OPFL: OPTIMISTIC VERIFICATION OF FEDERATED LEARNING VIA EMPIRICAL BOUNDARY

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated learning enables multiple clients to collaboratively train models without sharing their private data. However, the lack of visibility into local training makes it difficult to verify whether clients follow the prescribed training procedure or submit malicious updates, such as model poisoning. A natural approach is to replay client training for verification. However, privacy-preserving replay produces numerical results that cannot be directly matched with local client execution because the two run in different environments. We present OPFL, an optimistic verification framework for privacy-preserving federated learning. To protect data privacy, OPFL performs replay inside secure multi-party computation (MPC). Although gradients computed on MPC and local GPUs are not bitwise identical, we observe that their absolute differences are stable and bounded. OPFL therefore calibrates an empirical boundary offline and uses it to distinguish benign numerical deviations from malicious manipulation. To reduce the cost of expensive MPC replay, OPFL adopts optimistic verification by post auditing only sampled training steps. Experiments on LeNet, BERT, and Qwen show that the boundary generalizes across datasets, input lengths, and GPUs, while achieving 0% ASR against model poisoning and PGD-based attacks. On a LeNet workload, at $p = 0.01$, OPFL is approximately $98.6\times$ faster than full MPC-based FL and $625.5\times$ faster than ZK-based approach.

1 INTRODUCTION

Federated learning (FL) has become an important paradigm for collaborative machine learning in privacy-sensitive settings, including on-device learning and multi-institutional healthcare (McMahan et al., 2017; Molaei et al., 2024). Instead of collecting raw data at a central server, FL allows multiple clients to train a global model using their local datasets. In each round, clients download the current global model, perform local training, and return only model updates to the server for aggregation (McMahan et al., 2017). The raw training data therefore remains on client devices. However, this privacy benefit also makes local training difficult to observe. A malicious client may deviate from the prescribed training procedure or directly construct a poisoned update while still submitting it as the result of legitimate local training (Bagdasaryan et al., 2020; Zhuang et al., 2024; Pang et al., 2025). This creates an execution integrity problem: how can a server verify that a private client update was actually produced by the prescribed local training procedure?

Existing approaches address this problem from several directions. Secure aggregation with update validation can constrain which client updates are accepted without revealing them (Lycklama et al., 2023; Roy Chowdhury et al., 2022; Bell et al., 2023; Zhu et al., 2024). However, these methods primarily verify properties or constraints of the submitted update rather than directly establishing how the update was produced. Cryptographic approaches can verify training computations more directly. For example, recent systems use zero-knowledge proofs to enforce private computation integrity in federated or split learning settings (Zhu et al., 2024; Zheng et al., 2026). Nevertheless, encoding deep neural-network training into cryptographic proofs remains expensive and difficult to scale (Zhu et al., 2024; Zheng et al., 2026). Replay provides another natural alternative. Prior approaches to verifiable training record intermediate training states and re-execute selected computations to check consistency with the claimed training trajectory (Jia et al., 2021; Srivastava et al., 2024). Yet replay is difficult to apply directly in federated learning. A verifier would need to reproduce the client's execution environment as closely as possible and access the raw training data used in local computa-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

tion, which conflicts with the privacy goal of federated learning. A natural alternative is to perform replay inside a cryptographically protected environment such as MPC. However, such environments differ from native GPU execution and can introduce unavoidable numerical discrepancies (Goldberg, 1991; Demmel & Nguyen, 2015; Collange et al., 2015; Srivastava et al., 2024). Their outputs cannot generally be compared through exact equality, making direct verification ineffective.

To address these challenges, we introduce OPFL, a framework for privately verifying FL through an empirical gradient-discrepancy boundary. OPFL uses MPC to enable **privacy-preserving replay**, but does not require the replayed gradient to be bitwise identical to the gradient produced by native GPU training. Our key observation is that the gradient differences between GPU execution and MPC replay exhibit highly structured behavior and remain empirically bounded. This structured behavior gives rise to an **empirical boundary** that separates benign numerical discrepancies from malicious deviations. OPFL calibrates this boundary by executing the same training configuration in both native GPU and MPC environments and measuring the absolute gradient discrepancy $|G_{\mathrm{GPU}} - G_{\mathrm{MPC}}|$ in limited training steps. The resulting discrepancy distribution is summarized using a subset of percentile statistics, which preserves distributional information while avoiding the cost of checking every gradient coordinate. This calibration is performed offline once before FL starts and is reused during subsequent verification. Replaying every training step inside MPC is computationally expensive (Mohassel & Rindal, 2018; Ma et al., 2023). To reduce this cost, OPFL adopts an **optimistic auditing mechanism** and invokes MPC replay only on sampled training steps. This design follows a well-established paradigm in optimistic verification systems (Kalodner et al., 2018). Claims are accepted optimistically, while detected misbehavior can trigger challenges and economic penalties. Clients commit their training evidence before audit sampling. A verifier committee then replays only the selected steps inside MPC and checks the resulting gradient against the calibrated boundary. Failed audits trigger deposit slashing, under the assumption that a majority of committee members are honest.

We evaluate OPFL across diverse training settings, including CNN and Transformer architectures, multiple optimizers, and both full model training and LoRA finetuning (Hu et al., 2022). First, we evaluate whether OPFL can distinguish honest numerical discrepancies from malicious deviations. We consider four representative attacks adapted from prior work, including gradient reuse, gradient sign reversal, label flipping, and gradient scaling (Fraboni et al., 2021; Damaskinos et al., 2018; Fang et al., 2020; Bagdasaryan et al., 2020; Lycklama et al., 2023). OPFL achieves 0% ASR while maintaining 0% false rejection rate across these attacks. We further construct a white-box adaptive PGD attack. PGD is widely regarded as a strong first-order adversary (Madry et al., 2018), making it a suitable stress test for our verification framework. The ASR remains 0% for OPFL. In contrast, the evaluated baselines can detect large deviations but are more easily bypassed under smaller adaptive perturbations. Detailed results are provided in section 4. Second, we study the stability and generalization of the calibrated boundary. As shown in section 4.2, a stable boundary can be obtained from only a small number of calibration steps. Across LeNet, BERT, and Qwen3, boundaries calibrated using the first 100 training steps generalize across different training stages, datasets, sequence lengths, and GPU hardware. Finally, we evaluate the system overhead of OPFL. In our five-client LeNet workload, a 1% audit rate reduces the measured verification and aggregation time by approximately $98.6\times$ relative to full MPC-based verification and $625.5\times$ relative to ZKSL-LP-G (Zheng et al., 2026).

In summary, our key finding is that the gradient discrepancy between MPC execution and native GPU execution is bounded, and can therefore serve as an effective signal for identifying malicious training deviations, as demonstrated by our experiments. We further build a verification framework that audits FL clients without exposing their private training data, and introduce an optimistic verification mechanism to reduce system overhead. OPFL focuses on execution integrity and does not address data poisoning, which can be handled by complementary robust aggregation or trust-based defenses (Blanchard et al., 2017; Cao et al., 2021). The construction and incentivization of the verifier committee are also orthogonal to our design and can rely on established decentralized committee-selection and incentive mechanisms (Gilad et al., 2017; Kiayias et al., 2017). Finally, OPFL assumes an existing privacy-preserving aggregation layer, such as secure aggregation (Bonawitz et al., 2017), rather than introducing a new aggregation protocol.

