because OVT's verification time increases substantially when the mantissa precision is lower than 21 bits. Additionally, we set the number of verification samples to 43 for both methods, as discussed in Section IV. For the

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.




(a) VGG16 (b) VGG19 (c) ResNet50 (d) GPT2

Fig. 10. Processing time for training the four models using GPU, TrustFL, TrustFL+, CPU, and SGX.

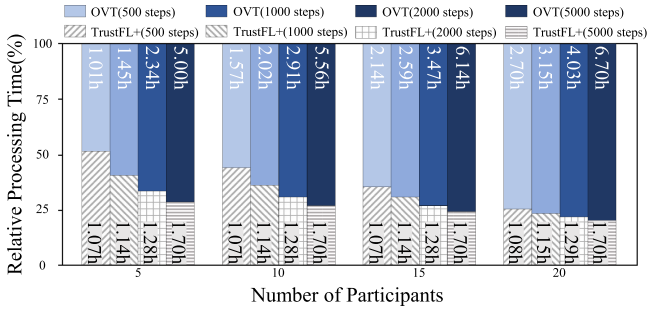



Fig. 11. Percentage-based comparison of processing times for OVT and TrustFL+ in federated learning with varying numbers of participants.

federated learning framework, we adopted GPT2 as the global model.

Fig. 11 presents a comparison of the processing time per federated learning epoch between OVT and TrustFL+ under varying numbers of participants and training steps. As shown, the advantage of TrustFL+ becomes increasingly apparent as the number of participants grows. Specifically, the time consumed by TrustFL+ remains relatively stable, while OVT incurs significantly higher overhead with more participants. When considering only the server-side proof verification overhead, OVT incurs a cost of 2005.9 s for a federated learning epoch with 5 participants, while TrustFL+ only requires 9.1 s. As the number of participants increases to 20, the overheads grow to 7998.0 s for OVT and 36.4 s for TrustFL+. Therefore, the proof verification overhead of TrustFL+ is significantly lower than that of OVT, approximately 0.5% of the cost, demonstrating the strong scalability of TrustFL+.

This is mainly due to two reasons: 1) Parallel proof generation: OVT requires the server to reproduce the training process on the submitted checkpoints to verify proofs, which is a sequential process. In contrast, TrustFL+ shifts the reproduction process to the participants during proof generation, allowing the server to simply verify the proofs, significantly reducing the overhead. 2) Optimized rounding strategy: Unlike the rounding method used in OVT, which requires bit-by-bit subtraction or addition of the target, NARS in TrustFL+ only modifies the least significant bit (the $b_{r_m}$ bit) of the valid precision and sets the invalid precision bits to zero. For example, if a 5-bit value 01011 is to be rounded to a 2-bit value 01000, OVT would need to subtract 1 from 01011 to get 01010, then subtract again to get 01001, and finally reach 01000: $01011 \rightarrow 01010 \rightarrow 01001 \rightarrow 01000$. In contrast, NARS simply checks the third bit to determine whether rounding is needed (e.g., rounding up if it's 1), then sets all invalid bits to zero: $01011 \rightarrow 01000$. As a result, NARS significantly reduces the overhead introduced by rounding compared to OVT.

Additionally, we evaluated OVT's hardware cost, as its server must be equipped with GPUs identical to those used by participants to configure rounding precision. Based on quotes from major cloud providers (AWS and Alibaba Cloud), renting five distinct GPU models costs at least 157 USD/day [67], [68]. TrustFL+ eliminates this expense.

## VI. Discussion

In this section, we discuss how TrustFL+ can be further optimized in several aspects. We focus on the participant's selection of rounding precision, analysis of potential verification failures, and methods for detecting whether the training data used by a participant is poisoned.

*Refinement on rounding precision:* In our design, $b_{r_m}$ simply represents the minimum precision that participants are allowed to use. In practice, using a precision higher than $b_{r_m}$ is often sufficient to control hardware non-determinism, and training with higher precision generally results in less degradation of model performance. Therefore, participants are free to use a precision above $b_{r_m}$ (ensure that verification can be passed at this precision) to reduce potential performance loss and potentially earn greater rewards as a result.

