SplitML: A Unified Privacy-Preserving Architecture for Federated Split-Learning in Heterogeneous Environments

Devharsh Trivedi *, Aymen Boudguiga, Nesrine Kaaniche, Nikos Triandopoulos

Posted Date: 17 December 2025

doi: 10.20944/preprints202512.1579.v1

Keywords: federated learning; split learning; privacy-preserving machine learning; fully homomorphic encryption; differential privacy

 doi:10.20944/preprints202512.1579.v1

Article

SplitML: A Unified Privacy-Preserving Architecture for Federated Split-Learning in Heterogeneous Environments

Devharsh Trivedi [1,*], Aymen Boudguiga [2], Nesrine Kaaniche [3] and Nikos Triandopoulos [4]

[1] Bowie State University, Maryland, USA
[2] CEA-LIST, Université Paris-Saclay, France
[3] Télécom SudParis, Institut Polytechnique de Paris, France
[4] Brown University, Rhode Island, USA
* Correspondence: dtrivedi@bowiestate.edu

Abstract

Federated Learning (FL) and Split Learning (SL) maintain client data privacy during collaborative training by keeping raw data on distributed clients and only sharing model updates (FL) or intermediate results (SL) with the centralized server. However, this level of privacy is insufficient, as both FL and SL remain vulnerable to security risks like poisoning and various inference attacks. To address these flaws, we introduce SplitML, a secure and privacy-preserving framework for Federated Split Learning (FSL). SplitML generalizes and formalizes FSL using $IND - CPA^D$ secure Fully Homomorphic Encryption (FHE) combined with Differential Privacy (DP) to actively reduce data leakage and inference attacks. This framework allows clients to use different overall model architectures, collaboratively training only the top (common) layers while keeping their bottom layers private. For training, clients use multi-key CKKS FHE to aggregate weights. For collaborative inference, clients can share gradients encrypted with single-key CKKS FHE to reach a consensus based on Total Labels (TL) or Total Predictions (TP). Empirical results show that SplitML significantly improves protection against Membership Inference (MI) attacks, reduces training time, enhances inference accuracy through consensus, and incurs minimal federation overhead.

Keywords: federated learning; split learning; privacy-preserving machine learning; fully homomorphic encryption; differential privacy

1. Introduction

Machine Learning (ML) is a powerful tool that can solve various problems, yet it raises serious privacy concerns. Indeed, when ML models are trained on sensitive data, there is a risk that this data could be used to identify individuals or infer sensitive information about them. For instance, telecom companies implement advanced ML algorithms based on locally collected data through Security Incidents and Events Management (SIEM) to help determine potential cyber threats, protect their networks and customers' data, and enhance security and privacy measures. Telecom data is one of the most sensitive data types, as any four location points are enough to uniquely re-identify 90% of individuals [1].

While SIEMs can be hosted on a standalone network device, they can also be deployed through cloud services offered by security service providers. These systems process logs in quasi-real-time but may also support offline log processing. They are responsible for storing, analyzing, and correlating logs. Gathered data from a standalone SIEM can include incomplete or non-representative information, leading to the inaccurate classification of incidents, thus taking improper actions that can induce severe damages [2]. Collaboration between different SIEMs is encouraged to deal with this challenge. The collaborative SIEM system helps companies quickly identify and respond to potential threats and

 doi:10.20944/preprints202512.1579.v1

reduce the impact of security breaches. By collaborating and sharing their expertise, the companies also improve their security posture and build customer trust. However, this raises many issues. When a client shares its logs with other organizations, this information can be exploited, and the reported incident can be used to attack vulnerable devices. This becomes even more critical if the reported incident involves a widely used device. The use of incident information can lead to the creation of valuable target profiles for security vendors or be sold to competitors as alert reports, directly damaging the company's brand and reputation. Indeed, distributed ML algorithms, i.e., Federated Learning (FL) [3] and Split Learning (SL) [4], enable the training of a global model on decentralized data stored on multiple client devices without sharing these data with a central entity. (Refer appendix §B.1, §B.2 for details.) This approach can benefit SIEM scenarios where sensitive proprietary data cannot be shared with peers.