2 Problem Formulation and Threat Model

Motivation. FL assumes that participating clients faithfully execute the prescribed local training procedure before submitting their updates. In practice, a malicious or economically motivated client may deviate from this procedure. For example, it may skip computation, reuse a stale gradient, or directly fabricate an update to reduce local training cost. A client may also manipulate its computation to steer the global model toward a malicious objective (Fraboni et al., 2021; Bagdasaryan et al., 2020). Since the server observes only the submitted update, such deviations are difficult to distinguish from honestly generated updates without additional verification.

FL Setting. We consider a FedSGD-style FL setting (McMahan et al., 2017) with a central server $\mathcal{S}$ and a set of clients $\{\mathcal{C}_i\}_{i=1}^{n}$. Each client $\mathcal{C}_i$ holds a private local dataset D_i. At training round t, the server distributes the current global model W_t to the selected clients. Let $z_{i,t}$ denote the private training input used by client $\mathcal{C}_i$ at round t, derived from D_i. In this setting, each selected client performs a single local training step per round and submits the resulting gradient for aggregation. We use this formulation for clarity; the same verification principle can be extended to multi-step local training by treating each local step as an auditable computation and binding the corresponding intermediate states. Each selected client performs the prescribed local computation according to a public training configuration Π, which specifies the training algorithm, hyperparameters, model and optimizer settings, data-processing rules, randomness configuration, and other metadata required for replay. Client $\mathcal{C}_i$ computes

$$G_{i,t}^{\mathrm{GPU}} = \mathrm{Train}_{\mathrm{GPU}}\left(W_t, z_{i,t}; \Pi\right), \tag{1}$$

and submits the resulting gradient for aggregation.

We augment this setting with a verifier committee $\mathcal{K}$ that audits randomly selected client-round pairs (i,t). Before audit selection, each client commits to its private training inputs and claimed gradients. The global model W_t and training configuration Π are fixed independently of the client. We assume a public and binding commitment layer that prevents committed evidence from being modified. Our goal is not to determine whether a client's private dataset is itself benign. Instead, we verify whether the committed gradient is consistent with executing the prescribed local computation on the committed private input.

Verification Goal. A direct way to verify a client gradient is to replay the same computation and compare the result with the client's claim. However, revealing $z_{i,t}$ to the verifier would violate the privacy objective of FL. We therefore perform replay inside secure multi-party computation (MPC) (Mohassel & Rindal, 2018; Ma et al., 2023). For an audited client-round pair (i,t), the committee obtains

$$G_{i,t}^{\mathrm{MPC}} = \mathrm{Train}_{\mathrm{MPC}}\left(W_t, z_{i,t}; \Pi\right) \tag{2}$$

without revealing $z_{i,t}$ to the committee members.

Exact replay would require $G_{i,t}^{\mathrm{GPU}} = G_{i,t}^{\mathrm{MPC}}$. In practice, this requirement is too strong. Native GPU execution and MPC replay use different numerical representations and computation environments, and can therefore produce different gradients even when both executions are honest (Srivastava et al., 2024).

We therefore verify the discrepancy between the two gradients rather than requiring exact equality. For gradient coordinate j, the absolute discrepancy is

$$\Delta_{i,t,j}^{\mathrm{abs}} = \left| \left[G_{i,t}^{\mathrm{GPU}}\right]_j - \left[G_{i,t}^{\mathrm{MPC}}\right]_j \right|. \tag{3}$$

Let $\mathcal{B}$ denote the empirical gradient-discrepancy boundary calibrated from honest GPU–MPC executions. An audited client-round pair is accepted if its discrepancy profile satisfies the thresholds specified by $\mathcal{B}$, and rejected otherwise. Section 3 defines the complete absolute and relative discrepancy profiles and describes how $\mathcal{B}$ is calibrated.

Accordingly, OPFL targets two properties. First, *privacy*: verification should not reveal the client's raw training input to the verifier committee. Second, *execution integrity*: an audited deviation that produces discrepancies beyond the calibrated numerical tolerance should be rejected.

Threat Model and Assumptions. We consider malicious clients that may arbitrarily deviate from the prescribed local computation by skipping computation, reusing previous gradients, modifying intermediate computation, or submitting arbitrary gradients. The adversary knows the verification procedure and boundary $\mathcal{B}$ and may adapt its strategy accordingly.

We assume:

- **Post-commitment auditing.** Client evidence is committed before audit randomness is revealed, while W_t, Π, and $\mathcal{B}$ are fixed independently of the client.
- **Binding commitments.** Committed inputs and gradients cannot be replaced without detection.
- **MPC security.** Committee members follow the MPC protocol, and corrupted members remain within its privacy threshold, preserving input confidentiality and correct verification.

Scope. OPFL focuses on verifying the integrity of local training execution. It does not prevent data poisoning when a client faithfully executes the prescribed computation on malicious or poisoned local data; such threats can be addressed by complementary robust aggregation or trust-based defenses (Blanchard et al., 2017; Cao et al., 2021). We also do not address the construction of the verifier committee, which can rely on established decentralized committee-selection mechanisms (Gilad et al., 2017; Kiayias et al., 2017). Finally, OPFL assumes an existing privacy-preserving aggregation mechanism, such as secure aggregation (Bonawitz et al., 2017), rather than introducing a new aggregation protocol. These components are orthogonal to our work.

3 OPFL Design

OPFL operates in three stages: preparation and commitment, local training, and pre-aggregation verification. The first stage fixes the verification rule and training configuration before execution. The second stage commits the client gradients produced during FL and sends them privately to the verifier committee. The third stage verifies every gradient against its commitment and performs MPC replay on randomly sampled client-round pairs before eligible contributions are released for aggregation. Figure 1 provides an overview of the workflow.

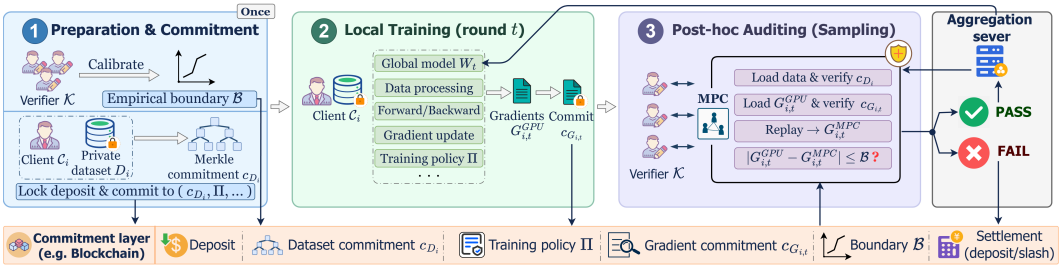


Figure 1: Overview of the OPFL protocol.

