

ZORRO: Zero-Knowledge Robustness and Privacy for Split Learning (Full Version)

Nojan Sheybani*
nsheyban@ucsd.edu

University of California San Diego
La Jolla, USA

Phillip Rieger
phillip.rieger@trust.tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

Alessandro Pegaso*
alessandro.pegasо@trust.tu-
darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

Elissa Mollakuqe
elissa.mollakuqe@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

Jonathan Knauer*
jonathan.knauer@stud.tu-
darmstadt.de

Technical University of Darmstadt
Darmstadt, Germany

Farinaz Koushanfar
fkoushanfar@ucsd.edu
University of California San Diego
La Jolla, USA

Ahmad-Reza Sadeghi
ahmad.sadeghi@trust.tu-
darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

on the client-side an overhead of less than 10 seconds.

CCS Concepts

- Security and privacy → Distributed systems security; Cryptanalysis and other attacks;
- Computing methodologies → Machine learning; Distributed artificial intelligence.

Keywords

Split Learning, Backdoor Defense, Poisoning Defense, Zero-Knowledge-Proof, ZKP, Discrete Cosine Transformation

ACM Reference Format:

Nojan Sheybani, Alessandro Pegaso, Jonathan Knauer, Phillip Rieger, Elissa Mollakuqe, Farinaz Koushanfar, and Ahmad-Reza Sadeghi. 2025. ZORRO: Zero-Knowledge Robustness and Privacy for Split Learning (Full Version). In *Proceedings of the 2025 ACM SIGSAC Conf. on Computer and Communications Security (CCS '25), Oct. 13–17, 2025, Taipei, Taiwan*. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3719027.3765160>

1 Introduction

Due to its ability to succeed in increasingly complex tasks, Deep Learning has been heavily applied in ubiquitous computing systems, including domains with sensitive data. A key challenge in such applications is the availability of sufficient training data, especially in times of rising privacy concerns and regulations such as the GDPR [47], HIPAA [48], and CCPA [12]. To address this, in the past, schemes such as Federated Learning have been developed where the training process is outsourced to clients holding the data [31]. However, as model sizes continue to grow, reaching hundreds of billions of parameters [2], many data holders lack the computational resources required for training while also being unable or unwilling to share their data.

Split Learning (SL) offers a promising solution. It is a distributed learning paradigm that allows clients to utilize external computational resources without exposing their data. In SL, the DNN is

*These authors contributed equally to this work.



This work is licensed under a Creative Commons Attribution 4.0 International License.

CCS '25, Taipei, Taiwan

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1525-9/2025/10

<https://doi.org/10.1145/3719027.3765160>

divided into partitions distributed across different entities [21]. In the commonly used U-shaped configuration, the initial and final layers reside on the client side, while the computationally intensive middle layers are processed on a powerful server. During training and inference, hidden representations (forward pass) and gradients (backward pass) are exchanged only at partition boundaries [49, 58]. By rotating the server-side model component among clients, SL facilitates collaborative training while maintaining data privacy [46]. This architecture not only enables efficient training and inference for large DNNs, but also significantly mitigates the risk of inference attacks, as clients do not share the model components responsible for sensitive feature extraction.

Backdoor Attacks and Defenses. While splitting the model between clients and servers improves both privacy and computational efficiency, it also introduces a critical limitation since no single party has access to the full model, making it difficult to detect potential manipulations. Recent work has explored backdoor attack vectors, originating either from the server side [37, 46, 61] or the client side [4, 23, 60]. Client-side attacks are particularly concerning, as adversarial clients, especially in mobile or distributed settings, can operate anonymously and face minimal reputational risk if detected. In contrast, the server is typically identifiable and has an intrinsic motivation to maintain its reputation by producing reliable models.

Only a few approaches have been made to mitigate backdoor attacks in SL. Pu et al. consider a malicious server that aims to inject a backdoor [37]. Only limited attention is given to client attacks [4]. Rieger et al. introduced SafeSplit, a server-side defense mechanism designed to mitigate backdoor threats in SL [39]. However, such server-centric defenses often impose significant computational overhead for the server and face scalability challenges, particularly when operations as pairwise distance computations are involved.

Goals and Contributions. We introduce \mathcal{ZORRO} , the first client-side defense mechanism designed specifically to protect SL from backdoor attacks performed by malicious clients. To understand our approach intuitively, consider how drivers place a warning triangle after an accident to alert others, allowing them to bypass danger safely. Similarly, in \mathcal{ZORRO} , each client proactively examines its own trained model segments and prevents sharing if it detects signs of poisoning. Built on zero-knowledge proofs (ZKPs) to verify the correct defense execution, \mathcal{ZORRO} forces attackers into a dilemma. They must either follow our defense protocol and eliminate their malicious updates or deviate and consequently fail the verification step.

Unlike previous solutions, which primarily rely on the central server for security checks, we empower clients themselves to carry out detailed inspections of their locally trained models. By leveraging frequency-domain analysis, a technique effective in revealing subtle manipulations in DNNs, we can detect poisoned model updates regardless of how many clients are compromised. To preserve model integrity and limit the propagation of poisoned contributions, each client forwards only a subset of the most recent model checkpoints to the succeeding client. The local inspection process ensures that poisoned models are filtered before being shared, mitigating the impact of backdoor attacks. At the same time, this approach minimizes the number of clients that gain visibility into any single model update, contributing to stronger privacy preservation within the collaborative training process.

To guarantee trustworthy execution of these checks without

exposing sensitive data, we employ interactive zero-knowledge proofs (ZKPs). They allow clients (the "provers") to convincingly demonstrate to the server (the "verifier") that they have executed the defense protocol correctly without revealing private details such as model parameters. In our SL setup, this means clients prove that their locally trained models have been checked for poisoning and processed according to the defense algorithm, while the server can verify these proofs without learning anything about the clients' data or models. Specifically, we use ZKPs based on vector oblivious linear evaluation (VOLE), enabling robust verification and strong security guarantees even in environments where malicious participants are present. A modular approach is taken to build \mathcal{ZORRO} to ensure that the scalable and efficient implementation can be generalized to new client-based SL defense schemes. Every part of the implementation can be swapped to achieve different goals in an efficient manner, such as replacing the ℓ_1 -norm module with an ℓ_2 -norm module.

Our main contributions are as follows:

- We propose \mathcal{ZORRO} , the first client-side backdoor defense against client-side backdoor attacks in SL. Built on zero-knowledge proofs (ZKPs), we create a dilemma for attackers, forcing them to reveal themselves and warn other clients or risk being dropped from the training pipeline altogether. Unlike prior work, \mathcal{ZORRO} delegates not only the training of the input and output layers but also the defense mechanism itself to the clients. This enables a fine-grained inspection of local training contributions while preserving the privacy of sensitive model parameters (Sect. 4.1).
- We propose a novel backdoor detection mechanism that analyzes the frequency-domain representation of model updates. This method enables the precise identification of backdoor artifacts without making assumptions about the proportion of malicious clients (Sect. 4.3).
- We design an interactive, modular, VOLE-based zero-knowledge proof (ZKP) protocol that allows honest clients to verify the correct execution of the defense scheme of untrusted or potentially malicious participants, ensuring protocol compliance without revealing sensitive model parameters. Its modular design enables general verification of client-side defenses, making the protocol adaptable also to future SL defense schemes (Sect. 4.5).
- We conduct an extensive empirical evaluation of \mathcal{ZORRO} across multiple datasets, model architectures, and data distributions. Our method is tested against both naive and defense-aware adversaries, demonstrating robust performance in realistic threat settings (Sect. 5).

2 Preliminaries

2.1 Split Learning

Split Learning is a collaborative machine learning framework particularly suited for scenarios where multiple clients, such as mobile devices or edge devices, hold sensitive data they do not want to share directly. In Split Learning, we assume a setting involving N clients $C = C_1, C_2, \dots, C_N$, each of whom possesses a private dataset $Data_i$. These clients jointly train a Deep Neural Network (DNN) in coordination with a central server. A key characteristic of SL is the model-splitting approach, where the DNN is partitioned

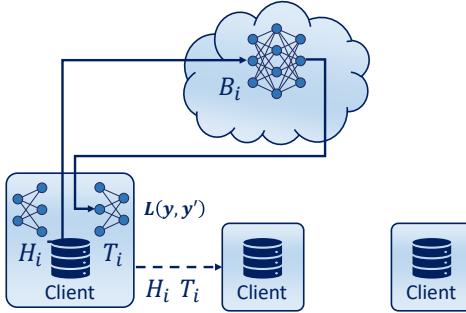


Figure 1: Overview of a Split Learning (SL) System.

into two or three sections. The first proposed split was Vanilla Split Learning [49], with an initial model executed locally on each client and a Backbone model run on the central server. In this work, we will instead consider the U-shaped Split Learning framework [30], as depicted in Figure 1, where each client C_i computes the forward pass through a small number of initial layers (the head, denoted H_i), and transmits the resulting activations (smashed data) to the server. The server then continues the forward pass through the majority of the remaining layers (the backbone, denoted B_i) and forwards its intermediate representation back to the client which feeds it to the final output layers (the tail, denoted T_i) to calculate the output. The client then calculates the Loss $L(y, y')$ based on the sample labels and their corresponding outputs before performing analogously the backward propagation across the model partitions. This design reduces the computational burden on edge devices and helps preserve data privacy, as the raw input never leaves the client device in both the Vanilla and U-shaped. Furthermore, the U-shaped design has the advantage of keeping the sample labels on the client device, while in the Vanilla split, they are shared with the server for backpropagation. After the training iteration, client C_i shares its Head H_i and Tail T_i with the next client C_{i+1} , as shown in Figure 1, which uses them as the starting model for its training iteration [46].

2.2 Zero-Knowledge Proofs

Zero-knowledge proofs (ZKPs) are cryptographic protocols that allow a prover \mathcal{P} to convince a verifier \mathcal{V} of a statement's truth without revealing any information beyond its validity. These protocols are also often used for the verifiable computation of a task. In a ZKP system, \mathcal{P} demonstrates knowledge of a secret value w (witness) that satisfies a computation C , while \mathcal{V} can confirm the computation's correctness without learning the secret. Formally, a ZKP system involves a prover \mathcal{P} and verifier \mathcal{V} executing probabilistic polynomial-time algorithms that adhere to three key properties: completeness, soundness, and zero-knowledge. These properties essentially ensure that a malicious prover cannot generate a valid proof of a false statement and, no matter what, no information about the secret value is revealed.

ZKPs can be categorized as interactive or non-interactive. Interactive ZKPs require multiple communication rounds, yielding proofs verifiable only by the specific verifier involved. Non-interactive ZKPs (NIZKs) allow a prover to generate a single, publicly-verifiable proof [7, 8]. Recent NIZK advancements like zk-SNARKs [8] and

zk-STARKs [7] offer compact, efficient proofs but rely on computationally intensive setups through a trusted third party or a computationally powerful verifier, hindering practicality despite widespread adoption. NIZKs also demand powerful provers for succinctness. While lacking public verifiability, interactive ZKPs achieve significantly better efficiency and scalability, suiting computationally complex tasks. In this work, the ability to interact with two \mathcal{V} s in parallel makes interactive ZK an elegant solution that balances performance and communication.

Vector Oblivious Linear Evaluation (VOLE)-based ZK protocols offer post-quantum security and efficiency through $\mathcal{P}\text{-}\mathcal{V}$ interaction [11]. These interactive protocols achieve high efficiency and scalability using IT-MAC commitments, efficiently implemented with VOLE [6]. Our work uses Wolverine [53], a state-of-the-art VOLE-based ZK protocol minimizing prover/verifier complexity interactively. In Wolverine, IT-MACs commit to authenticated wire values within arithmetic or boolean circuits (C). The prover \mathcal{P} proves knowledge of a private vector w , representing the inputs, outputs, and intermediate values of C . This is used to prove $C(w) = 1$, while proving the consistency of protocol values x [53]. Efficient boolean and arithmetic conversions [54] enable building efficient mixed-computation solutions. Due to the inherent interactivity, these protocols yield *designated-verifier* proofs, limiting verification by arbitrary parties. This limitation is acceptable for SL's peer-to-peer training. Using VOLE-based ZK integrates privacy, integrity, and robustness guarantees into ZORRO, enabling unparalleled scalability and efficiency.