*Solutions for potential verification failures:* Since testing all devices and operations is impossible, we cannot guarantee that our method will handle all hardware non-determinism in every scenario. However, such failures to eliminate hardware non-determinism have never occurred in any of our experiments. Even in [17], a recent study that also uses rounding to eliminate hardware non-determinism, the authors report that a $b_{r_m} \leq 32$ was sufficient across all their test cases. Of course, in the rare event that such a situation does occur, the participant can simply negotiate with the server to determine a lower $b_{r_m}$ that eliminates the non-determinism, and then restart training. The additional overhead remains small, far less than the substantial time costs introduced by cryptographic primitives or pure TEE-based training methods.

*Support for Other Precision Formats:* Our approach primarily targets typical FP32 training scenarios but can also be applied to other formats. For FP16/BF16 models, the GPU-side training follows the procedure: (1) converting parameters to FP64, (2) performing controlled rounding at the chosen rounding precision and logging rounding directions, and (3) finally converting the resulting parameters back to FP16/BF16. Accordingly, TEE-side verification replays the same sequence of operations using the recorded rounding directions for sampled training segments. This enables bitwise-consistent reproduction under controlled rounding, thereby ensuring reliable verification.

TrustFL+ can in principle operate in FP32-only settings (i.e., no more conversion to and from FP64). However,

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

eliminating hardware non-determinism under FP32 requires aggressive rounding (i.e., merely keeping three mantissa bits), which significantly degrades model performance. In our evaluation, successful verification in the FP32-only setting results in a 10%–20% accuracy loss for VGG16, VGG19, and ResNet50, whereas GPT2 failed to converge. In contrast, using FP64 provides sufficient mantissa space to remove nondeterministic errors while preserving model performance, and the resulting model performance remains close to that of standard FP32 training. This comes with additional cost: the runtime overhead ranges from 20.4% to 99.5%, and memory usage is approximately 2× that of standard FP32 training. We consider this overhead acceptable because TrustFL+ still performs most training on GPUs and remains substantially faster than approaches that execute the entire training process inside TEEs.

*Defense against unexpected training data:* As stated in Section II-B, unexpected training data like arbitrary or poisoned data constitutes an attack surface orthogonal to our focus on training integrity. Such attacks can be mitigated by state-of-the-art server-side defenses [44], [45], [46], [47], which perform statistical analysis among participants' local model updates and reduce the influence of suspicious updates during model aggregation. For instance, to mitigate the impact of using arbitrary data, FLTrust [47] introduces a server-side root of trust by maintaining well-crafted server model updates. It assigns trust scores to local model updates based on their directional similarity to the server updates and normalizes their magnitudes accordingly, thereby reducing the influence of updates that deviate from the intended training objective during aggregation. FDCR [46] assesses parameter importance using the Fisher Information Matrix and filters out updates that display anomalous parameter-importance patterns, as poisoned data often selectively strengthens specific neurons through malicious triggers.

In practice, TrustFL+ can be combined with these defenses in a straightforward manner. During each training epoch, the participant performs all local training inside the GPU, while the co-located TEE re-executes randomly sampled segments for verification. Only if these checks pass does the TEE generate a signed attestation on the final update, which is forwarded to the server. Upon receiving the model updates along with their attestation reports, the server verifies the reports and then feeds the verified updates into the aforementioned defenses. This modular composition enables TrustFL+ to provide training integrity while allowing existing server-side defenses to mitigate the impact of unexpected training data, without requiring any modification to either mechanism. We leave a full empirical evaluation of such combined systems to future work, as our focus is on ensuring participant-side training integrity, which is orthogonal to the mitigation of unexpected training data.