Preparation and commitment. OPFL relies on a public and binding commitment layer to record deposits and protocol commitments. A natural instantiation is a blockchain, which provides an immutable record and supports programmable deposit-and-slashing rules. Each client locks a predefined deposit on the blockchain.

Before FL begins, the verifier performs offline calibration using a set of honest calibration steps. Let s index a calibration step and j a gradient coordinate. We define the absolute and relative discrepancies as

$$\Delta^{\text{abs}}_{s,j} = \left| [G^{\text{GPU}}_s]_j - [G^{\text{MPC}}_s]_j \right|, \qquad \Delta^{\text{rel}}_{s,j} = \frac{\Delta^{\text{abs}}_{s,j}}{\max\{|[G^{\text{GPU}}_s]_j|, |[G^{\text{MPC}}_s]_j|\} + \epsilon}. \tag{4}$$

Here, $\epsilon > 0$ prevents division by zero and stabilizes the relative discrepancy near zero.

For each calibration step s, the verifier treats $\{\Delta^{\mathrm{abs}}_{s,j}\}_{j=1}^{d}$ and $\{\Delta^{\mathrm{rel}}_{s,j}\}_{j=1}^{d}$ as empirical discrepancy distributions over the d gradient coordinates. Let $Q^{\mathrm{abs}}_s(p)$ and $Q^{\mathrm{rel}}_s(p)$ denote their empirical p-quantiles. We evaluate these quantiles on $\mathcal{P} = \{0.02, 0.05, 0.10, 0.15, \ldots, 0.90, 0.95, 0.98\}$. The grid $\mathcal{P}$ can be adjusted to trade verification granularity for cost. A denser grid captures more distributional information and stricter but requires more quantile computation and boundary checks.

Percentile profiles characterize the overall discrepancy distribution but leave extreme tail coordinates unconstrained. We therefore additionally define $\Delta^{\infty}_s = \|G^{\mathrm{GPU}}_s - G^{\mathrm{MPC}}_s\|_{\infty}$. Let $\mathcal{S}_{\mathrm{cal}}$ denote the set of calibration steps. The raw bounds are

$$\widetilde{B}^{\mathrm{abs}}_p = \max_{s \in \mathcal{S}_{\mathrm{cal}}} Q^{\mathrm{abs}}_s(p), \quad \widetilde{B}^{\mathrm{rel}}_p = \max_{s \in \mathcal{S}_{\mathrm{cal}}} Q^{\mathrm{rel}}_s(p), \quad \widetilde{B}_{\infty} = \max_{s \in \mathcal{S}_{\mathrm{cal}}} \Delta^{\infty}_s. \tag{5}$$

To provide additional tolerance for deployment-time numerical variation, the raw bounds are enlarged by a predefined safety factors $\alpha_{\mathrm{abs}}, \alpha_{\mathrm{rel}}, \alpha_{\infty} \in \mathbb{R}_{>0}$:

$$B^{\mathrm{abs}}_p = \alpha_{\mathrm{abs}} \widetilde{B}^{\mathrm{abs}}_p, \quad B^{\mathrm{rel}}_p = \alpha_{\mathrm{rel}} \widetilde{B}^{\mathrm{rel}}_p, \quad B_{\infty} = \alpha_{\infty} \widetilde{B}_{\infty}. \tag{6}$$

The deployed verification boundary is $\mathcal{B} = \big(\{B^{\mathrm{abs}}_p, B^{\mathrm{rel}}_p\}_{p \in \mathcal{P}}, B_{\infty}\big)$.

Boundary calibration is performed only once for a given model and the resulting boundary is fixed before FL begins. The client then commits to the private training inputs used during training. Let $\widetilde{D}_i = (z_{i,0}, \ldots, z_{i,N_i-1})$ denote the canonicalized sequence of training inputs of client $\mathcal{C}_i$. For each input, the client samples private randomness $r_{i,t}$. The input commitment and dataset root are

$$c_{i,t} = H(\mathsf{tag}_D \| r_{i,t} \| \operatorname{Encode}(z_{i,t})), \qquad c_{D_i} = \mathsf{MerkleRoot}(c_{i,0}, \ldots, c_{i,N_i-1}), \tag{7}$$

where $\operatorname{Encode}(\cdot)$ denotes a deterministic canonical encoding.

The dataset commitment c_{D_i}, calibrated boundary $\mathcal{B}$, and training policy Π are fixed and publicly recorded on the commitment layer before training, while the underlying dataset remains private.

Local training and gradient commitment. At round t, client $\mathcal{C}_i$ uses the global model W_t, the committed training batch $z_{i,t}$, and public policy Π to compute $G^{\mathrm{GPU}}_{i,t} = \mathrm{Train}_{\mathrm{GPU}}(W_t, z_{i,t}; \Pi)$.

The client samples private randomness $r_{G_{i,t}}$ and publishes the gradient commitment $c_{G_{i,t}} = H(\mathsf{tag}_G \| r_{G_{i,t}} \| \operatorname{Encode}(G^{\mathrm{GPU}}_{i,t}))$ before audit selection. Instead of sending the gradient directly to the aggregation server, the client secret-shares $G^{\mathrm{GPU}}_{i,t}$ and $r_{G_{i,t}}$ to the verifier committee for pre-aggregation verification.

Pre-aggregation verification and auditing. Verification proceeds in two gates before aggregation. First, every submitted gradient is checked inside MPC against its previously published commitment. For client-round pair (i,t), the committee verifies

$$b_{G,i,t} = \left[H\big(\mathsf{tag}_G \| r_{G_{i,t}} \| \operatorname{Encode}(G^{\mathrm{GPU}}_{i,t})\big) = c_{G_{i,t}}\right]. \tag{8}$$

A contribution that fails this commitment check is rejected and never enters aggregation.

Second, after gradient commitments are fixed, the protocol randomly samples client-round pairs for replay auditing. An unaudited contribution that passes the commitment check is accepted optimistically. For an audited pair (i,t), the client additionally secret-shares the committed training input $z_{i,t}$ and its opening $r_{i,t}$. The committee verifies that the input belongs to the committed dataset and then replays the prescribed computation inside MPC: $G^{\mathrm{MPC}}_{i,t} = \mathrm{Train}_{\mathrm{MPC}}(W_t, z_{i,t}; \Pi)$.