3 Problem Setting

3.1 System Setting

In the following, we consider a system consisting of N clients, each holding a private dataset. The clients collaboratively train a DNN without revealing their data. Due to limited local computational capabilities, the clients employ SL, coordinated by a central server having strong computational resources. Aligned with prior work [39, 46], we focus on a U-shaped SL configuration, where the head and tail of the DNN are located on the client side, while the backbone is hosted on the server.

To avoid any privacy leakage through white-box model inference attacks [24, 34], no party must have access to the entire model but only parts of it. Thus, the clients will not share trained parameters of the head or tail with the server.

3.2 Threat Model

Objective: We consider an attacker \mathcal{A} that seeks to inject a backdoor into the collaboratively trained model. The backdoor causes the model to mispredict all samples $x \in \mathcal{X}_I$ showing an adversary chosen backdoor trigger I as a backdoor target class $\mathcal{T}_{\mathcal{A}}$.

Capabilities: To perform the attack, we assume \mathcal{A} can fully control $N_{\mathcal{A}}$ clients. Thus, \mathcal{A} can arbitrarily modify the local datasets and training procedure of these clients and also manually tamper all data the client processes, such as the head and tail. To ensure a strong adversary setting, we assume that all malicious clients know each other and are coordinated by \mathcal{A} .

Assumptions: Our assumptions are standard and aligned with

existing work [39]. \mathcal{A} does not control any of the benign clients, nor does it have knowledge about their local datasets. We assume that there is no collusion between malicious clients and the server, as the server's reputation would be damaged in case of an attack becoming public, while the clients can be anonymous mobile devices. Further, \mathcal{A} must avoid detection as otherwise the training process can be repeated. As such, the attack must not notably affect the model's utility. Further, an attacker is spotted and can be excluded from the training process if a deviation from the required protocol is detected, such as a failing ZKP, and this can be traced back to a specific client.

3.3 Requirements and Challenges

An effective and practical defense against backdoor attacks needs to fulfill a number of requirements:

R1 – Mitigate Backdoor: First, an effective defense needs to prevent \mathcal{A} from injecting the backdoor into the model.

R2 – Avoid Utility Degradation: To be practical, the defense scheme must not negatively affect the model's utility, i.e., predictions for untriggered samples.

R3 – No model sharing of head and tail with server: To ensure privacy and prevent that a party is enabled to analyze the parameters of the full model to violate clients' privacy, the defense must not share head or tail with the server.

A backdoor defense that effectively mitigates client-side backdoor attacks and fulfills these requirements faces a number of challenges:

C1 – Detecting Poisoned Contributions Without Prior Knowledge. A defining characteristic of backdoor attacks is their stealthiness. Given the large number of possible trigger patterns and target labels, defenders cannot make assumptions about the backdoor behavior, preventing a targeted inspection of the models. Furthermore, the data of different clients can differ from each other, be non-identical and independently distributed (non-IID). A major challenge, therefore, is distinguishing poisoned model updates from benign ones trained on non-IID data in the absence of knowledge about the backdoor.

C2 – Partial Observability and Non-Comparable Models in Split Learning. In SL, clients can only access specific partitions of the model (see R3), which limits their ability to perform comprehensive inspections. Additionally, due to the sequential nature of SL [39], each client trains its model update from a different checkpoint, resulting in models that are not directly comparable. A challenge is, therefore, how to reliably identify poisoned contributions when only partial model information is observable and the model updates are inherently non-comparable due to sequential training.

C3 – Verifiable and Privacy-Preserving Client-Side Defense. To offload computation from the server and enable scalable verification, we consider client-side defenses where each client analyzes its own model partition. However, malicious clients may manipulate this process to falsely validate poisoned updates. Also, requiring clients to inspect each other's models introduces additional vulnerabilities, especially in the presence of colluding adversaries. A challenge is thus to enforce the correct execution of the defense mechanism and ensure that malicious behavior is exposed, while preserving client privacy and maintaining robustness against collusion.

4 ZORRO

In the following, we outline the high-level design of ZORRO (Sect. 4.1) and ZORRO's flow (Sect. 4.2), before elaborating on how it inspects the client-side models for poisoned artifacts (Sect. 4.3), how ZORRO employs interactive ZKPs to guarantee the defense correctness (Sect. 4.5) and discuss ZORRO's security parameters (Sect. 4.6). In Sect. 4.7, we describe the details of the privacy-preserving and efficient end-to-end implementation of ZORRO. The high-level approach towards securing SL training with ZORRO is shown in Fig. 2.

4.1 High-Level Overview

ZORRO ensures the integrity of the collaborative training process through a client-side model inspection. The ZKP enforces that clients inspect their own model for backdoor artifacts in the frequency domain and malicious clients expose themselves.

When it is client C_i 's turn, it receives a set of k model checkpoints, along with a pointer BM indicating the most reliable model to continue training from. The client uses the model BM to perform local training via standard U-shaped Split Learning and appends the resulting client-side model partition to the list, resulting in a total of $k + 1$ models. To assess the integrity of each model, a poisoning detection function is applied, assigning a risk score to every model in the list. The model with the highest poisoning score is subsequently removed, and BM is updated to reference the most trustworthy remaining model. To ensure verifiable compliance with the defense mechanism, the client generates an interactive zero-knowledge proof π , which attests to the correct application of the protocol. This proof, together with the updated pointer and model list, is then forwarded to the next client.

A key aspect of ZORRO's design is that the employed zero-knowledge proofs (ZKPs) are both sound and complete, meaning that a client cannot falsely attest to the correct execution of the protocol and still produce a valid proof. Consequently, any failed ZKP can be unambiguously traced back to the responsible client, allowing both the server and other clients to identify the malicious actor with certainty. This forces adversarial clients to strictly follow the ZORRO protocol and execute the defense procedure on their own (potentially poisoned) model. This enforcement mechanism introduces a strategic dilemma for the adversary. On one hand, deviating from the protocol or manipulating the ZKP will result in the client being exposed and excluded. On the other hand, correctly executing the defense mechanism on a poisoned model will likely lead to its detection and removal. Since a malicious client's goal is to have their poisoned model selected as the best model (BM), they are incentivized to follow the protocol and ensure the generation of a valid proof, undermining the effectiveness of the attack itself.

4.2 Defense Flow

The individual steps each client executes once the system is initialized are outlined below and illustrated in Fig. 2.

Step 1 – Receive Model Checkpoints and Verification Data. At time step t , client C_i receives from the previous client C_{i-1} a list of k model checkpoints¹ M_{i-k}, \dots, M_{i-1} , their corresponding updates,

¹Each model M_j consists of the respective head H_j and tail T_j . Further, for notational simplicity, we denote the top- k best models received by client C_i as $[M_{i-k}, \dots, M_{i-1}]$, which assumes that all k clients that precede C_i are benign. This is not always the

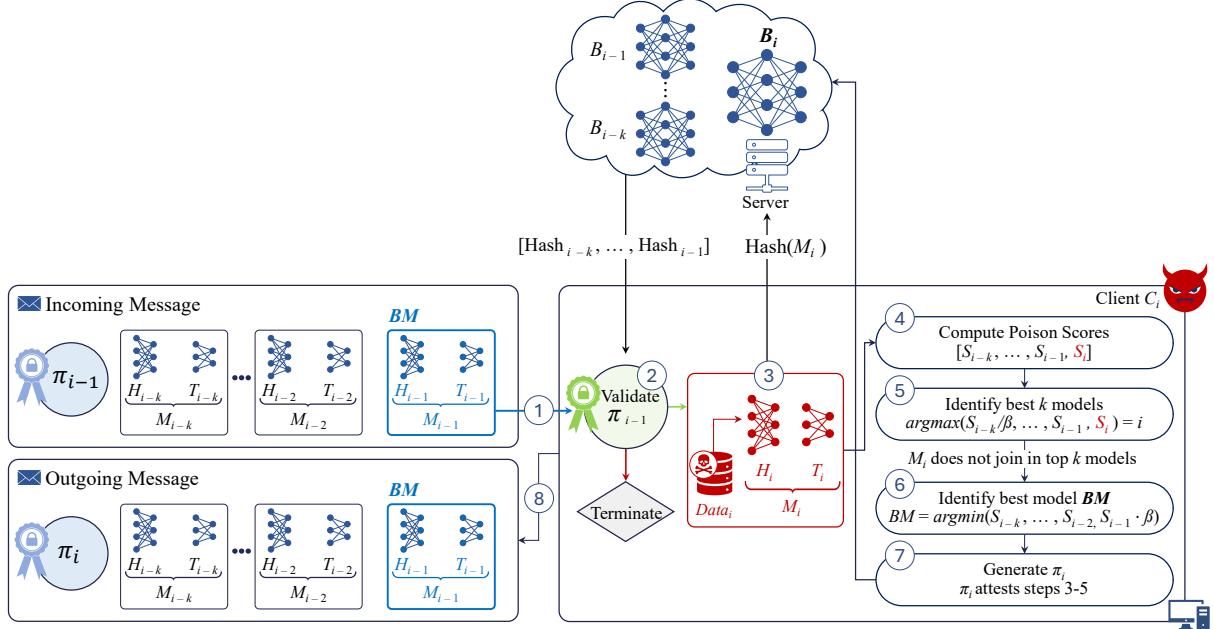


Figure 2: End-to-End Workflow of ZORRO at Each Client in Split Learning.

a pointer BM identifying the recommended model for continuation, and a zero-knowledge proof π_{i-1} attesting to the correct execution of the defense mechanism. Simultaneously, client C_i obtains from the server the corresponding model hashes $\text{Hash}_{i-k}, \dots, \text{Hash}_{i-1}$ allow client C_i verifying the integrity of the received models and updates.

Step 2 – Validate Proof and Integrity of Received Models. Client C_i verifies the received zero-knowledge proof π_{i-1} to confirm that the previous client correctly executed the defense mechanism. Additionally, it compares the model and update hashes against the values provided by the server to ensure the authenticity and integrity of the forwarded models (step ② in Fig. 2).

Step 3 – Conduct Local Training. Based on the pointer BM , client C_i instructs the server to use the corresponding backbone B_i from the server’s stored list B_{i-k}, \dots, B_{i-1} . C_i then trains a new model M_i using its private dataset and the referenced model checkpoint, following the standard U-shaped SL protocol (step ③ in Fig. 2).

Step 4 – Poison Risk Scoring. A poison-scoring function is applied to all $k+1$ models to determine a risk score that each model was trained on poisoned data. The scoring mechanism is based on the observation that backdoor attacks aim to change the predictions for triggered inputs toward the backdoor target label that differs from the correct label, thus being in contradiction to the model’s benign behavior. This behavioral conflict introduces detectable artifacts in the frequency representation of the model updates. The scoring function captures the magnitude of these artifacts and assigns a corresponding risk score to each model (step ④ in Fig. 2).

Step 5 – Prune Model List. Using the computed poison-risk scores, one model is selected for removal from the list that C_i is going to send to the next client C_{i+1} . This step serves a dual purpose.

It eliminates potentially poisoned models that could compromise subsequent clients, particularly in the case of a malicious C_i . Further, when no poisoned model is detected, it ensures training progress by removing the oldest model. To encourage the removal of the oldest model in benign scenarios, its score is adjusted by dividing it by a security parameter $0 < \beta \leq 1$, increasing its relative score (step ⑤ in Fig. 2).

Step 6 – Update Best Model Pointer BM . Following model pruning, the pointer BM is updated to reference the model with the lowest poison risk score, which will be used by the next client C_{i+1} as a checkpoint for the local training. To encourage the use of the most recent model, when it is not poisoned, and to support continued learning progress, the score of the newest model is scaled by β , reducing its score when it is likely benign.

Step 7 – Correctness Verification. Client C_i then interacts with the server and the next client C_{i+1} in parallel to provide an interactive ZKP π_i that attests to the correct calculation of poison scores (step 4), the worst model (step 5) and the other top- k models (step 5), as well as BM (Step 6). Additionally, client C_i submits the hash Hash_i of its newly trained model M_i as well as the hash of the update $M_i - M_{i-1}$ to the server. The proof π_i also verifies that the top- k models were derived using only models committed to via the server-provided hashes $\text{Hash}_{i-k}, \dots, \text{Hash}_{i-1}$ and that the update is actually the difference between M_i and M_{i-1} (step ⑦ in Fig. 2).