*FL Privacy Defenses:* Federated learning is known to be vulnerable to data reconstruction attacks [39]. TrustFL+ can be composed with existing FL privacy defense mechanisms like differential privacy and secure aggregation. In detail, after GPU-side training is completed, the co-located TEE verifies the training integrity and attests to the resulting update. Only updates that pass TEE-based verification are then eligible for subsequent privacy-preserving processing. Differential privacy mechanisms (e.g., norm clipping followed by noise injection [40]) can be applied to these verified updates. The resulting noisy updates can enter an existing secure aggregation protocol [41]. Throughout this process, the server never gains access to any plaintext per-participant updates.

Additionally, when stronger robustness or incentive mechanisms are desired, such as determining rewards based on update quality or mitigating the impact of unexpected training data, one possible extension is to deploy a TEE on the server side to host secure aggregation and robust aggregation logic. Here, individual participant updates are decrypted and analyzed only within the server-side TEE, where quality assessment, anomaly detection [46], or robust aggregation [47] can be performed. The aggregation result is then released from the TEE, making the server and external environment never gain access to plaintext per-participant updates for privacy preserving.

## VII. Related Work

*Privacy-preserving deep learning:* Recent advances proposed various implementations of federated learning [1], [2]. Under the FL framework, some proposals envision to enhance the data privacy via subspace learning [69] or fine-grained adjustment of neural network layers [70], accelerate the overall training process with a resource-aware selection strategy [71], and improve communication efficiency between server and participants [72], [73]. Our work is complementary to theirs, as we focus on eliminating the security concerns of the server by assuring honest training on participants' side.

*Secure and robust federated learning:* Federated learning is fragile to bad participants that may launch model poisoning or backdoor attacks [74], [75]. Auror [76] defends against model poisoning in collaborative learning [1] by clustering participants' model updates and removing outliers. Multi-Krum [77] achieves model convergence by discarding outliers from mean participant gradients in the presence of Byzantine participants. FoolsGold [78] prevents sybil-based poisoning by adjusting the learning rate of strange participants. All of these efforts are a kind of alternative aggregation mechanism at the server side and cannot effectively defend against advanced stealthy attacks, like [74], and lazy participants. In contrast, we enforce local training at the (possibly lazy) participant side. Moreover, our work can be easily combined with the above solutions to defend against malicious participants.

*TEE-protected machine learning:* Recently, TEE has become increasingly popular in the area of developing privacy-preserving machine learning. Wang et al. [79] propose a secure and efficient inference framework for deep learning in untrustworthy cloud platforms, which aims to protect both data and model confidentiality and mitigate side-channel attacks in untrusted clouds by leveraging a multi-exit early-exit network with on-demand enclave loading and joint training via knowledge distillation. Peterson et al. [80] study centralized ML tasks on an SGX-enabled data center for data privacy, and propose PLINIUS, which securely mirrors encrypted models and data between SGX enclaves and persistent memory to enable crash recovery while minimizing side-channel leakage. Our work is different, as we do not need to take side channel attacks into consideration in the data owner platform.

Another line of work also proposes GPU-TEE co-designed methods, such as TEESlice [81], [82], to protect data and model confidentiality during model inference. They usually put the sensitive part of the model inside the TEE while putting the rest on the GPU. In contrast, TrustFL+ focuses on selectively validating the training process inside the TEE, focusing on integrity rather than confidentiality.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

*Controlling hardware non-determinism:* Many existing works have tackled hardware non-determinism using varied strategies. Some approaches [19], [83] enforce determinism by fixing the order of floating-point accumulation, which limits their applicability in heterogeneous environments. Other methods tolerate non-determinism through approximate verification by introducing error tolerance thresholds. For instance, Proof-of-Learning (PoL) [16] validates model correctness by checking whether the submitted model falls within a plausible training trajectory, allowing deviations within a predefined threshold. A more recent approach, OVT [17], eliminates non-determinism by rounding the results of all operations to a fixed precision, ensuring bitwise-identical outputs across devices without relying on error thresholds. However, OVT suffers from poor scalability because its verification process requires the server to access GPUs of the same type as participants' GPUs and introduces extra hardware costs. The problem is addressed by TrustFL+.