The committee computes the absolute and relative discrepancy profiles between $G^{\mathrm{GPU}}_{i,t}$ and $G^{\mathrm{MPC}}_{i,t}$. Let $P^{\mathrm{abs}}_{i,t}(p)$ and $P^{\mathrm{rel}}_{i,t}(p)$ denote the corresponding empirical p-quantiles. It also computes the tail discrepancy $\Delta^{\infty}_{i,t} = \|G^{\mathrm{GPU}}_{i,t} - G^{\mathrm{MPC}}_{i,t}\|_{\infty}$. The replay audit passes only if

$$P^{\mathrm{abs}}_{i,t}(p) \le B^{\mathrm{abs}}_p, \quad P^{\mathrm{rel}}_{i,t}(p) \le B^{\mathrm{rel}}_p, \quad \forall p \in \mathcal{P}; \qquad \Delta^{\infty}_{i,t} \le B_{\infty}. \tag{9}$$

Only contributions that pass the commitment gate and, when sampled, the replay audit are authorized for aggregation. The committee forwards the same verified gradient shares to the aggregation layer,

so the client does not provide a second aggregation input. Contributions that fail either gate are excluded from aggregation and trigger the predefined penalty. Only the final eligibility decision is revealed.

Let $a_{i,t} \in \{0, 1\}$ denote the post-commitment audit indicator, where $a_{i,t} = 1$ indicates that (i, t) is selected for replay. Algorithm 1 summarizes the procedure.

Algorithm 1 Pre-aggregation verification in OPFL

Require: $W_t, \Pi, \mathcal{B}, c_{D_i}, c_{G_{i,t}}, c_{i,t}, a_{i,t}$
Require: Private $[G_{i,t}^{\mathrm{GPU}}], [r_{G_{i,t}}]$
Require: If $a_{i,t} = 1$: private $[z_{i,t}], [r_{i,t}]$; public Merkle path $\pi_{i,t}$
Ensure: $y_{i,t} \in \{\mathsf{PASS}, \mathsf{FAIL}\}$

$[b_G] \leftarrow [H(\mathsf{tag}_G \| [r_{G_{i,t}}] \| \mathrm{Encode}([G_{i,t}^{\mathrm{GPU}}])) = c_{G_{i,t}}]$
if $a_{i,t} = 1$ **then**
  $[b_D] \leftarrow [H(\mathsf{tag}_D \| [r_{i,t}] \| \mathrm{Encode}([z_{i,t}])) = c_{i,t}] \wedge \mathsf{MerkleVerify}(c_{i,t}, \pi_{i,t}, c_{D_i})$
  $[G_{i,t}^{\mathrm{MPC}}] \leftarrow \mathrm{Train}_{\mathrm{MPC}}(W_t, [z_{i,t}]; \Pi)$
  $[P_{i,t}^{\mathrm{abs}}], [P_{i,t}^{\mathrm{rel}}], [\Delta_{i,t}^{\infty}] \leftarrow \mathsf{DiscrepancyProfile}([G_{i,t}^{\mathrm{GPU}}], [G_{i,t}^{\mathrm{MPC}}])$
  $[b_{\mathcal{B}}] \leftarrow \bigwedge_{p \in \mathcal{P}} \left([P_{i,t}^{\mathrm{abs}}(p) \le B_p^{\mathrm{abs}}] \wedge [P_{i,t}^{\mathrm{rel}}(p) \le B_p^{\mathrm{rel}}] \right) \wedge [\Delta_{i,t}^{\infty} \le B_{\infty}]$
  $[y_{i,t}] \leftarrow [b_G] \wedge [b_D] \wedge [b_{\mathcal{B}}]$
else
  $[y_{i,t}] \leftarrow [b_G]$
end if
$y_{i,t} \leftarrow \mathsf{Open}([y_{i,t}])$
return $y_{i,t}$

4 Experimental Evaluation

We evaluate OPFL on three representative models with different scales and training settings, as summarized in Table 1. Our evaluation covers both CNN and Transformer architectures, multiple optimizers, and both full model training and LoRA finetuning. The attack experiments evaluate the verification rule of OPFL independently of audit sampling and assume that all tested instances are audited. System-level security further relies on optimistic auditing with deposit-and-slashing incentives.

Table 1: Models and training configurations used in our evaluation.

Model	#Parameters	Training	Optimizer	Datasets
LeNet	431K	Full training	Momentum SGD	MNIST, Fashion-MNIST
BERT-Base	110M	LoRA, last two layers	AdamW	SST-2, CoLA
Qwen3	0.6B	LoRA, last two layers	SGD	Alpaca, Dolly

Our evaluation focuses on three questions. First, because the empirical boundary is the central verification criterion in OPFL, we study its stability and generalization. In particular, we examine whether a reliable boundary can be calibrated from only a small number of training steps and subsequently generalized across different training conditions, reducing the cost of the calibration stage. Second, we evaluate the effectiveness of OPFL against malicious training deviations. We consider gradient reuse, gradient sign reversal, label flipping, gradient scaling, and an adaptive PGD-based attack designed to evade the calibrated boundary. We further compare OPFL with RoFL-L_2, RoFL-L_∞, EIFFeL, and RiseFL by evaluating whether each method can reject the same adaptive malicious updates under its respective verification rule. Finally, we evaluate the system overhead of OPFL and compare its cost with full MPC-based FL and ZK-based verification.

Experimental setup. We implement OPFL in PyTorch with Secure Processing Unit (SPU) (Ma et al., 2023) as the MPC backend under a three-party semi-honest setting. Experiments use four

NVIDIA RTX 4090 GPUs and two Intel Xeon Platinum 8336C CPUs at 2.30 GHz, while cross-device generalization is evaluated on an additional RTX A4500 server.

4.1 Effectiveness against model poisoning

General attacks. We first evaluate whether OPFL can reject malicious updates while still accepting honest computation. Reporting ASR alone is insufficient, since a verifier that rejects every update would trivially achieve 0% ASR. We therefore report both ASR and false rejection rate (FRR). ASR is the fraction of attacked steps that both evade verification and achieve their intended malicious effect, while FRR is the fraction of unattacked honest steps that are incorrectly rejected:

$$\mathrm{FRR} = \frac{\#\text{ rejected honest steps}}{\#\text{ honest verification steps}}. \tag{10}$$

We consider four representative attack patterns adapted from prior work on free-riding, Byzantine gradient manipulation, data poisoning, and model-replacement attacks in federated and distributed learning (Fraboni et al., 2021; Damaskinos et al., 2018; Fang et al., 2020; Bagdasaryan et al., 2020; Lycklama et al., 2023). For each model, we evaluate 2,000 training steps and randomly select 20% of them for attack. Let $G_{i,t}^{\mathrm{GPU}}$ denote the honest client gradient defined previously, and let $\widetilde{G}_{i,t}$ denote the gradient submitted under attack. We consider: (i) *Gradient Reuse*, where the attacker computes a fresh honest gradient once every k steps and reuses the most recent gradient in the intermediate steps, with replay interval $k \in \{2, 5, 10\}$; (ii) *Reverse*, where $\widetilde{G}_{i,t} = -\gamma G_{i,t}^{\mathrm{GPU}}$ with $\gamma \in \{0.5, 1, 2\}$; (iii) *Label Flip*, where the supervision target is replaced while keeping the model state and input unchanged, and the resulting gradient is submitted; and (iv) *Amplify*, where $\widetilde{G}_{i,t} = \gamma G_{i,t}^{\mathrm{GPU}}$ with $\gamma \in \{5, 10\}$. We evaluate these attacks on LeNet, BERT-Base, and Qwen3-0.6B. As summarized in Table 2(a), OPFL achieves 0% ASR and 0% FRR across different settings.