Step 8 – Forward Models and Metadata. In the final step, client C_i sends the updated list of the k best models and their corresponding updates, as determined in Step 5, along with the updated pointer BM , to the next client C_{i+1} (step ⑧ in Fig. 2).

case, but malicious clients will be detected and replaced by benign clients or dropped.

Algorithm 1 Poisoning Detection in \mathcal{Z} ORRO at Step i

```

1: Input: Model list  $\mathbb{M}_{i-1} = [M_{i-k}, \dots, M_{i-1}]$ 
2: Update list  $\mathbb{U}_{i-1} = [U_{i-k}, \dots, U_{i-1}]$ 
3: New model  $M_i$ , new update  $U_i$ , security parameter  $\beta \in (0, 1]$ 
4: Output: Updated model list  $\mathbb{M}_i$ , update list  $\mathbb{U}_i$ , best model index BM
5: function DETECTPOISONED( $\mathbb{M}_{i-1}, \mathbb{U}_{i-1}, M_i, U_i, \beta$ )
6:    $\mathbb{M}_{\text{tmp}} \leftarrow \mathbb{M}_{i-1} \cup \{M_i\}$ 
7:    $\mathbb{U}_{\text{tmp}} \leftarrow \mathbb{U}_{i-1} \cup \{U_i\}$ 
8:    $S \leftarrow []$                                 Initialize poison score list
9:   for each  $U_t \in \mathbb{U}_{\text{tmp}}$  do
10:     $L_t \leftarrow \text{LowFreq}(\text{DCT}(U_t))$ 
11:     $s_t \leftarrow \text{Taxicab}(L_t)$ 
12:     $S \leftarrow \text{append}(S, s_t)$ 
13:   end for
14:    $S[0] \leftarrow S[0]/\beta$ 
15:    $S[k] \leftarrow S[k] \cdot \beta$ 
16:    $j \leftarrow \text{argmax}(S)$ 
17:    $M_i \leftarrow [M_t \in \mathbb{M}_{\text{tmp}} \mid t \neq j]$ 
18:    $\mathbb{U}_i \leftarrow [U_t \in \mathbb{U}_{\text{tmp}} \mid t \neq j]$ 
19:   BM  $\leftarrow \text{argmin}(S \setminus \{S[j]\})$ 
20:   return  $\mathbb{M}_i, \mathbb{U}_i, \text{BM}$ 
21: end function

```

4.3 Poisoning Detection

To detect poisoned training contributions, \mathcal{Z} ORRO analyzes the updates to the head and tail model partitions in the frequency domain. The rationale is that, during the early stages of training, the introduction of new behaviors, such as those caused by backdoor attacks, leads to significant shifts in the *low-frequency components* of the model's spectral representation [38, 39, 57]. Since the backdoor target label, by definition, differs from the benign label, the backdoor behavior contradicts previously learned benign behavior. Thus, the model must revert from benign to backdoor behavior, resulting in further detectable modifications in these low-frequency components. This means that the presence of new and conflicting behaviors introduced during local training can be effectively captured by measuring deviations in the model update's low-frequency spectrum.

To compute the poison risk scores, \mathcal{Z} ORRO first calculates the frequency-domain representation of each of the $k+1$ model updates. It then isolates the relevant low-frequency components (see App. A) and applies the *Taxicab norm* (also known as ℓ_1 -norm) to quantify deviations. This choice of norm ensures robustness against outliers and emphasizes aggregate discrepancies across the spectrum.

To ensure training progress in the absence of poisoned models, \mathcal{Z} ORRO adjusts the scores of specific models using a tunable *security parameter* $\beta \in (0, 1]$. Specifically, the score of the *oldest model* is divided by β , increasing its relative likelihood of removal, while the score of the *most recent model*, which was trained by the current client, is scaled by β to slightly incentivize the selection of the most up-to-date benign model as the checkpoint for subsequent rounds.

4.4 Initialization

The poisoning detection in \mathcal{Z} ORRO relies on comparing poison risk scores across different models and selecting the one with the lowest score to update *BM*. This mechanism requires initialization with at least one benign model to ensure a benign alternative reference is available, especially for cases where the first clients in the training queue are all malicious. Several strategies exist for this initialization.

Algorithm 2 \mathcal{Z} ORRO game

Public Inputs (\mathcal{P} & \mathcal{V}): $[\text{Hash}_{i-k}, \dots, \text{Hash}_i], [\text{Hash}_{\text{DCT}(M_{i-k})}, \dots, \text{Hash}_{\text{DCT}(M_i)}], S_{BM}, S_{WM}$ Private Inputs (\mathcal{P}): $[M_{i-k}, \dots, M_i], [DCT(M_{i-k}), \dots, DCT(M_i)]$	Verifier \mathcal{V} (Client C_{i+1} & Server)
ZKP $\pi \leftarrow \text{ZK Circuit}$: begin circuit for M_t in $[M_{i-k}, \dots, M_i]$: <i>assert</i> $\text{Hash}(M_t) = \text{Hash}_t$ <i>compute</i> $S_t = S(M_t)$ <i>Receive r</i> $\xleftarrow{\hspace{1cm}} r$ <i>assert</i> $\text{Hash}(DCT(M_t)) = \text{Hash}_{DCT(M_t)}$ <i>assert</i> $DCT(M_t, r) = DCT(M_t) \cdot r$ end for <i>assert</i> $S_{WM} = \max(S_{i-k}/\beta, \dots, S_i)$ <i>assert</i> $S_{BM} = \min(S_{i-k}, \dots, S_i \cdot \beta)$ end circuit <i>Send π</i> $\xrightarrow{\hspace{1cm}} \pi$ <i>Receive π</i> <i>Verify proof π</i> Output: Accept / Reject	
<i>Note:</i> \mathcal{P} only wins game if π is accepted by \mathcal{V} . π is only accepted if all operations in ZK Circuit are computed soundly and with valid inputs.	

The most convenient approach is to use a (small) clean dataset or a pretrained model, if available. We introduce a secure initialization phase in which we temporarily increase the size of the reference list from k to N . Each client independently trains its local model for one round, all starting from the same random model provided by the server. The resulting models are added to the reference list, and before the first effective round of Split Learning, the last client applies \mathcal{Z} ORRO to select the *BM* from this expanded reference set. As shown in Figure 4, this initialization process is agnostic to client ordering, consistently identifies a benign starting model, and achieves performance comparable to using a pre-trained model. As such, the secure initialization phase can be seamlessly interchanged with a pre-trained model when one is available, making both approaches functionally equivalent in terms of effectiveness and resulting model integrity.

4.5 Cryptographic Attestation of \mathcal{Z} ORRO

Due to the client-based nature of \mathcal{Z} ORRO, malicious clients may stray from the protocol in an attempt to adversely affect the training process without being detected. Although \mathcal{Z} ORRO introduces an approach to dropping poisoned models based on their poison risk score, if the integrity of the poison risk score calculations is not ensured, then malicious clients can still collude and remain undetected. In Alg. 2, we outline \mathcal{Z} ORRO's approach to probabilistically guarantee faithful execution of the proposed client-side defense scheme while ensuring that client privacy is still maintained through the use of interactive ZKPs.

In ZK terminology, a "circuit" is defined as any arbitrary computation that a prover aims to attest the computational integrity and compliance of. Alg. 2 defines a circuit that is evaluated by a prover \mathcal{P} , resulting in a ZKP π . This circuit is evaluated on public and private inputs, in which public inputs are known to both the verifier \mathcal{V} and prover \mathcal{P} , and the private inputs are only known to the prover \mathcal{P} . The public inputs are the committed values of the top- k models and client C_i 's model, alongside the committed values of these models after undergoing the discrete cosine transform (DCT). Poison risk scores corresponding to the best (smallest poison risk) and worst (highest poison risk) S_{BM} and S_{WM} , respectively, are also public inputs. The only private inputs are the unmasked,

plaintext model parameters of the top- k models and client C_i 's model alongside their frequency representations. In the proposed setting, the prover \mathcal{P} is a client C_i who has already trained their model M_i on their private dataset D_i on the backbone B_i hosted on the server. The goal of the prover is to concurrently convince client C_{i+1} and the server, who each act as a verifier \mathcal{V} , that they are faithfully executing the required computations for the ZORRO defense scheme. The prover shares π with both the following client C_{i+1} and the server to ensure that if client C_{i+1} is malicious, then it cannot falsely reject the proof.

The initial loop iterates through each model to ensure correct analysis. For each model M_t , the first assertion $\text{Hash}(M_t) = \text{Hash}_t$ guarantees that the prover is indeed using the expected model corresponding to a previously published, publicly known hash. This prevents the prover from substituting a different, potentially benign-looking model just to generate a valid proof. Subsequently, the poison risk score S_t is computed based on the authenticated model M_t . One of the core operations that enables an accurate poison risk score is the DCT, which is a costly operation to verify in ZK. To address this, we take a modified approach using Freivald's algorithm [17], which utilizes verifier-generated randomness r to verify the correct calculation of the DCT and compares the committed result to a publicly known hash of the result to convince the verifier of the computational integrity. The circuit takes in the DCT of each model as a private input to allow for further consistency checks. This will be discussed in detail in section 4.7. This sequence tightly binds the original model and its calculated scores to the publicly available commitments, taken in as public inputs to the ZK circuit, to ensure faithful execution of the operations on each model.

Following the per-model checks, the circuit performs crucial aggregate verifications using the final two assertions. These checks concern the claimed best model score S_{BM} and the worst model score S_{WM} within the relevant window (models $i - k$ through i , including the prover C_i 's own model M_i). These assertions ensure that the prover correctly identifies the worst model, in terms of poison risk score, and proves that it is not included in the top- k models that it sends to the next client. Successfully passing these checks demonstrates that the prover not only computed the poison risk scores correctly, but also accurately modified the top- k models to appropriately reflect the poison risk scores. If all assertions hold, the entire computation is soundly encoded into a proof π_i that will be validated by the verifiers, the next client C_{i+1} and the server. When the verifiers receive and successfully verify π_i , they gain high confidence that client C_i followed the ZORRO protocol faithfully without releasing any sensitive information. A rejection of π_i signals a deviation from the ZORRO protocol and potential malicious intent. This ZKP mechanism forms the trust anchor for the client-based ZORRO defense strategy.

4.6 Security Parameters

The poisoning detection mechanism of ZORRO relies on two parameters, the *security parameter* β and the *queue length* k . In the following, we elaborate on these parameters and their impact on the performance of ZORRO. An ablation study evaluating different parameter configurations is provided in App. D to complement the theoretical discussion provided here.

Security Parameter β . The security parameter $\beta \in (0, 1]$ performs the trade-off between ZORRO's robustness against poisoned models and its ability to maintain a continuous training progress. The β parameter is used to scale the risk score of the newest model, by multiplication, and the oldest model, by division. A smaller β value reduces the used value of the most recent model's risk score, increasing the likelihood of it being selected for the next training iteration. Simultaneously, it increases the relative score of the oldest model, making it more likely to be pruned from the queue. This mechanism helps prioritize newer, potentially benign updates and avoid discarding recent training contributions unnecessarily. However, setting β too low may cause the scheme to overly favor the most recent model, even in the presence of subtle poisoning, weakening the system's security. We choose $\beta = 0.7$ to ensure a practical trade-off between training efficiency and defense robustness.

Queue Length k . The queue length k determines the number of past model checkpoints (and updates) that are retained and evaluated at each client during the training process. A larger k increases the defense's robustness by providing a broader context for detecting poisoned updates and reducing the risk of coordinated attacks from malicious clients. On the other side, a lower number of shared client models reduces the privacy risk as fewer models are shared with each client for lower values, reducing the risk of successful privacy attacks. Further, smaller values reduce the communication and computational overhead, as each client must evaluate more models. In our experiments in Sect. 5, we choose $k = 3$.