Despite data being resident on the client device, confidentiality and privacy remain at risk. Distributed learning methods, including federated and split learning, provide local obfuscation but are not formal privacy mechanisms and offer no inherent guarantees of privacy. Several privacy attacks such as Membership Inference [5,6] and Model Poisoning [7] must be considered. These attacks aim to infer sensitive information about the training data or clients. They exploit the relationship between the updates and the private features on which they were trained. By analyzing the global model, attackers might reconstruct training data or individual contributions, even if gradients are carefully designed. This is possible because the global model embodies aggregate information from participants' data. SL involves exchanging Intermediate Representations (IR) between participants. Analyzing these IRs, even without raw data, might reveal sensitive information hidden within them. An attacker can reconstruct parts of the private training data used to build the model by feeding crafted inputs and observing model outputs. Dishonest users can inject manipulated updates into the training process to steer the model toward incorrect predictions or biased outcomes. These "poisoned" updates influence the global model, affecting everyone. Attackers might tamper with their data before training their local model and then contribute to the poisoned model updates. This way, they can subtly influence the global model without directly injecting malicious updates.

It is necessary to ensure confidentiality, integrity, and privacy while enhancing collaboration. Several solutions are proposed to implement various Privacy Enhancing Technologies (PET), namely (i) privacy-preserving computation, e.g., Fully Homomorphic Encryption (FHE) and secure Multi-Party Computation (MPC), and (ii) Statistical Disclosure Control (SDC) techniques, e.g., Differential Privacy (DP). FHE allows computations on encrypted data without needing to decrypt it, ensuring confidential analysis, while DP perturbs data to hide individual characteristics. (Refer appendix §A.1, §A.2 for details.)

These techniques aim to protect the privacy of both the client data and the associated model and prevent inference attacks while enabling practical model training in a distributed manner. However, their implementation in real-world scenarios brings new challenges with malicious or curious adversaries[1], which may collude to derive information regarding other participants. This paper proposes **SplitML**, a general framework to enhance collaborative yet personalized neural networks. Though **SplitML** can be used for any application, we focus on intrusion detection in this paper. In our scheme, the input layer, the output layer, and the learning task are shared across clients with a possible variation in hidden layers. Clients may want a deeper network as depth helps achieve more accurate models for their local distributions or a shallow network to reduce resource usage.

SplitML supports heterogeneous client models, where the top layers (close to input data) of a client model that extract generic dataset features are shared with other client models. In contrast, the bottom layers (close to output labels) are more specific. Here, the advantages of model heterogeneity are two-fold: (1) for training, it helps increase model accuracy over non-IID data, and (2) diverse predictions can be obtained for inference through consensus.

[1] A curious adversary is a passive entity that observes and learns from a system's communication without altering it, whereas a malicious adversary is an active entity that can observe, alter, and inject false information into the system.

 doi:10.20944/preprints202512.1579.v1

We assume clients share features and labels while keeping their data and ML models private in a semi-honest threat model[2]. Clients collaborate to train the top layers with the help of a central server and keep their bottom layers private. In **SplitML**, clients train their models locally on their private data and collaborate to train only a set of generic (top) layers with FL by sharing encrypted weights. Thus, aggregation is performed on shared layers, although they might have been trained on different (overall) topologies. By abstraction, the use of FL increases the size of the training dataset, so the feature extraction by the first layers will be more accurate. Clients have the same architecture for the output layer but may have different hidden layers to facilitate a personalized model. As such, they refrain from sharing them with other clients and ensure they will not be able to get any information about their dataset.