PGD-based adaptive attack. We further evaluate OPFL against a white-box adaptive attack that explicitly optimizes against the verification rule. For each evaluated step t, let $G_{i,t}^{\mathrm{GPU}}$ be the honest gradient. The attacker submits

$$\widetilde{G}_{i,t} = G_{i,t}^{\mathrm{GPU}} + \delta_t, \qquad \|\delta_t\|_2 = \beta \|G_{i,t}^{\mathrm{GPU}}\|_2, \tag{11}$$

where $\beta \in \{0.01, 0.1, 0.5, 1, 2, 5, 10\}$ controls the attack strength. The attacker has access to the model state, optimizer state, and verification rule, and searches for a perturbation that changes a previously correct prediction while remaining acceptable to the verifier. Specifically, it maximizes the post-update prediction loss together with a soft penalty for violating the corresponding verification constraint. The final candidate must still pass the actual verification rule; the optimization penalty itself does not determine acceptance.

We construct the attack separately for each verifier, so each method is evaluated against an adversary adapted to its own acceptance rule. We compare OPFL with RoFL-L_2 and RoFL-L_∞ (Lycklama et al., 2023), EIFFeL-NormBall (Roy Chowdhury et al., 2022), and RiseFL-Gaussian (Zhu et al., 2024). These baselines are implemented as gradient-domain verification rules rather than full end-to-end reproductions of their cryptographic protocols. For OPFL, the attacker is optimized against the frozen deployed boundary and the same-step MPC reference $G_{i,t}^{\mathrm{MPC}}$. Detailed verifier definitions and attack parameters are provided in Appendix B.

We evaluate 200 instances per model for LeNet, BERT-Base, and Qwen3-0.6B, restricting evaluation to instances whose honest update produces the correct prediction. An attack succeeds only if the submitted gradient passes verification and changes that prediction from correct to incorrect. As summarized in Table 2(b), OPFL achieves 0% ASR for every model and attack strength. In contrast, the baseline rules remain vulnerable to adaptive perturbations. Full results for all values of β are reported in Appendix B. Together with the 0% FRR above, these results indicate that OPFL separates the evaluated malicious deviations from honest computation rather than simply applying an overly restrictive acceptance rule.

We additionally evaluate tail-sparse PGD attacks on LeNet, restricting perturbations to the top 1% or 10% of coordinates ranked by absolute loss-gradient magnitude, with the same attack settings and strengths. The 1% setting uses the full percentile grid, whereas the 10% setting retains only

checks at or below P90; both retain the infinity-norm guard. Without verification, ASR reaches 43.0% and 56.5%, respectively; with OPFL, it remains 0% at every tested strength (Appendix B). These results support robustness in the evaluated settings, but do not establish security for arbitrary percentile grids or support sets. Adding percentile checks while retaining existing thresholds can further constrain accepted discrepancies, at additional verification cost.

Table 2: Summary of attack evaluation. (a) General-attack results across all evaluated models and settings. (b) Maximum ASR (%) over PGD attack strengths $\beta \in \{0.01, 0.1, 0.5, 1, 2, 5, 10\}$.

(a) General attacks

Attack	Parameters	ASR	FRR
Gradient Reuse	$k \in \{2, 5, 10\}$	0%	0%
Reverse	$\gamma \in \{0.5, 1, 2\}$	0%	0%
Label Flip	full replacement	0%	0%
Amplify	$\gamma \in \{5, 10\}$	0%	0%

(b) Adaptive PGD

Verifier	LeNet	BERT	Qwen3
No verifier	100.0	90.0	35.0
RoFL-L_2	90.5	33.5	7.0
RoFL-L_∞	99.5	37.5	32.0
EIFFeL	89.5	32.0	7.0
RiseFL	97.99	34.5	12.42
OPFL	**0**	**0**	**0**

4.2 Stability and Generalization of the Empirical Boundary

Finally, we study how many calibration steps are needed to obtain a stable boundary and whether the resulting boundary generalizes beyond the calibration setting. For BERT-Base, we calibrate on SST-2 with sequence length 32 and batch size 1 using the first $n \in \{20, 50, 100, 200, 300, 500\}$ steps. Each resulting boundary is then frozen and evaluated without recalibration across held-out settings with different input sequence lengths, datasets, and GPUs.

As shown in Table 3, small calibration sets can yield overly tight boundaries. With 20 steps, the FRR reaches 32.4% on SST-2/64 and 2.0% on the RTX A4500 setting; with 50 steps, SST-2/64 still has a 4.1% FRR. In contrast, 100 calibration steps achieve 0% FRR across all four evaluated settings, and increasing the calibration size to 200–500 steps provides no further improvement while increasing calibration cost. We therefore adopt 100 steps as the default calibration budget.

Table 3: BERT-Base FRR versus calibration steps.

Steps	SST-2/32	SST-2/64	CoLA/32	A4500	Cost
20	2.33%	32.40%	0.10%	2.00%	0.2×
50	0%	4.10%	0%	0%	0.5×
100	**0%**	**0%**	**0%**	**0%**	1×
200	0%	0%	0%	0%	2×
300	0%	0%	0%	0%	3×
500	0%	0%	0%	0%	5×

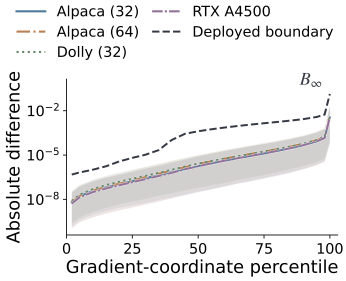


Figure 2: Generalization of the Qwen3 boundary.

We observe the same trend for Qwen3 and LeNet: 100 calibration steps yield 0% FRR across all evaluated generalization settings. Figure 2 shows the generalization of Qwen3 absolute boundary and tail guard calibrated from 100 steps across different settings, with additional results in Appendix A. These results support the stability and generalization of the calibrated boundary.

4.3 System Overhead and Cost Amortization

This section evaluates the computational cost of OPFL and the trade-off between audit rate and economic deterrence. Our goal is to quantify how optimistic auditing amortizes expensive private verification, rather than to provide a strict system-level comparison across different cryptographic backends.

Table 4: Verification cost and normalized deposit. K denotes 10^3. Deposit ratios are normalized to OPFL with $p = 0.10$ and use $p_{\mathrm{FR}} = 0.01$.

Method	Time (s)	Time ratio	Deposit	Deposit ratio
ZKSL-G	16.09K	8.38	0.00	0.00
ZKSL-LP-G	12.19K	6.35	0.00	0.00
Full MPC-based FL	1.92K	1.00	0.00	0.00
OPFL, $p = 0.10$	192.35	0.10	1.11	1.00
OPFL, $p = 0.05$	96.32	0.05	1.12	1.01
OPFL, $p = 0.01$	19.49	0.01	1.76	1.59

We construct a five-client FedSGD workload using LeNet with 0.0617M parameters, following the configuration evaluated in ZKSL (Zheng et al., 2026). Each client performs one local gradient step per round with batch size 1, and the server equally aggregates the five gradients. We consider 1,000 rounds, corresponding to 5,000 client gradient computations and 1,000 aggregations.