4.7 Implementation

ZKP Circuit Implementation. ZORRO utilizes Zero-Knowledge Proofs (ZKPs) based on Vector Oblivious Linear Evaluation (VOLE), leveraging its advantageous scalability compared to alternative ZKP schemes, a key finding highlighted in [42]. As discussed in Sect. 2.2, many other ZKP constructions involve computationally demanding setup phases, which might necessitate significant communication overhead or reliance on a trusted third party. VOLE-based ZK strikes an effective balance between computational efficiency during proof generation and verification and scalability, making it well-suited for our application.

The ZK circuit, detailed in Alg. 2, is realized using the emp-zk library [52], a state-of-the-art framework specifically designed for VOLE-based ZK protocols. emp-zk is built upon the emp-tool toolkit [51], a widely adopted library for efficient multi-party computation. To optimize performance, we employ a customized version of the emp-tool backend, specifically enhanced to support parallel proof generation and further multithreading capabilities, reducing latency of proof generation and verification.

Our ZK circuit design accommodates computations involving both arithmetic and Boolean values. Arithmetic circuits are employed for operations like addition and multiplication, primarily used when computing the poison risk scores (S_t). Boolean circuits are utilized for operations such as cryptographic hashing. The seamless integration of both circuit types is facilitated by the efficient conversion techniques presented in [54], enabling a fine-grained optimization strategy for complex computations. Specifically, commitments within the ZK circuit are computed and verified using a custom Boolean SHA-256 implementation. This implementation

efficiently processes model parameters, represented as serialized bitstreams, to produce a 256-bit cryptographic hash digest.

Instead of a monolithic design, we adopt a modular approach to building the ZK circuits in \mathcal{ZORRO} . This design choice promotes extensibility, enabling the reuse of core components (e.g., DCT, SHA-256, matrix multiplication) across different client-side defense schemes. Developers can also integrate new defense-specific modules, such as ℓ_2 -norm or thresholding, within the same framework.

Speeding Up DCT Verification. The core operation influencing the poison risk score calculation is the Discrete Cosine Transform (DCT) applied to model parameters M_t , represented as a matrix of size $\lceil \sqrt{|H_t + T_t|} \rceil \times \lceil \sqrt{|H_t + T_t|} \rceil$, where H_t is the number of head parameters and T_t is the number of tail parameters. Computing this matrix directly within a ZK circuit is highly inefficient due to the required cosine functions and the large number of multiplications and additions over potentially very large matrices. The conversions between arithmetic and Boolean domains for cosine approximations would introduce significant overhead.

To mitigate this complexity, we avoid computing the DCT directly within the ZK circuit. Instead, we leverage a technique inspired by Freivald's algorithm, a classical method for probabilistically verifying matrix multiplication [17]. Through our experiments, we found that this approach was around 2 orders of magnitude faster than the naive approach of computing the DCT within the ZK circuit. Freivald's algorithm allows checking if $C = AB$ for matrices A, B, C by choosing a random binary vector r and verifying if $Cr = A(Br)$. This check is significantly faster than performing the full matrix multiplication AB . If $C = AB$, the equality $Cr = A(Br)$ always holds. If $C \neq AB$, the equality holds only with a small probability. This process can be repeated many times to ensure high probabilistic guarantees.

$$\text{Check: } Cr \stackrel{?}{=} A(Br) \quad \text{for random } r \quad (1)$$

This probabilistic verification fits naturally within interactive ZK protocols, where \mathcal{V} can provide the random challenge vector r . \mathcal{P} then performs the computations involving r inside the ZK circuit to convince the Verifier of the original matrix relationship's validity without revealing the private matrices (like M_t in our case).

Applying this idea to our DCT scenario, the 2D DCT can be expressed in matrix form as $D_t = C_N M_t (C_N)^T$, where C_N represents a DCT transformation matrix, respectively, whose entries depend on the cosine terms from the standard definition of the DCT. This is how the DCT is efficiently computed on the opencv C++ package [35]. The matrix C_N is known, fixed and depends only on the dimension $\lceil \sqrt{|H_t + T_t|} \rceil$. It can be pre-computed offline and treated as a constant within the ZK circuit. \mathcal{P} 's goal is not to explicitly re-compute $DCT(M_t)$ inside the circuit but rather to use a randomized check to prove that the privately held $DCT(M_t)$ is consistent with the privately held M_t , thus ensuring the integrity of the inputs for the calculation of the poison risk score S_t .

We achieve this using the randomized check provided by \mathcal{V} 's random vector r (of dimension $T_t \times 1$). Instead of checking the full matrix equality $DCT(M_t) = C_N \cdot M_t \cdot (C_N)^T$, we check the related equality probabilistically by multiplying by r : $DCT(M_t) \cdot r = (C_N \cdot M_t \cdot (C_N)^T) \cdot r$. The right-hand side can be computed efficiently within the arithmetic ZK circuit using matrix-vector multiplications

Table 1: Overview of default experiment parameters.

Hyperparameter	Value
Poison Data Rate (PDR)	75%
Number of Clients	10
Poison Model Rate (PMR)	20%
IID Degree	0.8
Dataset	CIFAR-10
DNN Architecture	ResNet18
β	0.7
Queue Length k	3
Initialization	Pretrained

by exploiting associativity:

- (1) Compute $v_1 = (C_N)^T \cdot r$. (Size $\lceil \sqrt{|H_t + T_t|} \rceil \times 1$)
- (2) Compute $v_2 = M_t \cdot v_1$. (Size $\lceil \sqrt{|H_t + T_t|} \rceil \times 1$)
- (3) Compute $Z = C_N \cdot v_2$. (Size $\lceil \sqrt{|H_t + T_t|} \rceil \times 1$)

This computation, denoted as $DCT(M_t, r)$ in Alg. 2, involves only matrix-vector products using \mathcal{P} 's private matrix M_t , the constant cosine transformation matrix C_N , and \mathcal{V} 's randomness r . All operations are arithmetic (multiplications and additions), which are efficiently handled by emp-zk's arithmetic circuit backend. The final step within the ZK circuit is to assert that Z matches the corresponding private input times the random vector r , $DCT(M_t) \cdot r$. This ensures, with high probability, that \mathcal{P} is correctly computing the poison risk score S_t for model M_t .

5 Experimental Evaluation

In the following, we perform an extensive evaluation of \mathcal{ZORRO} 's robustness using various settings and also measure the runtime overhead of the ZKP-protocol. Further, in App. D we provide an ablation study.

5.1 Experimental Metrics

In our experiments, we evaluate the effectiveness of \mathcal{ZORRO} on the model using four metrics: *Backdoor Accuracy (BA)*, *Main Task Accuracy (MA)*, *Poisoned Removal Rate (PRR)*, and *Better Benign Rate (BBR)*. These metrics capture both security-related and utility aspects of the training process.

Backdoor Accuracy (BA) measures the effectiveness of the backdoor embedded in a model. It is computed as the model's accuracy on a backdoored test set, i.e., a set of inputs containing the trigger pattern. The BA indicates the proportion of triggered samples classified as the attacker's target label. The adversary \mathcal{A} seeks to maximize BA, whereas an effective defense aims to minimize it (see R1 in Sect. 3.1).

Main Task Accuracy (MA) reflects the model's utility on the intended benign task. It is evaluated as standard accuracy on an unaltered, benign test set, following non-security-focused deep learning benchmarks. Both the adversary \mathcal{A} and the defense seek to preserve a high MA, the \mathcal{A} to remain undetected, and the defense to ensure practical applicability (see requirement R2 in Sect. 3.1).

Poisoned Removal Rate (PRR) quantifies the defense's effectiveness in eliminating poisoned models. It is defined as the fraction of time steps in which at least one poisoned model was present

Table 2: Effectiveness of \mathcal{Z} ORRO for different datasets in terms of Backdoor Accuracy (BA), Main Task Accuracy (MA), Poisoned Removal Rate (PRR), and Better Benign Rate (BBR).

Dataset	Defense	BA	MA	PRR	BBR
Cifar10	Benign	3.71%	74.63%	-	-
	No Defense	93.29%	70.56%	-	-
	\mathcal{Z} ORRO	4.45%	73.02%	86.10%	8.02%
	Gold Standard	3.30%	73.40%	100.00%	0.00%
MNIST	Benign	0.23%	99.29%	-	-
	No Defense	100.00%	67.72%	-	-
	\mathcal{Z} ORRO	36.99%	99.25%	100.00%	3.79%
	Gold Standard	0.00%	99.24%	100.00%	0.00%
FMNIST	Benign	1.19%	86.65%	-	-
	No Defense	100.00%	84.67%	-	-
	\mathcal{Z} ORRO	1.38%	86.22%	58.81%	6.57%
	Gold Standard	1.46%	86.40%	100.00%	0.00%
Forest Cover Type	Benign	0.00%	99.79%	-	-
	No Defense	87.40%	45.37%	-	-
	\mathcal{Z} ORRO	0.03%	99.77%	94.99%	14.36%
	Gold Standard	0.00%	99.83%	100.00%	0.00%
CIFAR-100	Benign	0.24%	57.11%	-	-
	No Defense	99.98%	47.04%	-	-
	\mathcal{Z} ORRO	0.30%	55.86%	100.00%	15.79%
	Gold Standard	0.25%	56.27%	100.00%	0.00%
Tiny ImageNet	Benign	0.09%	66.36%	-	-
	No Defense	99.96%	58.50%	-	-
	\mathcal{Z} ORRO	0.07%	66.90%	94.10%	10.19%
	Gold Standard	0.10%	66.74%	100.00%	0.00%

in the queue and the model selected for removal was indeed poisoned. This metric only considers scenarios where poisoned models are present and removable. To prevent any backdoor from persisting, an effective defense must strive to maximize PRR by promptly removing newly introduced poisoned models.

Better Benign Rate (BBR) captures the frequency with which the defense fails to select the most recent benign model despite its availability. A high BBR suggests that the defense unnecessarily reverts to older models, potentially hindering convergence and training progress. Therefore, a robust defense should aim to minimize BBR, promoting continuous learning while ensuring model integrity.

5.2 Experimental Setup

5.2.1 Training Setup. In the following, we extensively investigate various settings for the training process. The default parameters of the experiment scenario are shown in Tab. 1, which are always used unless other parameters are described. Notably, in our default setting, we use 10 clients in total and a PMR of 20%, resulting in 2 malicious clients. However, when increasing the client numbers in Sect. 5.4 using the same PMR, the absolute number of malicious clients increases accordingly. The positions of the malicious clients are randomly chosen. For the evaluation, we employ five image benchmark datasets (CIFAR-10, MNIST, FMNIST, CIFAR-100, Tiny ImageNet) that are typically used to assess the security of distributed learning [25, 39, 46] and one tabular dataset, the Forest Cover Type dataset. We describe them in App. B.

DNN Architectures. While using ResNet-18 as the default architecture, we extend the evaluation to a wide range of other architectures to show \mathcal{Z} ORRO’s scalability and generalizability. An overview of the leveraged models and the number of their parameters is shown in Tab. 4.

Table 3: Effectiveness of \mathcal{Z} ORRO for different model architectures in terms of Backdoor Accuracy (BA), Main Task Accuracy (MA), Poisoned Removal Rate (PRR), and Better Benign Rate (BBR).

Model	Defense	BA	MA	PRR	BBR
GoogLeNet [45]	Benign	4.20%	67.14%	-	-
	No Defense	98.61%	59.85%	-	-
	Gold Standard	3.85%	65.65%	100.00%	0.00%
	\mathcal{Z} ORRO	5.23%	65.35%	80.09%	4.12%
MicronNet [55]	Benign	4.23%	68.85%	-	-
	No Defense	99.51%	53.55%	-	-
	Gold Standard	5.78%	67.97%	100.00%	0.00%
	\mathcal{Z} ORRO	5.64%	68.01%	55.48%	3.12%
ResNet-18 [22]	Benign	3.71%	74.63%	-	-
	No Defense	93.29%	70.56%	-	-
	Gold Standard	3.30%	73.40%	100.00%	0.00%
	\mathcal{Z} ORRO	4.45%	73.02%	86.10%	8.02%
ResNet34 [22]	Benign	2.18%	69.74%	-	-
	No Defense	100.00%	66.24%	-	-
	Gold Standard	2.15%	69.20%	100.00%	0.00%
	\mathcal{Z} ORRO	2.69%	69.17%	85.98%	3.12%
VGG11 [43]	Benign	3.00%	69.55%	-	-
	No Defense	99.98%	69.55%	-	-
	Gold Standard	4.97%	69.07%	100.00%	0.00%
	\mathcal{Z} ORRO	4.82%	67.20%	55.48%	2.67%
WideResNet52 [62]	Benign	2.36%	80.84%	-	-
	No Defense	99.97%	74.95%	-	-
	Gold Standard	2.33%	80.30%	100.00%	0.00%
	\mathcal{Z} ORRO	2.15%	79.93%	100.00%	18.25%

Initialization. For our experiments, we use a pre-trained model for our default setting, obtained via 3 rounds of benign training in the respective setup. In Sect. 5.4 we empirically analyze the impact of our initialization described in Sect. 4.4 on \mathcal{Z} ORRO’s performance.