## VIII. Conclusion

In this paper, we propose TrustFL+, an efficient, scalable, and reliable verification scheme to ensure training integrity of untrusted participants in federated learning systems. We utilize GPUs to perform all training tasks, leverage TEE to verify the integrity of the training process, and introduce the novel adaptive rounding strategy to ensure the reliability of TEE verification results. We have conducted theoretical analysis and extensive experiments. The experimental results demonstrate that TrustFL+ effectively addresses the potential false positive issue in TrustFL, achieves nearly a $10\times$ speedup over pure TEE training, and exhibits strong scalability with a large number of participants.

## References


[1] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1310–1321.

[2] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[3] A. Rajkomar et al., "Scalable and accurate deep learning with electronic health records," *NPJ Digit. Med.*, vol. 1, no. 1, 2018, Art. no. 18.

[4] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, and A. Narayanaswamy, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *Jama*, vol. 316, no. 22, pp. 2402–2410, 2016.

[5] European Parliament and Council of the European Union, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)," 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679

[6] Q. Yang et al., "Stealthy backdoor attack in federated learning via adaptive layer-wisegradient alignment," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2025, pp. 29163–29172.

[7] H. Zhuang, M. Yu, H. Wang, and Y. Hua, "Backdoor federated learning by poisoning backdoor-critical layers," in *Proc. Int. Conf. Learn. Representations*, 2024, pp. 16427–16452.

[8] M. Li, W. Wan, Y. Ning, and S. Hu, "DarkFed: A data-free backdoor attack in federated learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2024, pp. 4443–4451.

[9] N. Hynes, D. Dao, D. Yan, R. Cheng, and D. Song, "A demonstration of sterling: A privacy-preserving data marketplace," in*Proc. VLDB Endowment*, vol. 11, pp. 2086–2089, 2018.

[10] T. Liu, X. Xie, and Y. Zhang, "zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 2968–2985.

[11] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "Scaling up trustless DNN inference with zero-knowledge proofs," 2022, *arXiv:2210.08674*.

[12] S. Garg et al., "Experimenting with zero-knowledge proofs of training," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 1880–1894.

[13] Y. Chen, F. Luo, T. Li, T. Xiang, Z. Liu, and J. Li, "A training-integrity privacy-preserving federated learning scheme with trusted execution environment," *Inf. Sci.*, vol. 522, pp. 69–79, 2020.

[14] A. Mondal, Y. More, R. H. Rooparaghunath, and D. Gupta, "Flatee: Federated learning across trusted execution environments," 2021, *arXiv:2111.06867*.

[15] A. Coates, B. Huval, and T. Wang, "Deep learning with COTS HPC systems," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1337–1345.

[16] H. Jia, M. Yaghini, C. A. Choquette-Choo, and N. Dullerud, "Proof-of-learning: Definitions and practice," in *Proc. IEEE Symp. Secur. Privacy*, 2021, pp. 1039–1056.

[17] M. Srivastava, S. Arora, and D. Boneh, "Optimistic verifiable training by controlling hardware nondeterminism," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 95639–95661.

[18] A. Schlögl, N. Hofer, and R. Böhme, "Causes and effects of unanticipated numerical deviations in neural network inference frameworks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 56095–56107.

[19] H. Jooybar, W. W. Fung, M. O'Connor, J. Devietti, and T. M. Aamodt, "GPUdet: A deterministic GPU architecture," in *Proc. Int. Conf. Architectural Support Program. Languages Operating Syst.*, 2013, pp. 1–12.

[20] B. Chen, M. Wen, Y. Shi, and D. Lin, "Towards training reproducible deep learning models," in *Proc. Int. Conf. Softw. Eng.*, 2022, pp. 2202–2214.