We compare ZKSL-G, ZKSL-LP-G, full MPC-based FL, and OPFL with audit rates $p \in \{0.10, 0.05, 0.01\}$. ZKSL costs are extrapolated from the original work, while MPC costs are estimated from our SPU implementation. The reported time includes verification and aggregation but excludes initialization, communication, commitment, and one-time boundary calibration costs.

Table 4 shows that the cost of OPFL decreases nearly linearly with the audit rate. At $p = 0.01$, OPFL is approximately $98.6\times$ faster than full MPC-based FL and $625.5\times$ faster than ZKSL-LP-G under this workload. These results should be interpreted as a cost analysis rather than a strict end-to-end speedup because the measurements use different hardware environments. They illustrate that OPFL pays the cost of expensive private replay only for a small fraction of committed computations.

Audit rate and deposit trade-off. A lower audit rate reduces verification cost but also decreases the probability of detecting deviations. Consider a client that skips 100 of its 1,000 rounds, with the saved computation normalized to one unit. If each round is independently audited with probability p, the probability of detecting at least one skipped computation is $\Pr_{\mathrm{det}}(p) = 1 - (1 - p)^{100}$.

Because the verification boundary is empirically calibrated, we conservatively allow an overall false-rejection probability $p_{\mathrm{FR}} = 0.01$ for an honest client. Let $\mathsf{Stake}(p)$ denote the refundable deposit. Relative to honest execution, the expected additional utility of deviation is $\Delta U(p) = 1-(\Pr_{\mathrm{det}}(p)-p_{\mathrm{FR}})\mathsf{Stake}(p)$. We use a 10% deterrence margin and set $\mathsf{Stake}(p) = 1.1/(\Pr_{\mathrm{det}}(p) - p_{\mathrm{FR}})$, which gives $\Delta U(p) = -0.1$. As shown in Table 4, reducing p from 0.10 to 0.05 has little effect on the required deposit, while $p = 0.01$ increases it to $1.59\times$ the deposit required at $p = 0.10$.

Deposit considerations. This analysis illustrates the trade-off between computational auditing and economic deterrence rather than prescribing a universal deposit value. Malicious behavior may provide benefits beyond saved computation and can be difficult to quantify. Deployed blockchain systems therefore often require nontrivial collateral and penalize provable misbehavior (Ethereum Foundation; Offchain Labs). For example, Ethereum validators are subject to stake slashing, while Arbitrum uses stake-backed claims and challenge penalties. A deployment of OPFL can therefore choose the deposit conservatively according to the expected attack benefit and measured reliability of the deployed boundary.

5 Conclusion

We presented OPFL, an optimistic verification framework for privacy-preserving federated learning. OPFL combines private MPC replay with an empirical gradient-discrepancy boundary to verify sampled client computations without requiring bitwise agreement across execution environments. Our experiments show that the boundary remains stable across diverse training settings and detects the evaluated direct and adaptive attacks while maintaining a low FRR. By applying an optimistic verification mechanism, OPFL also substantially reduces verification cost. We believe this provides a practical direction for verifying federated learning while preserving data privacy.

AI USE STATEMENT

We used generative AI tools to improve the clarity and readability of the manuscript, assist with literature searches, and support code development and debugging. The authors reviewed and revised the AI-assisted text, checked the cited sources for accuracy and relevance, and reviewed and tested the AI-assisted code. The authors take responsibility for the final manuscript, including all claims, references, code, and reported results.

REFERENCES

Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pp. 2938–2948. PMLR, 2020.

James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. ACORN: Input validation for secure aggregation. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 4805–4822. USENIX Association, 2023.

Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191. ACM, 2017. doi: 10.1145/3133956.3133982.

Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. FLTrust: Byzantine-robust federated learning via trust bootstrapping. In *Network and Distributed System Security Symposium (NDSS)*, 2021. doi: 10.14722/ndss.2021.24434.

Sylvain Collange, David Defour, Stef Graillat, and Roman Iakymchuk. Numerical reproducibility for the parallel reduction on multi- and many-core architectures. *Parallel Computing*, 49:83–97, 2015. doi: 10.1016/j.parco.2015.09.001.

Georgios Damaskinos, El-Mahdi El-Mhamdi, Rachid Guerraoui, Rhicheek Patra, and Mahsa Taziki. Asynchronous byzantine machine learning (the case of SGD). In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1145–1154. PMLR, 2018.

James Demmel and Hong Diep Nguyen. Parallel reproducible summation. *IEEE Transactions on Computers*, 64(7):2060–2070, 2015. doi: 10.1109/TC.2014.2345391.

Ethereum Foundation. Proof-of-stake rewards and penalties. `https://ethereum.org/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/`. Accessed 2026.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 1605–1622. USENIX Association, 2020.

Yann Fraboni, Richard Vidal, and Marco Lorenzi. Free-rider attacks on model aggregation in federated learning. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pp. 1846–1854. PMLR, 2021.

Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles*, pp. 51–68. ACM, 2017. doi: 10.1145/3132747.3132757.

David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. doi: 10.1145/103162.103163.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1039–1056, 2021. doi: 10.1109/SP40001.2021.00106.

Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1353–1370. USENIX Association, 2018.

Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology – CRYPTO 2017*, volume 10401 of *Lecture Notes in Computer Science*, pp. 357–388. Springer, 2017. doi: 10.1007/978-3-319-63688-7_12.

Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. RoFL: Robustness of secure federated learning. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 453–476, 2023. doi: 10.1109/SP46215.2023.10179400.

Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. SecretFlow-SPU: A performant and User-Friendly framework for Privacy-Preserving machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pp. 17–33. USENIX Association, 2023.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, 2017.

Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 35–52. ACM, 2018. doi: 10.1145/3243734.3243760.

Soheila Molaei, Anshul Thakur, Ghazaleh Niknam, Andrew Soltan, Hadi Zare, and David A. Clifton. Federated learning for heterogeneous electronic health records utilising augmented temporal graph attention networks. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics*, volume 238 of *Proceedings of Machine Learning Research*, pp. 1342–1350. PMLR, 2024.

Offchain Labs. Arbitrum nitro: A second-generation optimistic rollup. `https://docs.arbitrum.io/nitro-whitepaper.pdf`. Accessed 2026.

Xiaoyi Pang, Chenxu Zhao, Zhibo Wang, Jiahui Hu, Yinggui Wang, Lei Wang, Tao Wei, Kui Ren, and Chun Chen. PoiSAFL: Scalable poisoning attack framework to byzantine-resilient semi-asynchronous federated learning. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 6461–6479. USENIX Association, 2025.

Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. EIFFeL: Ensuring integrity for federated learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2535–2549, 2022. doi: 10.1145/3548606.3560611.