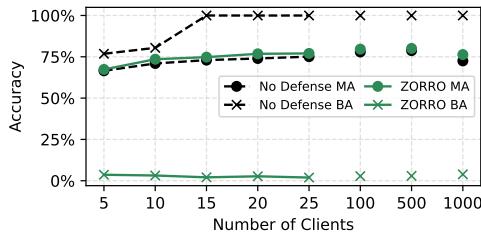
5.2.2 Computational Setup. The ML experiments are conducted on two servers. One server is equipped with an AMD EPYC 9954, 576 GB of main memory, and 4 NVIDIA A6000. The second server, which was used for the experiments for different models (Tab. 3) and datasets (Tab. 2), as well as the ZKP runtime evaluations, was equipped with an AMD EPYC 7742, 4 NVIDIA Quadro RTX 8000 and 1 TB main memory. For measuring the main memory consumption (see Sect. 5.8) of the SL training, we used a Raspberry PI 4B with 4GB main memory.

All experiments were conducted using Pytorch [36]. While we ensured the reproducibility of the experiments by seeding the random generators, different library versions on both servers caused slightly varying results as the random generators were initialized differently. All experiments were repeated 3 times with different seeds, and the average results are shown. In App. C, we evaluate and discuss the variance in the results.

5.2.3 Baseline Defenses. In our evaluation, we use two baselines, No Defense as well as a defense we denote as Gold Standard. The second one is a hypothetical defense that knows the benign and poisoned models, allowing it to always exclude poisoned models and use the latest benign model for future training. While in practice, this defense is not possible, it shows the optimal performance that \mathcal{Z} ORRO should achieve.

Table 4: Overview of employed Deep Neural Network (DNN) architectures.

Model	Dataset	Number of Parameters		
		Head	Backbone	Tail
VGG11	CIFAR-10	1 920	13 418 880	5 130
ResNet34	CIFAR-10	9 536	21 275 136	5 130
GoogLeNet	CIFAR-10	9 536	5 590 368	10 250
WideResNet52	CIFAR-10	9 536	66 824 704	204 900
ResNet-18	CIFAR-10	9 536	11 166 976	5 130
ResNet-18	MNIST	3 264	11 166 976	5 130
ResNet-18	FMNIST	3 264	11 166 976	5 130
MicronNet	CIFAR-10	824	411 192	3 010
WideResNet52	CIFAR-100	9 536	66 824 704	204 900
WideResNet52	Tiny ImageNet	9 536	66 824 704	409 800
SimpleMLP	Forest Cover Type	14 848	570 336	4 357

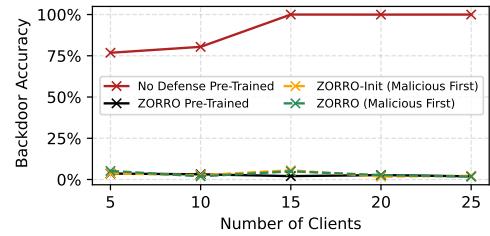
**Figure 3: \mathcal{Z} ORRO’s effectiveness for different client numbers.**

5.3 High-Level Results

Tab. 2 shows \mathcal{Z} ORRO’s effectiveness for different datasets². As the table shows \mathcal{Z} ORRO effectively mitigates the attack. \mathcal{Z} ORRO effectively mitigates the attack for all datasets. For CIFAR-10 and FFMNIST, it reduces the BA to a similar level as the Gold Standard. Only for MNIST the BA is in average 36.99%, although \mathcal{Z} ORRO always excludes poisoned models, guaranteeing that the model is backdoor-free. The relatively high BA (36.99%) observed on MNIST can be attributed to an artifact rather than a failure of the defense. Since \mathcal{Z} ORRO excluded the poisoned models in every repetition, and only one out of three runs exhibited high BA, it is likely that the benign model was distracted by the trigger. In such cases, the model may consistently predict a specific label in the presence of the trigger, which, if it by chance is the backdoor target label, results in a high BA, even though the model is benign. We provide a detailed investigation of this phenomenon in Appendix F.

Tab. 3 shows the effectiveness of \mathcal{Z} ORRO across different model architectures. As the table shows, \mathcal{Z} ORRO consistently mitigates the backdoor attacks, reducing BA to levels comparable with the Gold Standard while maintaining high MA. Notably, although the PRR for MicronNet is only 55.48%, the BA remains low at 5.64%, showing that the poisoned models were either neutralized or ineffective. Similarly, for WideResNet52, \mathcal{Z} ORRO achieves perfect PRR (100.00%) and the lowest BA (2.15%) among all defenses, showcasing its robustness even for large-scale models. Overall, \mathcal{Z} ORRO is effective across a diverse range of architectures.

²Notably, CIFAR-100 and Tiny ImageNet deviate from the setting described in Tab. 1, as discussed in App. B.

**Figure 4: BA for different client counts and initializations, including scenarios where malicious clients train first.**

5.4 Data Settings

Fig. 3 visualizes the effectiveness of \mathcal{Z} ORRO across different numbers of clients. As the figure shows, while BA under no defense remains consistently high, \mathcal{Z} ORRO maintains a low BA across all settings, even in cases with a high client count, such as 1000 clients. \mathcal{Z} ORRO remains effective, maintaining the MA stable, except for a small drop in the case of 1000 clients in both scenarios.

In Fig. 4 we compare \mathcal{Z} ORRO in scenarios where training starts with malicious clients, both with and without benign pre-training. To address the more challenging case without pre-training and adversarial client order, we propose a novel initialization method (see Sect. 4.4). As Fig. 4 shows, while \mathcal{Z} ORRO alone can effectively remove malicious clients and maintain a low BA, the novel initialization is critical for handling edge cases as it is independent of client order and resilient to early-stage poisoning. In all experiments, it consistently selected a benign client as the entry point.

Tab. 5 shows the performance of \mathcal{Z} ORRO under varying degrees of IID among clients. Across all settings, \mathcal{Z} ORRO significantly reduces BA compared to No Defense and closely tracks the performance of the Gold Standard. While MA increases with higher IID degrees for all defense schemes, most likely due to the training data becoming more representative, \mathcal{Z} ORRO maintains consistently low BA even in highly non-IID scenarios, demonstrating its robustness in settings with heterogeneous data.

Fig. 5 shows the performance of \mathcal{Z} ORRO for different PMRs, including PMR= 0%. As the figure shows, \mathcal{Z} ORRO is also capable of achieving a decent accuracy in the absence of an attack, as it does not enforce discarding models if no attack is detected. The effectiveness for varying PMRs will be further discussed in Sect. 5.5.

5.5 Attack Settings

Fig. 5 shows the effectiveness of \mathcal{Z} ORRO under varying poisoning model ratios (PMR). As expected, BA under no defense remains

Table 5: Effectiveness of \mathcal{Z} ORRO for different IID degrees in terms of Backdoor Accuracy (BA) and Main Task Accuracy (MA).

IID	No Defense		Gold Standard		\mathcal{Z} ORRO	
	MA	BA	MA	BA	MA	BA
0.4	17.87%	58.43%	36.24%	1.01%	36.21%	2.78%
0.6	55.70%	66.22%	65.52%	5.81%	65.21%	4.30%
0.8	70.95%	80.41%	73.75%	2.82%	73.51%	3.16%
1.0	73.24%	88.80%	74.57%	5.33%	74.43%	4.61%

Table 6: Effectiveness of ZORRO against various adaptive attack strategies, including Differential Privacy (DP)-based disguise, loss-constrained frequency optimization, varying poisoned data rates (PDRs) [5], learning rate manipulations and Out Of Distribution samples (OOD) [50].

Defense	Attack	MA	BA	PRR	BBR
ZORRO	Default Attack	73.75%	2.82%	100.00%	0.00%
	DP Attack ($\epsilon=3.0, \delta=1.4$)	73.02%	6.20%	100.00%	7.29%
	Frequency Attack	73.51%	3.16%	94.66%	3.01%
	Low PDR (25%) [5]	73.51%	3.16%	88.17%	3.01%
	Low PDR (50%) [5]	73.51%	3.16%	94.17%	3.01%
	Default Attack	73.51%	3.16%	94.66%	3.01%
	High PDR (100%)	73.51%	3.16%	98.87%	3.01%
	lr=0.0001	73.50%	3.35%	84.96%	3.79%
	lr=0.01	73.51%	3.16%	99.71%	3.01%
	OOD [50]	73.29%	1.27%	90.55%	4.80%

consistently high across all PMR levels, while ZORRO maintains a low BA close to the Gold Standard baseline even as the number of malicious clients increases. Since with increasing PMR less benign clients are involved and, as such, also less benign data contribute, the MA decreases for all settings.

Tab. 6 evaluates the robustness of ZORRO against various adaptive attack strategies. These include attempts to disguise backdoors using Differential Privacy, targeted manipulation of the frequency spectrum, and learning rate variations. As the results show, ZORRO maintains strong defense performance across all scenarios, consistently keeping BA below 4% and MA around 73.5%. Even under aggressive conditions such as full poisoning (PDR 100%) or with learning rate manipulations, ZORRO achieves high PRR and low BBR. Lastly, we adapted existing Federated Learning attacks into the Split Learning paradigm. Specifically, we implemented the A-Little-Is-Enough attack [5] using reduced PDRs, and enhanced the Out-of-Distribution (OOD) attack with Projected Gradient Descent [50]. More details are available in App. H.

5.6 Comparison with other Defenses

Tab. 7 compares ZORRO against various baseline defenses. Particularly, we use Differential Privacy that clips the models at 1.4 and adds Gaussian noise with a standard deviation of 0.01. Further, we adapted SafeSplit [39] and calculated the pairwise distances between the frequencies of the models in the current queue. In addition, we used k-means to spot anomalous model updates. However, as Tab. 7 shows, except for the Gold Standard, only ZORRO achieves a strong balance between security and utility, with a low BA of 3.16%, a high MA of 73.51%, and a PRR of 94.66%. In contrast, Differential Privacy and Frequency Analysis show either weak backdoor

Table 7: Comparison of ZORRO with different baselines in terms of Backdoor Accuracy (BA), Main Task Accuracy (MA), Poisoned Removal Rate (PRR), and Better Benign Rate (BBR).

Defense	MA	BA	PRR	BBR
No Defense	70.95%	80.41%	-	-
Differential Privacy	65.70%	50.38%	-	-
Frequency Analysis	61.51%	12.22%	54.64%	86.37%
Gold Standard	73.75%	2.82%	100.00%	0.00%
k-means	52.06%	4.66%	100.00%	87.93%
ZORRO	73.51%	3.16%	94.66%	3.01%

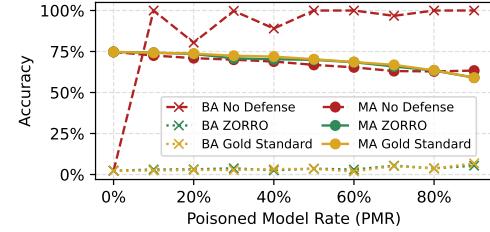


Figure 5: Effectiveness of ZORRO for different ratios of poisoned models (PMR).

mitigation or significant drops in MA. While k-means achieves high PRR and low BA, its MA is considerably reduced to 52.06%, and it suffers from a high BBR of 87.93%, indicating that it frequently disregards newer benign models. Overall, ZORRO outperforms prior defenses by combining strong robustness with practical training performance.