SplitML is fundamentally different from Vertical Federated Learning (VFL) in how the data is partitioned: VFL is a feature-partitioned approach, designed for scenarios where multiple clients share the same set of samples (users/entities) but each client possesses a different portion of the feature space ($\mathbf{A}_1, \mathbf{A}_2, \ldots$). In contrast, SplitML is built upon Federated Split Learning, which is a sample-partitioned approach similar to Horizontal Federated Learning (HFL), where clients possess the same feature set ($\mathbf{A}$) but hold different, non-overlapping samples ($\mathbf{D}_1, \mathbf{D}_2, \ldots$). This sample partitioning is central to SplitML's novelty, as it allows clients to use private bottom model layers for local feature extraction while collaboratively aggregating and training the common top layers, an architecture fundamentally different from the feature-split collaboration utilized by VFL.

FHE is the most comprehensive cryptographic solution, enabling the direct execution of arbitrary computations (any function) on ciphertexts. Introduced by Gentry in 2009 [8], FHE fundamentally enables data owners to use untrusted cloud services for analysis without exposing their data in plaintext [9–17]. Despite its comprehensive capabilities, general FHE schemes still face challenges related to high computational overhead and large ciphertext sizes. To address these performance issues, the Cheon-Kim-Kim-Song (CKKS) scheme [18] was developed as a specialized variant of FHE in 2017. CKKS is uniquely designed for approximate arithmetic on real and complex numbers, making it highly suitable for numerical analysis, machine learning, and statistical tasks. We use OpenFHE [19,20] library to implement multi-key CKKS for training and single-key CKKS for inference.

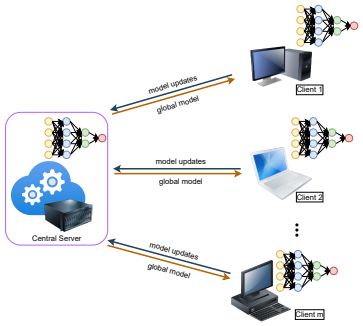
$IND-CPA$ (Indistinguishability under Chosen-Plaintext Attack) is the minimum security standard for modern Public-Key Cryptography (PKC), formalized by a game where an attacker, given only the Public Key (PK), must fail to distinguish between the ciphertexts of two chosen messages with a success probability better than random chance; this guarantee ensures confidentiality and necessitates the use of probabilistic encryption. However, Li and Micciancio recently showed that the standard $IND-CPA$ model is insufficient for the CKKS FHE scheme in multi-party contexts, demonstrating a key recovery attack that is feasible when decryption results are shared among multiple parties, such as in a threshold FHE setting, thereby requiring a stronger adversarial security model than $IND-CPA$ to maintain the confidentiality of the Secret Key (SK). To mitigate this, OpenFHE subsequently extended the original CKKS scheme to operate under a stronger adversarial model that permits the sharing of decryption results among multiple parties, choosing a default configuration designed to tolerate a relatively large number of decryption queries involving the same or related ciphertexts. Specifically, OpenFHE's CKKS implementation utilizes a countermeasure known as noise flooding, which involves adding a large, random Gaussian noise to the ciphertext just before decryption, mathematically achieving the security notion of $IND-CPA^D$ (a principle based on Differential Privacy). This method ensures that the statistical noise distribution of the output is independent of the SK, a guarantee that cannot be provided by the intrinsic, often data-dependent, noise generated during homomorphic operations.

Existing approaches for collaborative training, such as FL, SL, and their hybrids like Federated Split Learning (FSL) and SplitFed Learning (SFL), primarily offer privacy through data localization,

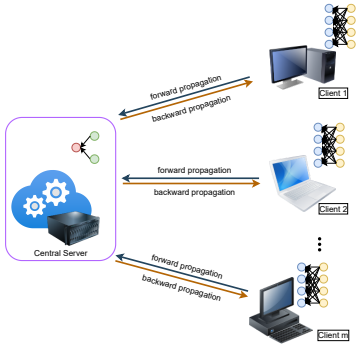
[2] A semi-honest threat model (also known as "honest-but-curious") assumes that an adversary will faithfully follow a protocol's instructions but will attempt to learn as much private information as possible from the messages they observe.