[21] X. Zhang, F. Li, Z. Zhang, Q. Li, and J. Wu, "Enabling execution assurance of federated learning at untrusted participants," in Proc. IEEE Conf. Comput. Commun.,2020, pp. 1877–1886.

[22] C. Fang et al., "Proof-of-learning is currently more broken than you think," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2023, pp. 797–816.

[23] R. Zhang, J. Liu, Y. Ding, Z. Wang, Q. Wu, and K. Ren, ""Adversarial examples" for proof-of-learning," in *Proc. IEEE Symp. Secur. Privacy*, 2022, pp. 1408–1422.

[24] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, and Johnson, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2019, pp. 185–200.

[25] Y. Shen et al., "Occlum: Secure and efficient multitasking inside a single enclave of Intel SGX," in *Proc. Int. Conf. Architectural Support Program. Languages Operating Syst.*, 2020, pp. 955–970.

[26] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[27] Intel Corporation, "Intel software guard extensions programming reference," 2014. [Online]. Available: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf

[28] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions," *HASP@ISCA*, vol. 11, no. 10.1145, pp. 2487726–2488370, 2013.

[29] Intel Corporation, "Intel software guard extensions: Intel attestation service API," 2016. [Online]. Available: https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf

[30] F. Zhang, E. Cecchetti, and K. Croman, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 270–282.

[31] X. Zhang et al., "Secure collaborative learning in mining pool via robust and efficient verification," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2023, pp. 794–805.

[32] W. Kahan, "Pracniques: Further remarks on reducing truncation errors," *Commun. ACM*, vol. 8, 1965, Art. no. 40.

[33] N. Whitehead and A. Fit-Florea, "Precision & performance: Floating point and IEEE 754 compliance for NVIDIA GPUs," *NVIDIA*, 2011. [Online]. Available: https://developer.download.nvidia.com/assets/cuda/files/NVIDIA-CUDA-Floating-Point.pdf

[34] NVIDIA, "Determinism across GPU architectures?" 2022. [Online]. Available: https://github.com/NVIDIA/framework-reproducibility/issues/28

[35] Y. H. Chou, C. Ng, S. Cattell, J. Intan, and M. D. Sinclair, "Deterministic atomic buffering," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2020, pp. 981–995.

[36] A. K. Kamath and A. Basu, "iGUARD: In-GPU advanced race detection," in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.*, 2021, pp. 49–65.

[37] M. D. Sinclair, J. Alsop, and S. V. Adve, "Efficient GPU synchronization without scopes: Saying no to complex consistency models," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2015, pp. 647–659.


Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

[38] F. Tramer, F. Zhang, H. Lin, J.-P. Hubaux, A. Juels, and E. Shi, "Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2017, pp. 19–34.
[39] D. Pham, S. Kulkarni, and A. Houmansadr, "RAIFLE: Reconstruction attacks on interaction-based federated learning with adversarial data manipulation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2025.
[40] Z. Xu et al., "Federated learning of Gboard language models with differential privacy," in *Proc. 61st Annu. Meeting Assoc. Comput. Linguistics*, 2023, pp. 629–639.
[41] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1175–1191.
[42] H. Wang et al., "Attack of the tails: Yes, you really can backdoor federated learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 16070–16084.
[43] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "DBA: Distributed backdoor attacks against federated learning," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 5830–5848.
[44] T. D. Nguyen, P. Rieger, H. Chen, and H. Yalame, "{FLAME}: Taming backdoors in federated learning," in *Proc. USENIX Secur.*, 2022, pp. 1415–1432.
[45] P. Rieger, T. D. Nguyen, M. Miettinen, and A.-R. Sadeghi, "DeepSight: Mitigating backdoor attacks in federated learning through deep model inspection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–18.
[46] W. Huang, M. Ye, Z. Shi, G. Wan, H. Li, and B. Du, "Parameter disparities dissection for backdoor defense in heterogeneous federated learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2024, pp. 120951–120973.
[47] X. Cao, M. Fang, J. Liu, and N. Gong, "FLTrust: Byzantine-robust federated learning via trust bootstrapping," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.
[48] J. Van Bulck et al., "Foreshadow: Extracting the keys to the Intel kingdom with transient execution," in *Proc. USENIX Secur.*, 2018, pp. 991–1008.
[49] O. Ohrimenko et al., "Oblivious multi-party machine learning on trusted processors," in *Proc USENIX Secur.*, 2016, pp. 619–636.
[50] K. Bonawitz, H. Eichner, and W. Grieskamp, "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst.*, 2019, vol. 1, pp. 374–388.
[51] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 9360–9378.
[52] N. Hynes, R. Cheng, and D. Song, "Efficient deep learning on multi-source private data," 2018, *arXiv:1807.06689*.
[53] *American National Standard IEEE Standard for Binary Floating-Point Arithmetic*, Std. ANSI/IEEE 754-1985, American National Standards Institute, Inc., Washington, D.C., USA, 1985.
[54] *Standard for Floating-Point Arithmetic*, IEEE Standard 754-2008, Aug. 2008.
[55] NVIDIA, "NVIDIA A100 tensor core GPU architecture," NVIDIA Corporation, Santa Clara, CA, USA, White Paper V1.0, May 2020. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf
[56] C. Jiang, A. Jayarajan, H. Lu, and G. Pekhimenko, "Arbitor: A numerically accurate hardware emulation tool for {DNN}accelerators," in *Proc. USENIX Annu. Tech. Conf.*, 2023, pp. 519–536.
[57] Z. Li, B. Ding, L. Yao, Y. Li, X. Xiao, and J. Zhou, "Performance-based pricing for federated learning via auction," in*Proc. VLDB Endowment*, vol. 17, pp. 1269–1282, 2024.
[58] W. Zhang, P. Janson, and R. Aljundi, "Overcoming generic knowledge loss with selective parameter update," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2024, pp. 24046–24056.
[59] W. Du, J. Jia, M. Mangal, and M. Murugesan, "Uncheatable grid computing," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 4–11.
[60] Y. Chang, H. Jiang, C. Lin, X. Huang, and J. Weng, "Towards understanding and enhancing security of {proof-of-training}for {DNN}model ownership verification," in *Proc. USENIX Secur.*, 2025, pp. 7233–7250.
[61] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Tech. Rep., Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
[62] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1851–1865.
[63] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[64] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–14.
[65] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[66] A. Radford et al., "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, 2019, Art. no. 9.
[67] Alibaba Cloud, "Alibaba cloud (Aliyun)," 2026. [Online]. Available: https://www.aliyun.com/
[68] Amazon Web Services, "Amazon elastic compute cloud (EC2)," 2026. [Online]. Available: https://aws.amazon.com/ec2/
[69] J. Li, N. Li, and B. Ribeiro, "MIST: Defending against membership inference attacks through membership-invariant subspace training," in *Proc. USENIX Secur.*, 2024, pp. 2387–2404.
[70] N. Xu, B. Wang, R. Ran, W. Wen, and P. Venkitasubramaniam, "NeuGuard: Lightweight neuron-guided defense against membership inference attacks," in *Proc. Annu. Comput. Secur. Appl. Conf.*, 2022, pp. 669–683.
[71] A. L. Nunes, C. Boeres, L. M. A. Drummond, and L. L. Pilla, "Optimal time and energy-aware client selection algorithms for federated learning on heterogeneous resources," in*Proc. IEEE 36th Int. Symp. Comput. Architecture High Perform. Comput.*, 2024, pp. 148–158.
[72] Z. Tang, S. Shi, B. Li, and X. Chu, "GossipFL: A decentralized federated learning framework with sparsified and adaptive communication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 3, pp. 909–922, Mar. 2023.
[73] D. Wu, X. Zou, S. Zhang, H. Jin, W. Xia, and B. Fang, "SmartIdx: Reducing communication cost in federated learning by exploiting the CNNs structures," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 4254–4262.
[74] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 2938–2948.
[75] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 634–643.
[76] S. Shen, S. Tople, and P. Saxena, "Auror: Defending against poisoning attacks in collaborative deep learning systems," in *Proc. Annu. Conf. Comput. Secur. Appl.*, 2016, pp. 508–519.
[77] P. Blanchard et al., "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 118–128.
[78] C. Fung, C. J. Yoon, and I. Beschastnikh, "The limitations of federated learning in sybil settings," in *Proc. Int. Symp. Res. Attacks, Intrusions Defenses*, 2020, pp. 301–316.
[79] Y. Wang, K. Deng, F. Meng, Z. Chen, M. Yin, and R. Yang, "Branchy-TEE: Deep learning security inference acceleration using trusted execution environment," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, 2023, pp. 499–504.
[80] P. Yuhala, P. Felber, and V. Schiavoni, "Plinius: Secure and persistent machine learning model training," in *Proc. Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2021, pp. 52–62.
[81] D. Li, Z. Zhang, M. Yao, Y. Cai, Y. Guo, and X. Chen, "TEESlice: Protecting sensitive neural network models in trusted execution environments when attackers have pre-trained models," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, pp. 1–49, 2024.
[82] Z. Zhang, L. K. Ng, B. Liu, and Y. Cai, "TEESlice: Slicing DNN models for secure and efficient deployment," in *Proc. ACM Int. Workshop AI Softw. Testing/Anal.*, 2022, pp. 1–8.
[83] D. Defour and C. Collange, "Reproducible floating-point atomic addition in data-parallel environment," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, 2015, pp. 721–728.