5.7 Runtime Evaluation

We evaluated the scalability of ZORRO in terms of runtime and communication overhead. Fig. 6 shows the prover’s runtime for different numbers of threads (1, 25, and 100), alongside the communication cost in megabytes. As shown, the runtime scales sublinearly with the number of parameters for all thread configurations, demonstrating the benefits of parallelization. Communication overhead increases linearly but remains below 200 MB even for models with up to 10^7 parameters. These results demonstrate the scalability and efficiency of ZORRO, even for large client-side model partitions.

5.8 Memory Overhead of ZORRO

We show the RAM usage for the models with the smallest (VGG11) and largest (WideResNet52) amount of head and tail parameters in Tab. 8. As can be seen, for the largest model, WideResNet52 trained on Tiny ImageNet, ZORRO only requires at most 942.77 MB of RAM to ensure robustness during SL. Although ZKPs are generally memory-intensive, our meticulous design, paired with the fact that SL reduces the amount of parameters a client processes, ensures that any client with an edge device capable of vanilla SL can ensure security and robustness with ZORRO with very little overhead. For instance, executing SL with WideResNet52 on a Raspberry Pi requires only 859.5 MB of RAM (median over 10 runs). Since the training process completes before ZKPs are executed, memory used for training becomes available again, rendering the additional memory overhead effectively negligible.

Table 8: RAM Measurements for ZORRO’s execution

Model	Dataset	Client Parameters	Max. RAM (MB)	Avg. RAM (MB)
VGG11	CIFAR-10	7050	636.91	449.97
WideResNet52	CIFAR-10	214436	823.45	553.10
WideResNet52	Tiny ImageNet	419336	942.77	666.63

6 Security Analysis

6.1 Security of the ZKP Protocol

The security of \mathcal{ZORRO} against malicious clients attempting to subvert the backdoor defense critically relies on the cryptographic guarantees of the interactive zero-knowledge proof (ZKP) protocol, as described in Sect. 4.5. We focus our analysis on the two fundamental properties of the ZKP: (1) *computational soundness*, which ensures that no adversary can produce a valid proof for a false claim, and (2) *zero-knowledge*, which guarantees that private inputs remain confidential. Together, these properties ensure the fulfillment of challenge C3 outlined in Sect. 3.1.

Let \mathcal{P} denote the prover (client C_i), and \mathcal{V} the verifiers (client C_{i+1} and the server). Let Φ denote the statement being proven, namely: \mathcal{P} has correctly executed the defense protocol (Alg. 2) using private model inputs $[M_{i-k}, \dots, M_i]$ consistent with their public commitments and discrete cosine transforms, resulting in an updated top- k model list forwarded to C_{i+1} . Let pub and $priv$ denote the public and private inputs to the ZKP protocol, respectively, and let π_i be the generated proof.

6.1.1 Preventing Proof Forgery. The central security guarantee is *computational soundness*. The Wolverine ZKP protocol [53] ensures that a polynomial-time bounded malicious prover \mathcal{P}^* cannot convince an honest verifier \mathcal{V} to accept a proof π^* for a false statement Φ , except with negligible probability ε , inherent to the VOLE-based construction. Formally, let \mathcal{L} be the language of true statements such that $(pub, priv) \in \mathcal{L}$ iff $\Phi(pub, priv)$ is true. Then:

$$\forall \text{PPT } P^*, \Pr[\mathcal{V} \text{ accepts } \pi^* \text{ from } P^*(pub) : (pub, \cdot) \notin \mathcal{L}] \leq \varepsilon \quad (2)$$

\mathcal{ZORRO} 's use of interactive ZKPs guarantees that a malicious client C_i cannot generate a valid proof if it deviates from the defense protocol. Examples of invalid behaviors include:

- *Manipulating poison risk scores*: Falsifying S_t^* to misclassify a poisoned model as benign.
- *Providing inconsistent inputs*: Using model parameters not matching their public commitments.
- *Tampering with pruning logic*: Incorrectly updating the top- k model list by retaining a high-risk model.

Any such deviation leads to a failed verification of π^* , resulting in rejection by honest verifiers with probability at least $1 - \varepsilon$. In scenarios where a malicious verifier colludes with the prover, \mathcal{ZORRO} mitigates this risk by requiring all proofs to be sent to the semi-honest server, which can terminate the protocol if a violation is detected. Thus, clients cannot successfully forge proofs without detection, ensuring the integrity of the defense.

6.1.2 Preserving Model Privacy. The *zero-knowledge* property ensures that the ZKP reveals no information about the prover's private inputs beyond the validity of the claim. Formally, for any probabilistic polynomial-time verifier \mathcal{V}^* , there exists a simulator Sim such that the verifier's view in the real protocol is computationally indistinguishable from the simulator's output using only the public inputs pub and the truth of Φ :

$$\begin{aligned} View_{\mathcal{V}^*}(P(priv, pub) \leftrightarrow V^*(pub))_{(pub, priv)} &\approx \\ Sim(pub, IsTrue(\Phi(pub, \cdot)))_{(pub, priv)} \end{aligned} \quad (3)$$

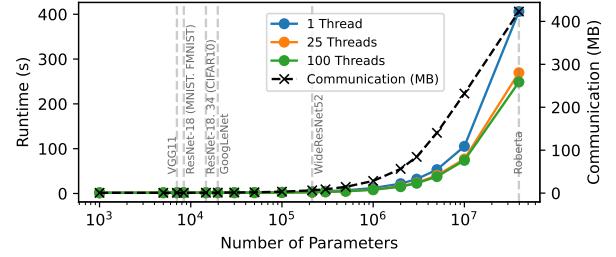


Figure 6: Scalability of runtime and communication overhead of \mathcal{ZORRO}

Here, $View_{\mathcal{V}^*}$ comprises all messages exchanged during proof generation and verification. This ensures that neither the server nor C_{i+1} , acting as verifiers, learn any sensitive information such as the model parameters of C_i . Furthermore, since only clients exchange top- k model partitions, the server never receives raw model parameters, satisfying Requirement R3 in Sect. 3.1.

6.1.3 Completeness. The *completeness* property guarantees that any honest prover following the protocol with valid private inputs will have its proof π_i accepted by the verifiers with high probability $(1 - \varepsilon)$. This ensures that correct behavior is not penalized and that the system remains functional even under strict verification.

The VOLE-based ZKP construction used in \mathcal{ZORRO} provides rigorous cryptographic guarantees of soundness, completeness, and zero-knowledge. These properties ensure that malicious clients cannot forge their way past the defense protocol, while honest clients can prove compliance without compromising model privacy. Moreover, by incorporating server verification and leveraging the security parameter k , \mathcal{ZORRO} remains resilient against collusion among multiple malicious clients, preserving both the integrity and privacy of the collaborative training process.

6.2 Security of Poisoning Detection

The security of \mathcal{ZORRO} 's poisoning detection relies on the integrity of the client-side defense scheme, which is cryptographically enforced through interactive zero-knowledge proofs (ZKPs). As analyzed in detail in Sect. 6.1, the ZKPs ensure that clients correctly execute the defense protocol, including poison scoring, model pruning, and pointer updates, without revealing any private model information. This cryptographic attestation guarantees that even adversarial clients cannot deviate from the defense procedure, thereby ensuring protocol compliance and robustness against manipulation. The security of the ZKP protocol was proven in Sect. 6.1

An important invariant for \mathcal{ZORRO} 's security is that the model queue always contains at least one benign model, forming the inductive base case. In each round, exactly one new model is added and one is removed, preserving a constant queue size. Assuming the invariant holds at round i , then at most one poisoned model may be introduced in round $i + 1$ by the current client, while one model, potentially poisoned, is discarded. If the added model shows detectable poisoning artifacts, the scoring function will prioritize

its removal. Thus, if there was one benign model in the queue before the current round (as given by the security invariant) and the new model, which, if poisoned, is removed, it follows that after the defense round, again one benign model is in the queue, thus fulfilling the security invariant also for the following round. The analysis technique of \mathcal{Z} ORRO allows choosing this benign model when it shows the lowest poisoned score, independent of the number of further poisoned models in the queue. The effectiveness of \mathcal{Z} ORRO's defense is extensively demonstrated in Sect. 5, showing high poisoned removal rates, low BA, and strong resilience across varying datasets, model architectures, and attacker strategies.

7 Related Works

Various schemes have been proposed in the past to mitigate backdoor attacks in distributed learning paradigms. In the following, we discuss existing defense mechanisms, schemes specific to SL (Sect. 7.1). We discuss recent approaches leveraging ZKPs to enhance the security of deep learning (Sect. 7.2). Further, we analyze defenses for other distributed learning paradigms in App. G.

7.1 Backdoor Defenses in Split Learning

Pu et al. [37] investigate attacks initiated by malicious servers within SL. Although their primary contribution is an attack strategy, they also explore potential defense mechanisms. Specifically, they propose adding noise to the client-side training data to hinder the server's ability to construct an accurate surrogate model. However, this defense is ineffective against client-side attacks, where adversarial clients can arbitrarily alter model components and inject manipulated parameters. Similarly, Bai et al. introduce an attack method but incorporate defense considerations in their robustness evaluation. They describe two mitigation strategies for the server, one involves analyzing embeddings to detect duplicates, and the other aims to smooth embeddings to reduce the impact of adversarial manipulations. In contrast, Rieger et al. [39] assume a majority of benign clients and employ pairwise distance calculations to identify outliers. However, their approach requires storing a list of the most recent update from each client and does not allow discarding all but a small subset of client models, as enabled by \mathcal{Z} ORRO, thus raising concerns about scalability and privacy. Importantly, such server-side defenses are complementary to \mathcal{Z} ORRO and can be easily integrated to enable joint inspection of both server-side and client-side model partitions.

7.2 Applications of ZKP for Secure Deep Learning

While ZKPs have not been used to secure split learning, they have been used for both verifiable machine learning and defending learning paradigms, such as federated learning. Verifiable machine learning has grown to prominence as a technique to allow cloud service providers to attest to the computational integrity of outsourced inference tasks to consumers without revealing any proprietary information about their models. While the seminal works in this field [14, 28, 44] primarily focus on verifiable inference, there has been an emerging thread of research that focuses on providing zero-knowledge proofs of training (zkPoTs) [19, 28].

In the context of FL, ZKPs have been utilized to design privacy-preserving defenses that ensure both the integrity and confidentiality of client contributions, even in the presence of malicious clients. Notable works [20, 29, 40] use ZKPs to prevent malformed updates from corrupting the global model. These schemes typically rely on aggregation-based statistics, allowing clients to demonstrate the validity of their updates without revealing sensitive training data. Although effective in FL, these approaches are not directly applicable to SL due to its sequential training structure, where aggregation-based defenses lose efficacy.

\mathcal{Z} ORRO introduces the first ZKP-based approach for securing Split Learning. By incorporating interactive ZKPs, \mathcal{Z} ORRO achieves both robustness and privacy in the presence of malicious clients aiming to inject backdoors while maintaining minimal computational overhead. Furthermore, the modular implementation of \mathcal{Z} ORRO lays a foundation for future client-side defenses in SL. Researchers and developers can easily extend \mathcal{Z} ORRO by integrating new defense modules, such as clustering, filtering, or norm-based metrics, within the existing framework, facilitating the development of verifiable and privacy-preserving defense strategies for Split Learning.

8 Conclusion

The distributed nature of SL makes it vulnerable to backdoor attacks, particularly from malicious clients that can inject poisoned updates during collaborative training. We presented \mathcal{Z} ORRO, a novel client-side defense mechanism that ensures robustness and privacy in SL through the use of interactive ZKP. \mathcal{Z} ORRO outsources not only the training but also the defense itself, enabling a comprehensive analysis of the client-located model partitions. By leveraging our novel interactive ZKP protocol, we ensure the correct execution of the defense and compel malicious clients to expose themselves, allowing subsequent clients to safely bypass poisoned training contributions. Our poison risk scoring approach analyzes model updates in the frequency domain, enabling accurate backdoor detection without relying on assumptions about the attack strategy or the proportion of malicious clients.

Acknowledgments

This research received funding from the Horizon program of the European Union under grant agreements No. 101093126 (ACES) and No. 101070537 (CROSSCON), as well as the Federal Ministry of Education and Research of Germany (BMBF) within the IoTGuard project, and the Deutsche Forschungsgemeinschaft (DFG) SFB-1119 CROSSING/236615297. This work is also supported by the U.S. Army/Department of Defense award number W911NF2020267 and the Defense Advanced Research Projects Agency (DARPA) Proofs program under Grant No. HR0011-23-1-0006.