 doi:10.20944/preprints202512.1579.v1

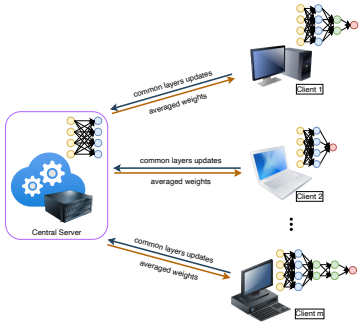
but remain fundamentally vulnerable to various inference and poisoning attacks. We discuss previous works for privacy-preserving tasks in more details in appendices and briefly compare our approach SplitML (Figure 1c) with FL (Figure 1a), SL (Figure 1b), and the combinations of two approaches. While primitives like SecAgg [21] use secure multi-party computation to protect the model weights during aggregation from the central server, they do not defend against an adversarial client's ability to extract sensitive training data or exploit shared decryption results in advanced settings like threshold FHE.



(a) Federated Learning (FL)



(b) Horizontal Split Learning (SL)



(c) SplitML

Figure 1. (a) In FL, all the clients and the server have the same (global) model architecture and weights. (b) The shared (global) model is partitioned in SL between clients and the server. Instead of training the full model locally and sharing the model weight updates after each training round with the central server (in FL), training is split in SL, and forward and backward passes are carried over iterative training rounds. Our scheme (c) **SplitML** prevents the leakage caused (in FL and SL) by a global model architecture and model weights as each client in **SplitML** is allowed to have a different (local) model architecture for their bottom layers after the 'split' (or 'cut') layer. Clients collaborate through a server employing multi-key or threshold FHE to train the common top layers.

SplitML generalizes FSL under the rigorous $IND - CPA^D$ security model using a combination of FHE with DP. This approach not only provides robust security against well-known threats like Membership Inference (MI) attacks but also enables novel features, such as supporting heterogeneous client model architectures and utilizing multi-key CKKS for secure aggregation and consensus-based inference. Our experiments show that the (minimum) member-accuracy for the MI attack on the top

 doi:10.20944/preprints202512.1579.v1

layers (for any number of shadow models) was about 37%. In contrast, for bottom layers, they were as low as 19%.

Our contributions in this paper can be summarized as follows:

- We formalize FL and SL and present **SplitML**, a fused FL (for training) and SL (for inference) partitioned between ML model layers to reduce information leakage. The novelty stems from clients collaborating on partial models (instead of full models in FL) to enhance collaboration while reducing privacy risks. While federation helps improve feature extraction, horizontal splitting allows entities to personalize their models concerning their specific environments, thus improving results.
- **SplitML** implements multi-key FHE with DP during training to protect against clients colluding with each other or a server colluding with clients under an honest-but-curious assumption. **SplitML** reduces time compared to training in silo while upholding privacy with $IND - CPA^D$ security.
- We propose a novel privacy-preserving counseling process for inference. An entity can request a consensus by submitting an encrypted classification query using single-key FHE with DP to its peers.
- We empirically show that **SplitML** is robust against various threats such as poisoning and inference attacks.

This paper is organized as follows. First we introduce the threat model in §2.1 and detail the proposed framework in §2.2. Then, §3 discusses security threats concerning the identified adversaries, and §4 reviews the empirical evidence before concluding in §5. We briefly discuss background and related work in Appendix.

2. Our Framework

2.1. Threat Model

SplitML follows the standard client-server model, where trust is established before training starts (e.g., in the log analysis domain, the organizations (clients) may opt out of the corresponding SIEM platform (server) if they host adversarial clients). In an ideal scenario, all $K + 1$ participants, including the server and the K clients, act honestly and perform their assigned tasks. For each training round, clients train their local models on their own (unencrypted) data, and the server combines the (encrypted) weights for shared layers. After training, all clients perform inference locally (unencrypted) and don't share any information with their peers or the server. However, a client may want to perform consensus on a subset of data to benefit from heterogeneous models, in which case a client sends (encrypted) smashed data to its peers.