Megha Srivastava, Simran Arora, and Dan Boneh. Optimistic verifiable training by controlling hardware nondeterminism. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95639–95661, 2024. doi: 10.52202/079017-3030.

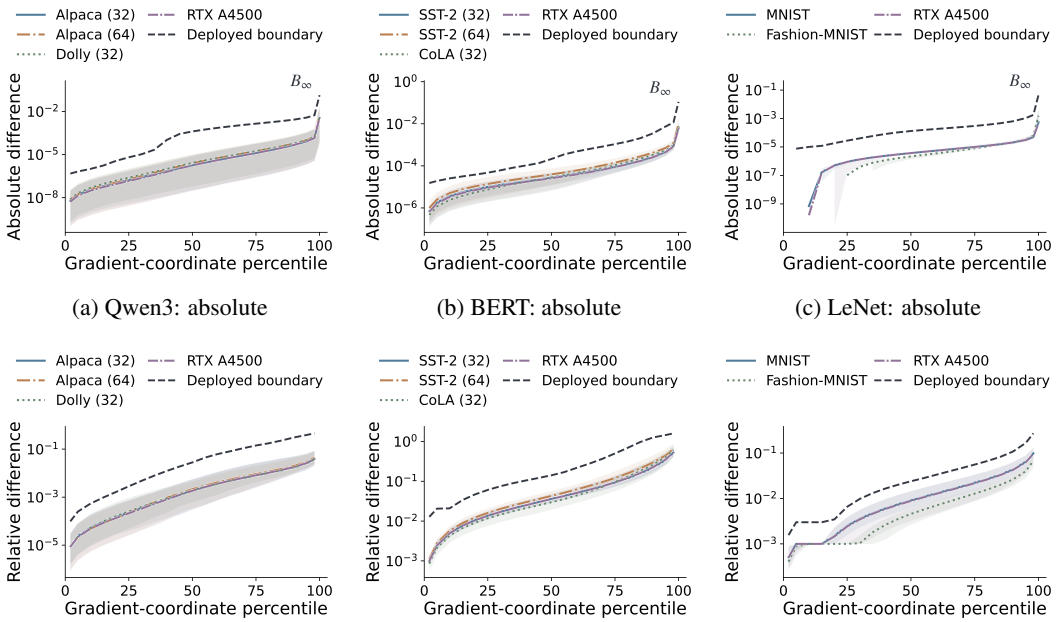
Yixiao Zheng, Changzheng Wei, Xiaodong Qi, Hanghang Wu, Yuhan Wu, Li Lin, Tianmin Song, Ying Yan, Yanqing Yang, Zhao Zhang, Cheqing Jin, and Aoying Zhou. ZKSL: Verifiable and efficient split federated learning via asynchronous zero-knowledge proofs. In *Network and Distributed System Security Symposium (NDSS)*, 2026. doi: 10.14722/ndss.2026.242008.

Yizheng Zhu, Yuncheng Wu, Zhaojing Luo, Beng Chin Ooi, and Xiaokui Xiao. Secure and verifiable data collaboration with low-cost zero-knowledge proofs. *Proceedings of the VLDB Endowment*, 17(9):2321–2334, 2024. doi: 10.14778/3665844.3665860.

Haomin Zhuang, Mingxian Yu, Hao Wang, Yang Hua, Jian Li, and Xu Yuan. Backdoor federated learning by poisoning backdoor-critical layers. In *International Conference on Learning Representations*, 2024.

A Additional Boundary Generalization Results

Figure 3 complements Figure 2 by providing a visual comparison of the boundary and gradient differences for all three models: Qwen3, LeNet, and BERT-Base. The deployed boundary is calibrated using the first 100 training steps and is then frozen for evaluation. We use a safety factor of $\alpha = 4$ for Qwen3 and $\alpha = 3$ for LeNet. Across all evaluated generalization settings, the resulting boundaries achieve 0% FRR, consistent with the trend observed for BERT-Base.



(a) Qwen3: absolute (b) BERT: absolute (c) LeNet: absolute

(d) Qwen3: relative (e) BERT: relative (f) LeNet: relative

Figure 3: Generalization of the deployed boundaries calibrated from 100 steps for Qwen3, BERT, and LeNet. The top and bottom rows show absolute and relative gradient discrepancies, respectively. Solid lines and shaded regions show the median and percentile range across evaluation steps, respectively; dashed lines indicate the deployed boundaries.

B Adaptive PGD Attack Details

Attack construction. We evaluate a white-box adaptive PGD attack against each verification rule. For an evaluated client-step pair (i, t), let $G_{i,t}^{\mathrm{GPU}}$ denote the honest gradient. The attacker submits

$$\widetilde{G}_{i,t} = G_{i,t}^{\mathrm{GPU}} + \delta_t, \qquad \|\delta_t\|_2 = \beta \|G_{i,t}^{\mathrm{GPU}}\|_2, \tag{12}$$

where

$$\beta \in \{0.01, 0.1, 0.5, 1, 2, 5, 10\}.$$

Let $U_t(G)$ denote the model parameters obtained by applying gradient G from the saved model parameters and optimizer state at step t. The attacker searches for a perturbation that maximizes the post-update prediction loss while remaining compatible with the target verifier:

$$\max_{\|\delta\|_2=\beta\|G_{i,t}^{\mathrm{GPU}}\|_2} \ell\Big(f_{U_t(G_{i,t}^{\mathrm{GPU}}+\delta)}(x_t), y_t\Big) - \lambda \mathcal{P}_V\left(G_{i,t}^{\mathrm{GPU}} + \delta\right), \tag{13}$$

where $\mathcal{P}_V$ is a differentiable penalty associated with verifier V and $\lambda = 100$.

We optimize each verifier separately rather than generating a single unconstrained attack and evaluating it against all methods. Optimization uses normalized gradient ascent for 20 iterations. After each iteration, the perturbation is projected back to the fixed-radius sphere. The step size is set to one fifth of the perturbation radius. Each attack uses three initializations: one aligned with the prediction-loss gradient and two random directions. We use random seed 42. Importantly, the soft penalty is used only to guide optimization; the final candidate must still pass the actual verification rule.

We evaluate 200 instances for each model. All selected instances are required to produce a correct prediction after the honest update. For LeNet and BERT, an attack succeeds when the post-update class prediction changes from correct to incorrect. For Qwen3, we evaluate the final supervised token position and require the originally correct token prediction to become incorrect. Thus, the attack objective is an untargeted prediction flip.

Verification baselines. We compare OPFL against gradient-domain adaptations of RoFL-L_2, RoFL-L_∞, EIFFeL-NormBall, and RiseFL-Gaussian, together with a no-verification baseline. These experiments compare the corresponding gradient acceptance rules rather than reproducing the complete end-to-end cryptographic protocols.