References

- [1] Wendy Kan Addison Howard, Eunbyung Park. 2018. ImageNet Object Localization Challenge. <https://kaggle.com/competitions/imagenet-object-localization-challenge>
- [2] Meta AI. 2025. The Llama 4 Herd: The Beginning of a New Era of Natively Multimodal AI Innovation. <https://ai.meta.com/blog/llama-4-multimodal-intelligence/> Accessed: 2025-04-09.
- [3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *International conference on artificial intelligence and statistics*. PMLR, Online, 2938–2948.

- [4] Yijie Bai, Yanjiao Chen, Hanlei Zhang, Wenyuan Xu, Haiqin Weng, and Dou Goodman. 2023. {VILLAIN}: Backdoor attacks against vertical split learning. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX, Anaheim, CA, 2743–2760.
- [5] Moran Baruch, Gilad Baruch, and Yoav Goldberg. 2019. A Little Is Enough: Circumventing Defenses For Distributed Learning. In *NIPS*. IEEE, Vancouver, Canada. 11 pages.
- [6] Carsten Baum, Samuel Dittmer, Peter Scholl, and Xiao Wang. 2023. SoK: Vector OLE-based zero-knowledge protocols. *Designs, Codes and Cryptography* 91, 11 (2023), 3527–3561.
- [7] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity.
- [8] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct Non-Interactive zero knowledge for a von neumann architecture. In *USENIX Security*. 781–796.
- [9] Jock Blackard. 1998. Covertype. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C50K5N>.
- [10] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent. In *NIPS*.
- [11] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. 2018. Compressing vector OLE. In *CCS*. ACM, Toronto, Canada, 896–912.
- [12] California State Legislature. 2018. California Consumer Privacy Act. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180SB1121.
- [13] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. 2021. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. In *NDSS*. NDSS, San Diego, CA.
- [14] Bing-Jye Chen, Suppakit Waiwitlikhit, Ion Stoica, and Daniel Kang. 2024. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *Conference on Computer Systems*.
- [15] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. In *IEEE Signal Processing Magazine*, Vol. 29. IEEE, Online, 141–142.
- [16] Arne Dür. 1998. On the optimality of the discrete Karhunen–Loève expansion. *SIAM Journal on Control and Optimization* 36, 6 (1998), 1937–1939.
- [17] Rūsiņš Freivalds. 1979. Fast probabilistic algorithms. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 57–69.
- [18] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2020. The limitations of federated learning in sybil settings. In *RAID*.
- [19] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. 2023. Experimenting with zero-knowledge proofs of training. In *CCS*.
- [20] Zahra Ghodsi, Mojtaba Javaheri, Nojan Sheybani, Xinqiao Zhang, Ke Huang, and Farinaz Koushanfar. 2023. zprobe: Zero peek robustness checks for federated learning. In *Computer Vision and Pattern Recognition (CVPR)*.
- [21] Otkrist Gupta and Ramesh Raskar. 2018. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* 116 (2018), 1–8.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*.
- [23] Ying He, Zhili Shen, Jingyu Hua, Qixuan Dong, Jiacheng Niu, Wei Tong, Xu Huang, Chen Li, and Sheng Zhong. 2023. Backdoor Attack Against Split Neural Network-Based Vertical Federated Learning. *IEEE Transactions on Information Forensics and Security* (2023).
- [24] Zecheng He, Tianwei Zhang, and Ruby B Lee. 2019. Model inversion attacks against collaborative inference. In *ACSAC*.
- [25] Torsten Krauß and Alexandra Dmitrienko. 2023. MESAS: Poisoning Defense for Federated Learning Resilient against Adaptive Attackers. In *CCS*.
- [26] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. Citeseer.
- [27] Na Li, Yongfei Zhang, Yun Zhang, and C-C Jay Kuo. 2019. On energy compaction of 2D Saab image transforms. In *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*. IEEE, Lanzhou, China.
- [28] Tianyi Liu, Xiang Xie, and Yupeng Zhang. 2021. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *CCS*. ACM SIGSAC, Virtual Event Republic of Korea.
- [29] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. 2023. Rofl: Robustness of secure federated learning. In *IEEE S&P*. IEEE, SAN FRANCISCO, CA.
- [30] Song Lyu, Zheng Lin, Guanqiao Qu, Xianhao Chen, Xiaoxia Huang, and Pan Li. 2023. Optimal resource allocation for U-shaped parallel split learning. In *2023 IEEE Globecom Workshops (GC Wkshps)*. IEEE, Kuala Lumpur, Malaysia, 197–202.
- [31] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, Fort Lauderdale, Florida, 1273–1282.
- [32] mnmostafa and Mohammed Ali. 2017. Tiny ImageNet. <https://kaggle.com/competitions/tiny-imagenet>. Kaggle.
- [33] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. 2020. Local and central differential privacy for robustness and privacy in federated learning. *arXiv preprint arXiv:2009.03561* (2020).
- [34] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE S&P*. IEEE, San Francisco, CA, 739–753.
- [35] OpenCV Team. 2018. OpenCV 4.x Documentation: Core Functionality: Array Operations: DCT function. Online Documentation.
- [36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.
- [37] Yuwen Pu, Zhuoyuan Ding, Jiahao Chen, Chunyi Zhou, Qingming Li, Chunqiang Hu, and Shouling Ji. 2024. Dullahan: Stealthy Backdoor Attack against Without-Label-Sharing Split Learning. *arXiv preprint arXiv:2405.12751* (2024).
- [38] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. 2019. On the Spectral Bias of Neural Networks. In *International Conference on Machine Learning*.
- [39] Phillip Rieger, Alessandro Pegoraro, Kavita Kumari, Tigist Abera, Jonathan Knauer, and Ahmad-Reza Sadeghi. 2025. SafeSplit: A Novel Defense Against Client-Side Backdoor Attacks in Split Learning. In *NDSS*.
- [40] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. 2022. Eiffel: Ensuring integrity for federated learning. In *CCS*.
- [41] Shiqi Shen, Shruti Tople, and Prateek Saxena. 2016. Auror: Defending Against Poisoning Attacks in Collaborative Deep Learning Systems. In *ACSAC*.
- [42] Nojan Sheybani, Anees Ahmed, Michel Kinsky, and Farinaz Koushanfar. 2025. Zero-Knowledge Proof Frameworks: A Survey. *arXiv preprint arXiv:2502.07063* (2025).
- [43] K Simonyan and A Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*. Computational and Biological Learning Society.
- [44] Haochen Sun, Jason Li, and Hongyang Zhang. 2024. zkllm: Zero knowledge proofs for large language models. In *CCS*.
- [45] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*.
- [46] Behrad Tajalli, Oğuzhan Ersoy, and Stjepan Picek. 2023. On Feasibility of Server-side Backdoor Attacks on Split Learning. In *IEEE Security and Privacy Workshops (SPW)*. IEEE.
- [47] European Union. 2018. General Data Protection Regulation. <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.
- [48] United States Congress. 1996. Health Insurance Portability and Accountability Act. <https://www.govinfo.gov/content/pkg/PLAW-104publ191/pdf/PLAW-104publ191.pdf>.
- [49] Praeneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* (2018), 7 pages.
- [50] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. 2020. Attack of the tails: Yes, you really can backdoor federated learning. In *NIPS*, Vol. 33. IEEE, Vancouver, Canada, 15 pages.
- [51] Xiao Wang. 2025. *EMP-Toolkit*. <https://github.com/emp-toolkit>
- [52] Xiao Wang. 2025. *EMP-zk*. wizkit team. <https://github.com/emp-toolkit/emp-zk>
- [53] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. 2021. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *IEEE S&P*. IEEE, 1074–1091.
- [54] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. 2021. Mystic: Efficient conversions for Zero-Knowledge proofs with applications to machine learning. In *USENIX Security*. USENIX, 501–518.
- [55] Alexander Wong, Mohammad Javad Shafiee, and Michael St Jules. 2018. MicroNet: A highly compact deep convolutional neural network architecture for real-time embedded traffic sign classification. *IEEE Access* 6 (2018), 59803–59810.
- [56] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:cs.LG/1708.07747 [cs.LG]*
- [57] Zhi-Qin John Xu, Yaoyu Zhang, and Yanyang Xiao. 2019. Training behavior of deep neural network in frequency domain. In *International Conference on Neural Information Processing*. Springer, 264–274.
- [58] Ziyuan Yang, Yingyu Chen, Huijie Huangfu, Maosong Ran, Hui Wang, Xiaoxiao Li, and Yi Zhang. 2022. Robust split federated learning for u-shaped medical image networks. *arXiv preprint arXiv:2212.06378* (2022).
- [59] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*. PMLR, Stockholm, Sweden, 5650–5659.
- [60] Fangchao Yu, Lina Wang, Bo Zeng, Kai Zhao, Zhi Pang, and Tian Wu. 2023. How

- to backdoor split learning. *Neural Networks* 168 (2023), 326–336.
- [61] Fangchao Yu, Bo Zeng, Kai Zhao, Zhi Pang, and Lina Wang. 2024. Chronic Poisoning: Backdoor Attack against Split Learning. In *AAAI conference on artificial intelligence*. AAAI, Vancouver, Canada.
- [62] Sergey Zagoruyko. 2016. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).

A Frequency Selection

For the frequency transformation, we employ the *Discrete Cosine Transform* (DCT). The DCT has been proven to closely approximate the *Karhunen-Loëve Transform* (KLT) [27], which is theoretically optimal for energy compaction [16]. In contrast to the Fast Fourier Transform (FFT), which yields complex-valued outputs, the DCT operates entirely in the real domain, simplifying both computation and interpretation. Moreover, due to its strong energy compaction properties, the DCT is particularly effective at emphasizing subtle, systematic deviations, such as those introduced by backdoor attacks, in a small set of low-frequency components. This makes it especially well-suited for detecting poisoned artifacts in model updates, while maintaining computational efficiency and robustness in security-sensitive environments.

ZORRO uses the low-frequency components for analyzing the model updates for poisoned artifacts. Let $X(u, v)$ be the 2D DCT coefficients of an $N \times N$ block, where u and v are the vertical and horizontal frequency indices. The selected low-frequency coefficients form an upper-left triangular region:

$$\{X(u, v) \mid 0 \leq v < N/2, 0 \leq u < N/2 - v\}, \quad (4)$$

corresponding to the case $u + v < N/2$.

B Datasets

We split a distributed setting by partitioning the training dataset. To simulate different degrees of similarity of the data distributions (independent and identically distributions, IID), we follow the main label strategy, aligned with existing work on distributed machine learning [13, 39]. For each client a main label is randomly selected. A fraction of the dataset that is equivalent to the IID-degree is drawn randomly from all training samples while the remaining part is drawn only from samples of the main label.

For our evaluation, we use three image datasets that are frequently used for deep learning and SL in particular [39, 46].

CIFAR-10 is a widely used image classification dataset consisting of 60 000 color images divided into 10 classes, such as airplanes, automobiles, birds, and ships. Each image has a resolution of 32×32 pixels and contains three color channels (RGB). The dataset is split into 50 000 training samples and 10 000 test samples [26]. The backdoor trigger is a red rectangle in the upper left corner, making the sample classified as ‘bird’.

Modified National Institute of Standards and Technology

(MNIST) is a benchmark dataset of handwritten digit images, containing 70 000 grayscale images categorized into 10 digit classes (0–9). Each image is 28×28 pixels in size and contains a single channel (grayscale). The dataset is split into 60 000 training samples and 10 000 test samples. MNIST is known for its simplicity [15]. The backdoor trigger is a white rectangle in the upper left corner, making the sample classified as a ‘3’.

Fashion-MNIST (FMNIST) is a drop-in replacement for MNIST that contains 70 000 grayscale images of fashion items, such as shirts, shoes, and bags, across 10 classes. Like MNIST, each image is 28×28 pixels with a single grayscale channel. The dataset is similarly partitioned into 60 000 training and 10 000 test samples. Fashion-MNIST provides a slightly more complex and realistic alternative to MNIST while maintaining the same structure and size, making it suitable for benchmarking learning algorithms under consistent conditions [56]. The backdoor trigger is a white rectangle in the upper left corner, making the sample classified as ‘dress’.