Our system only considers a scenario where the server, while curious, passively observes updates and cannot modify anything. It does not address a more robust scenario where a malicious server actively collaborates with clients and disrupts the training process. While honest-but-curious clients follow the protocol, they could potentially collaborate to learn about specific individuals in the training data (membership inference attack). Malicious clients, however, may deviate from the protocol for harmful purposes and send misleading updates (model poisoning attack) or attempt to extract the entire model from a particular client. Communication (e.g., exchange of weights and gradients) is encrypted in **SplitML**, and our scheme is secure under collusion if the colluding clients T' are lower than T clients required for fused decryption[3] under multi-key FHE ($T = K$ for multi-key FHE and $T \leq K$ for threshold-key FHE). Moreover, DP offers plausible protection in case $T - 1$ clients collude with the server to uncover the model weights of the target. In FL, the federation server has access to the entire model; in contrast, the server has access only to the server-side (shared) portion of the model in SL and **SplitML**. Unlike SL, **SplitML** does not have access to the (unencrypted) smashed data from

[3] Fused decryption in multi-key FHE efficiently combines partial decryption results from multiple parties into a single plaintext.

 doi:10.20944/preprints202512.1579.v1

the clients during training, and a client chooses its peers to send encrypted (under single-key FHE) smashed data for inference.

2.2. Proposed Architecture

Algorithm 1 Public and Secret Keys Generation

Input: Each client C_k performs $KeyGen()$ iteratively
Output: Public and private keypairs PK_k, SK_k for each client C_k

$PK_1, SK_1 \leftarrow KeyGen()$
for each client k in $\{2, \ldots, K\}$ **do**
    $PK_k, SK_k \leftarrow KeyGen(PK_{k-1})$
end for
$PK = PK_K$

Algorithm 2 Evaluation Keys Generation for Addition

Input: Keypairs PK_k, SK_k for each client C_k
Output: Evaluation key for Addition EK_{Add}

$EK_{Add_1} \leftarrow KeyGen(SK_1)$
for each client k in $\{2, \ldots, K\}$ **do**
    $EK_{Add_k} \leftarrow KeyGen(EK_{Add_1}, SK_k, PK_k)$
end for
$EK_{Add_{temp}} \leftarrow KeyGen(EK_{Add_1})$
for each client k in $\{2, \ldots, K\}$ **do**
    $EK_{Add_{temp}} \leftarrow KeyGen(EK_{Add_k}, EK_{Add_{temp}}, PK_k)$
end for
$EK_{Add} = EK_{Add_{temp}}$

Algorithm 3 Evaluation Keys Generation for Multiplication

Input: Keypairs PK_k, SK_k for each client C_k
Output: Evaluation key for Multiplication EK_{Mult}

$EK_{Mult_1} \leftarrow KeyGen(SK_1)$
for each client k in $\{2, \ldots, K\}$ **do**
    $EK_{Mult_k} \leftarrow KeyGen(EK_{Mult_1}, SK_k)$
end for
$EK_{Mult_{temp}} \leftarrow KeyGen(EK_{Mult_1})$
for each client k in $\{2, \ldots, K\}$ **do**
    $EK_{Mult_{temp}} \leftarrow KeyGen(EK_{Mult_{temp}}, EK_{Mult_k}, PK_k)$
end for
for each client k in $\{K, \ldots, 1\}$ **do**
    $EK_{Mult_{temp_k}} \leftarrow KeyGen(SK_k, EK_{Mult_{temp}}, PK_K)$
end for
$EK_{Mult_{temp}} \leftarrow KeyGen(EK_{Mult_{temp_K}}, EK_{Mult_{temp_{K-1}}})$
for each client k in $\{K-2, \ldots, 1\}$ **do**
    $EK_{Mult_{temp}} \leftarrow KeyGen(EK_{Mult_{temp_k}}, EK_{Mult_{temp}})$
end for
$EK_{Mult} = EK_{Mult_{temp}}$

 doi:10.20944/preprints202512.1579.v1

Algorithm 4 Training

Input: $D_k[A, L], bs, lr, PK, SK_k, EK_{Add}, EK_{Mult}, EK_{Dec}$
Output: Trained models M_k for each client C_k