Using the first 100 honest training steps, we define

$$B_2 = \max_{t<100} \left\|G_{i,t}^{\mathrm{GPU}}\right\|_2, \qquad B_\infty^{\mathrm{RoFL}} = \max_{t<100} \left\|G_{i,t}^{\mathrm{GPU}}\right\|_\infty, \tag{14}$$

and

$$R = \max_{t<100} \left\|G_{i,t}^{\mathrm{GPU}} - V_t\right\|_2, \tag{15}$$

where V_t is a reference gradient computed from a fixed public reference dataset at the saved model state.

Table 5: Verification rules used in the adaptive PGD evaluation.

Verifier	**Acceptance rule**
No verifier	Always accept
RoFL-L_2	$\|\widetilde{G}_{i,t}\|_2 \le B_2$
RoFL-L_∞	$\|\widetilde{G}_{i,t}\|_\infty \le B_\infty^{\mathrm{RoFL}}$
EIFFeL-NormBall	$\|\widetilde{G}_{i,t} - V_t\|_2 \le R$
RiseFL-Gaussian	Gaussian projection norm test
OPFL	$P_{i,t}^{\mathrm{abs}}(p) \le B_p^{\mathrm{abs}}$ and $P_{i,t}^{\mathrm{rel}}(p) \le B_p^{\mathrm{rel}}$, $\forall p \in \mathcal{P}$ and $\Delta_{i,t}^{\infty} \le B_\infty$

The corresponding acceptance rules are summarized in Table 5. All constraints are applied to the submitted gradient $\widetilde{G}_{i,t}$ rather than to the perturbation δ_t.

For the deterministic norm-based baselines, the adaptive penalty is defined as

$$\mathcal{P}_V(u) = \left[\max\left(\frac{c_V(u)}{b_V} - 1, 0\right)\right]^2, \tag{16}$$

where $c_V(u)$ denotes the corresponding norm statistic and b_V its threshold.

For RiseFL-Gaussian, we use $K = 1000$ Gaussian projections and $\epsilon = 2^{-128}$. For a fixed submitted gradient u, the quantile q, the analytical acceptance probability $p_{\mathrm{acc}}(u)$ of u, and the penalty $\mathcal{P}_V(u)$ used in the adaptive objective are defined as

$$\begin{aligned} q &= F^{-1}_{\chi^2_K}(1-\epsilon), \\ p_{\mathrm{acc}}(u) &= F_{\chi^2_K}\left(\frac{qB_2^2}{\|u\|_2^2}\right), \\ \mathcal{P}_V(u) &= \left(1 - p_{\mathrm{acc}}(u)\right)^2. \end{aligned} \tag{17}$$

After optimization, the selected candidate is evaluated using an independent randomized audit.

For OPFL, the submitted gradient is compared with the same-step MPC replay $G_{i,t}^{\mathrm{MPC}}$. The deployed empirical boundary is frozen before the attack and is not recalibrated using adversarial samples. We use safety factors $(\alpha_{\mathrm{abs}}, \alpha_{\mathrm{rel}}, \alpha_\infty) = (3, 3, 3)$ for LeNet, $(3, 3, 6)$ for BERT, and $(4, 4, 4)$ for Qwen3.

Attack success rate. An attack is considered successful only if it both passes verification and changes an originally correct prediction to an incorrect one. For $N = 200$ evaluated instances, we report

$$\mathrm{ASR} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\left[\text{prediction flip}\right] p_{\mathrm{acc}}\left(\widetilde{G}_{i,t}^{(n)}\right). \tag{18}$$

For deterministic verifiers, $p_{\mathrm{acc}} \in \{0, 1\}$. For RiseFL, we report the expected ASR weighted by its analytical acceptance probability. If optimization fails to find an acceptable malicious candidate and falls back to the honest gradient, the instance is counted as an attack failure and remains in the denominator.

Full results. Tables 6, 7, and 8 report the complete results for all tested attack strengths.

Table 6: Adaptive PGD attack success rate (%) on LeNet.

β	**None**	**RoFL-L_2**	**RoFL-L_∞**	**EIFFeL**	**RiseFL**	**OPFL**
0.01	0.0	0.0	0.0	0.0	0.0	0.0
0.1	4.5	4.5	4.5	4.5	4.5	0.0
0.5	29.5	29.5	29.5	29.5	29.5	0.0
1	57.0	57.0	57.0	57.0	57.0	0.0
2	90.0	90.0	90.0	89.5	90.0	0.0
5	99.0	90.5	99.0	68.0	97.99	0.0
10	100.0	37.0	99.5	13.0	71.48	0.0

Table 7: Adaptive PGD attack success rate (%) on BERT-Base.

β	**None**	**RoFL-L_2**	**RoFL-L_∞**	**EIFFeL**	**RiseFL**	**OPFL**
0.01	0.0	0.0	0.0	0.0	0.0	0.0
0.1	4.5	4.5	4.5	4.5	4.5	0.0
0.5	20.0	20.0	20.0	20.0	20.0	0.0
1	35.5	33.5	35.0	32.0	34.5	0.0
2	49.5	33.5	37.5	27.0	21.0	0.0
5	72.0	0.0	14.0	0.0	0.0	0.0
10	90.0	0.0	0.5	0.0	0.0	0.0

Table 8: Adaptive PGD attack success rate (%) on Qwen3-0.6B.

β	**None**	**RoFL-L_2**	**RoFL-L_∞**	**EIFFeL**	**RiseFL**	**OPFL**
0.01	0.0	0.0	0.0	0.0	0.0	0.0
0.1	0.0	0.0	0.0	0.0	0.0	0.0
0.5	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0
2	7.0	7.0	7.0	7.0	7.0	0.0
5	20.0	5.5	20.0	5.5	12.42	0.0
10	35.0	1.0	32.0	1.0	2.99	0.0

Across all three models, OPFL maintains 0% ASR for every tested attack strength. The norm-based baselines reject some large perturbations, but remain vulnerable to adaptive attacks that satisfy their respective acceptance constraints, particularly at moderate perturbation strengths.

Tail-sparse PGD attacks. Table 9 reports the LeNet results for perturbations restricted to the top 1% or 10% of coordinates ranked by absolute loss-gradient magnitude, using the attack settings and strengths described above. This experiment uses independently specified verification grids. For the 1% support setting, we use $\mathcal{P}_{1\%} = \{0.02, 0.05, 0.10, 0.15, \ldots, 0.90, 0.95, 0.98, 0.99\}$. For the 10% support setting, we use $\mathcal{P}_{10\%} = \{p \in \mathcal{P}_{1\%} : p \leq 0.90\}$. Both settings retain the infinity-norm guard B_∞.

Table 9: Tail-sparse PGD attack success rate (%) on LeNet.

	1% support		**10% support**	
β	No verifier	OPFL	No verifier	OPFL
0.01	0.0	0.0	0.0	0.0
0.1	0.0	0.0	0.0	0.0
0.5	0.0	0.0	0.0	0.0
1	1.5	0.0	4.5	0.0
2	6.5	0.0	12.0	0.0
5	24.5	0.0	28.5	0.0
10	43.0	0.0	56.5	0.0