**Cheng Lyu** is currently working toward the ME degree with the University of Science and Technology Beijing. His research interests include trusted computing, verifiable computation, and privacy in large language models.

**Xiaoli Zhang** received the PhD degree from Tsinghua University. She is currently an associate professor with the Department of Computer Science and Technology, University of Science and Technology Beijing. Her research interests include trusted computing, cloud security, and network security.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.

**Jiaqing Cheng** is currently working toward the ME degree with the University of Science and Technology Beijing. His research interests include TEEs and system security.

**Wenmao Liu** received the PhD degree from the Harbin Institute of Technology. He is currently the chief innovation officer and the leader with the XingYun Lab (Cloud security) of NSFOCUS Information Technology Company, Ltd. His research interests include cloud security, IoT security, and data-driven analytics.

**Xiaohu Ye** received the PhD degree from Tsinghua University. He is currently the chief technology officer with NSFOCUS Information Technology Company, Ltd. His research interests include network security, data security, and industrial internet security.

**Ke Xu** (Fellow, IEEE) received the PhD degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, where he is currently a full professor. He has authored or coauthored more than 200 technical articles and holds more than 10 U.S. patents in the research areas of next-generation Internet architecture, network security, and AI-driven networking. He was the recipient of numerous accolades for his work including multiple Distinguished Paper Awards at top-tier venues such as USENIX Security (2023, 2024), ACM CCS 2025, and NDSS 2025.

**Qi Li** (Senior Member, IEEE) received the PhD degree from Tsinghua University. He is currently an associate professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network and system security, with particular emphasis on Internet security and the security of machine learning systems. He is an editorial board member for *IEEE Transactions on Dependable and Secure Computing*, and *ACM Transactions on Privacy and Security*.

**Xu-Cheng Yin** (Senior Member, IEEE) received the PhD degree from the Institute of Automation, Chinese Academy of Sciences. He is currently a full professor with the Department of Computer Science and Technology, University of Science and Technology Beijing. His research interests include pattern recognition, machine learning, and computer vision.

Authorized licensed use limited to: BEIJING INSTITUTE OF TECHNOLOGY. Downloaded on November 15,2026 at 03:00:59 UTC from IEEE Xplore. Restrictions apply.