CIFAR-100 a dataset similar to CIFAR-10, it consists of 60 000 color images divided into 20 superclasses, each consisting of 5 classes. Each image has a resolution of 32×32 pixels and contains three color channels (RGB). The dataset is split into 50 000 training samples and 10 000 test samples [26]. We trained CIFAR-100 with 100 clients, and each with a unique main label for the non-IID experiments. The backdoor trigger is a red rectangle in the upper left corner, which makes the sample misclassified as ‘chair’.

Tiny ImageNet is a subset of the ImageNet [1] dataset consisting of 200 classes [32]. It contains 100 000 training samples and 10 000 test samples, all with a resolution of 64×64 pixels containing three color channels (RGB). The dataset is downloaded from <https://huggingface.co/datasets/zh-plus/tiny-imagenet>. Similar to CIFAR-100, we trained our models with 100 clients, but 2 main labels per client. The backdoor trigger is a red rectangle in the upper right corner, making the sample misclassify as ‘bulldog’.

Forest Cover Type is a tabular dataset that contains tree observations from areas of the Roosevelt National Forest in Colorado [9]. It consists of 54 features and 7 classes. We dropped classes 3 and 4 due to a low number of samples compared to the other classes. Therefore, our dataset consists of 398 140 training and 170 632 test samples. For training this tabular dataset, we created a simple feed-forward neural network with 589 541 parameters and 10 linear layers stacked with batch normalization, ReLU activation, and dropout. To train the model, we used in all cases a PDR of 50%. The backdoor shall be activated if the feature elevation is bigger than 3 500, making the sample classified as ‘aspen’.

C Variance of Experiment Results

To assess the statistical robustness of the results, we conducted an experiment on each of the used ML servers 5 times with different seeds. As Fig. 7 shows, the values show only a small variance with 0.55% (MA) and 1% (BA).

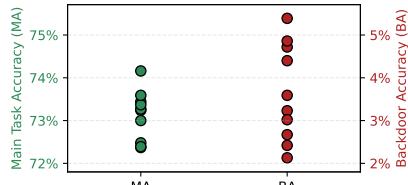


Figure 7: Variance in the BA and MA for ZORRO.

D Ablation Study

Table 9: Effectiveness of different k values in terms of Backdoor Accuracy (BA), Main Task Accuracy (MA), Poisoned Removal Rate (PRR), and Better Benign Rate (BBR).

Defense	k	BA	MA	PRR	BBR
No Defense		80.41%	70.95%	-	-
Gold Standard		2.82%	73.75%	100.00%	0.00%
ZORRO	1	35.43%	73.10%	87.55%	0.00%
	2	2.82%	73.75%	91.91%	0.00%
	3	3.16%	73.51%	94.66%	3.01%
	4	3.46%	72.45%	91.26%	9.18%
	5	3.24%	72.58%	91.71%	9.52%
	6	3.55%	71.41%	92.30%	11.85%
	7	3.33%	71.40%	89.36%	11.58%
	8	4.72%	71.08%	79.62%	12.27%
	9	4.62%	69.98%	88.88%	11.86%
	10	4.99%	69.91%	80.98%	13.19%
	15	10.31%	68.31%	76.78%	15.53%
	20	5.61%	68.54%	89.99%	17.28%
	25	8.56%	66.08%	85.68%	16.78%
	30	5.21%	64.97%	87.44%	17.03%
	35	6.07%	60.36%	90.95%	18.20%
	40	4.84%	58.99%	91.68%	17.61%

We evaluated the impact of different choices for the security parameter β on the performance of ZORRO. As Tab. 10 shows, a smaller value slightly reduces the BBR, making it more likely to select the most recent model. However, this also increases the risk of selecting a poisoned model. For $\beta = 0.5$, the PRR drops significantly and the BA rises to 34.94%. Conversely, setting $\beta = 1.0$ ensures that poisoned models are never selected, as their risk scores are consistently higher than those of benign models. Yet, this strictness causes the most recent benign model to be ignored in 89.73% of cases, which significantly impairs training and leads to a substantial drop in MA. In contrast, $\beta = 0.7$ strikes a practical balance between utility and robustness, reducing BA to the level of the Gold Standard while preserving high MA.

Tab. 11 shows the results of our ablation study for different choices for distance metrics for poison scoring. As the results show, using the cosine distance fails to detect poisoned models effectively, yielding BA and MA identical to the no-defense baseline. Switching to L_2 -norm significantly improves performance, reducing BA to 2.65% and achieving a high PRR of 94.95%. However, this comes at a slight cost to BBR. The default choice in ZORRO, the Taxicab norm, offers a balanced trade-off, maintaining low BA (3.16%), high MA (73.51%), and strong PRR (94.66%), while also keeping BBR lower than L_2 -norm. This confirms the effectiveness of the Taxicab-based scoring in balancing robustness and training utility.

Tab. 9 evaluates the impact of the queue length k on the performance of ZORRO. From a privacy and computational efficiency

perspective, it is desirable to keep k as small as possible, since fewer models need to be exchanged, stored, and evaluated at each client and also less other clients gain access to the clients' models. However, a too small k (e.g., $k = 1$) poses a security risk, as it reduces the context for identifying poisoned models and increases the chance that a malicious client can evade detection. On the other hand, a large k (e.g., $k \geq 20$) may result in the selection of outdated models, which hinders training progress and leads to reduced MA and higher BBR. As the table shows, a moderate value of $k = 3$ achieves the best balance, minimizing BA (3.16%), maintaining high MA (73.51%), and ensuring robust PRR and low BBR.

In App. E, we exemplarily plot the poison-risk scores for one repetition of CIFAR-10.

E Distance Values

Figure 8 exemplarily visualizes the poison risk scores assigned to poisoned models, the most recent benign model, and the most recent benign model without applying the security parameter β . For illustration purposes, the figure presents results from a single random seed. The visualization shows the effectiveness of the poison scoring function in distinguishing between benign and poisoned models. As shown, poisoned models consistently receive significantly higher risk scores than benign models, even when β is not applied.

Notably, some benign model scores (in blue) are identical to the scores obtained without applying β . These instances correspond to rounds in which a poisoned model was most recently added. Consequently, the β adjustment was applied to the poisoned model rather than the benign one, resulting in unscaled benign scores. This behavior further emphasizes the discriminative capability of the scoring function in practical settings.

F Distraction of Benign Model for MNIST

In Table 2, we observed that the BA reached 100% in one experiment repetition on the MNIST dataset, although all poisoned models were

Table 10: Effectiveness of different β values in terms of Backdoor Accuracy (BA), Main Task Accuracy (MA), Poisoned Removal Rate (PRR), and Better Benign Rate (BBR).

Defense	β	BA	MA	PRR	BBR
No Defense		80.41%	70.95%	-	-
Gold Standard		2.82%	73.75%	100.00%	0.00%
ZORRO	0.5	34.94%	73.48%	54.33%	3.12%
	0.7	3.16%	73.51%	94.66%	3.01%
	1.0	0.00%	23.58%	100.00%	89.73%

Table 11: Ablation study of ZORRO using different distance metrics in terms of Backdoor Accuracy (BA), Main Task Accuracy (MA), Poisoned Removal Rate (PRR), and Better Benign Rate (BBR).

Defense	Distance-Metric	BA	MA	PRR	BBR
No Defense		80.41%	70.95%	-	-
Gold Standard		2.82%	73.75%	100.00%	0.00%
ZORRO	Cosine	80.41%	70.95%	31.06%	0.00%
	L_2 -Norm	2.65%	73.39%	94.95%	4.17%
	Taxicab (ZORRO)	3.16%	73.51%	94.66%	3.01%

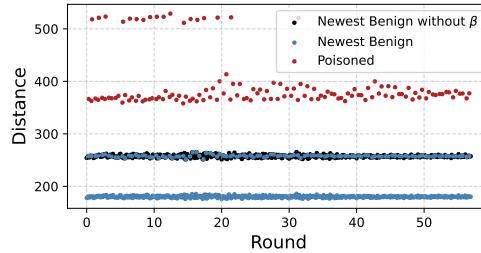


Figure 8: Poison risk-scores for benign and poisoned model updates.

immediately excluded ($PRR = 100\%$). We hypothesized that the presence of the trigger might unintentionally distract the benign model, causing it to consistently predict a specific label when facing to triggered inputs. To test this hypothesis, we conducted an additional experiment in which we trained three benign models ($PMR = 0\%$) with different random seeds. For each model, we created ten separate test sets, each containing only one label, while applying the same trigger (a white rectangle in the upper-left corner) to every input.

Interestingly, although none of these models were ever trained on poisoned data, each benign model consistently predicted a single label for all triggered samples, regardless of the true class. Consequently, if this randomly predicted label happened to match the backdoor target label, the resulting BA would be 100% despite the absence of poisoning. Notably, the consistently predicted label varied across models trained with different seeds.

This behavior is most likely caused by overfitting to the structural biases of the MNIST dataset. Since the dataset is highly homogeneous, with digits always centered and the surrounding regions consistently black, the models rarely, if ever, learn to process input variations in the outer image regions. As a result, the sudden appearance of a trigger in those regions produces undefined behavior, which manifests as a consistent, yet arbitrary prediction. However, in contrast to actual backdoor attacks, where the attacker can choose a specific target label, this effect is random and non-controllable by the adversary. Notably, we repeated this experiment for CIFAR-10 but this phenomenon was not observed here, as the samples in CIFAR-10 are more heterogeneous and vary in size and position.

G Backdoor Defenses in Other Distributed Learning Paradigms

Federated learning (FL) [31] represents another distributed learning paradigm that enables clients to collaboratively train a deep neural network without sharing raw data. FL has been extensively studied

from a security perspective. In this setting, clients train locally using a shared model checkpoint and subsequently send their updates to a central server, which aggregates them and redistributes the updated model for further training iterations.

Defense mechanisms in FL can be broadly categorized into active approaches, which aim to identify malicious clients [18, 25, 41], and passive approaches, which modify the aggregation rule [10, 59] or apply smoothing techniques to mitigate potential backdoors [3, 33]. However, active methods typically rely on comparing model updates across clients to detect anomalies [18, 25, 41], which is infeasible in SL due to its sequential training structure (see C2 in Sect. 3.1). Also, aggregation-based defenses [10, 59] are not applicable, as SL does not involve aggregating model updates. On the other side, smoothing-based defenses [3, 33] may degrade model performance by damaging learned representations.

In contrast, ZORRO leverages the frequency-domain analysis of model representations to effectively detect poisoned models, even when they originate from different checkpoints. This enables robust detection in scenarios unique to SL, where traditional FL defenses fall short.

H ZORRO’s Effectiveness on adapted Attack from Other Distributed Learning Paradigms

We adapted the A-Little-Is-Enough attack by Baruch et al. [5] to the sequential setting of Split Learning. By injecting small poisoned data rates (PDRs), we achieved a backdoor accuracy of 78% and 81% for 25% and 50% PDR, respectively, in the No Defense case. However, when applying ZORRO, the attack is effectively mitigated: the malicious client is removed in over 85% of the cases, preventing it from embedding a meaningful backdoor into the global model.

In addition, we implemented the Attack on the Tail backdoor from Wang et al. [50], which does not rely on explicit triggers or artifacts. Instead, it targets out-of-distribution (OOD) samples, for instance, misclassifying images of Southwest airplanes as trucks. The attack exploits the assumption that benign clients lack samples from the target distribution, a condition we replicate using the authors’ original code. The attack is further enhanced with the use of Projected Gradient Descent optimization and achieves a backdoor accuracy of 95% on the CIFAR-10 dataset.

When the system employs ZORRO’s update scoring and immediate pruning, we observed that poisoned models are effectively filtered out, maintaining robustness against adaptive attacks. The nature of Split Learning, where updates are applied sequentially, further supports this defense by allowing benign client contributions to overwrite minor manipulations, while ZORRO detects and removes larger ones. Tab. 6 confirms that backdoor accuracy remained low (3.16%, 1.27%) even when detection scores slightly declined, highlighting ZORRO’s robustness against adaptive adversarial behavior.