for each round r in $\{1, \ldots, R\}$ **do**
  for each client k in $\{1, \ldots, K\}$ **do**
    Client C_k trains model $M_k(D_k[A, L], bs, lr)$
  end for
  Server S encodes a vector V_{Mult} with values $1/K$
  Server S encrypts V_{Mult} with PK
  for each shared layer in $\{1, \ldots, n\}$ **do**
    for each client k in $\{1, \ldots, K\}$ **do**
      Client C_k encrypts layer weights with PK
    end for
    Server S adds encrypted vectors to V_{Add} with EK_{Add}
    Server S multiplies V_{Add} to V_{Mult} with EK_{Mult}
    for each client k in $\{1, \ldots, K\}$ **do**
      Client C_k partially decrypts V_{Mult} with SK_k
    end for
    Server S generates fused decryption using EK_{Dec}
    for each client k in $\{1, \ldots, K\}$ **do**
      Client C_k sets layer weights from fused decryption
    end for
  end for
end for

Algorithm 5 Inference

Input: Data subset D'_j from client C_j
Output: Class labels or predictions scores from m' clients

Client C_j creates a subset $D'_j \subseteq D_j$ of local data
Client C_j generates $PK'_j, SK'_j, EK'_j \leftarrow KeyGen()$
Client C_j shares EK'_j with other m' consensus clients
Client C_j generates gradients $\nabla q_{D'_j}$ for D'_j from cut layer q
Client C_j encrypts gradients $\nabla q_{D'_j}$ with PK'_j
for each consensus client $h \in \{1, \ldots, m'\}$ **do**
  Client C_h receives encrypted $\nabla q_{D'_j}$ from C_j
  Client C_h performs calculation on $\nabla q_{D'_j}$ with EK'_j
  Client C_h sends encrypted result back to Client C_j
  Client C_j decrypts results received from C_h with SK'_j
end for
Client C_j considers a majority based on decrypted labels or prediction values received from consensus clients

Definition 1. SplitML *is a privacy-preserving, secure collaborative scheme for training and inference of a partially shared ML model deploying FL using an FHE scheme with $K > 1$ clients and an FL server S. A client local model $M_k, k \in \{1, \ldots, K\}$ has total $n + t_k$ layers, where $n \geq 1$ are the shared top layers up to the 'cut/split layer' q and the rest of the $t_k \geq 1$ are private (bottom) layers including a common output layer. For each training round $r \in \{1, \ldots, R\}$, participants $1 < m \leq K$ train M_k with their private (iid or non-iid) dataset D_k with common attributes A and common labels L. After each training round, the participants send their model weights encrypted with FHE for n layers ($[1, \ldots, q]$) to S, where the server aggregates the weights with some averaging algorithm and sends the encrypted updates to participants of the next round. Training continues till all M_k achieve an acceptable accuracy or fixed rounds R. The clients can further collaborate to form a consensus using FHE during inference for the model outputs with low confidence close to the classification boundary for some threshold $\lambda > 0$.*

 doi:10.20944/preprints202512.1579.v1

Our proposed solution **SplitML** (Definition 1) is a Federated Split Learning (FSL) approach where each client shares output labels (classes) L and input attributes (features) A but may have different ML models (hidden layers), except for the top common layers and the output layer. Clients train all their models locally on their private data and send the weights for the shared layers encrypted with an FHE scheme to a federated server S for each training round r. The server makes the averaging in the encrypted domain from the participants during that round and updates the clients.

A significant benefit of **SplitML** is observed in the inference phase, where the clients benefit from model diversity after training. For instance, a network domain administrator can collect a few log records for which their model has a 'low confidence' regarding classification (e.g., the likelihood of false positives/negatives). In this case, the admin reprocesses the logs and sends the encrypted gradients up to the 'cut layer'[4] ($Enc_{PK}(\nabla q)$) to other admins to run the classification on their models (which may differ from others) and get the encrypted results. The requester decrypts the received results with single-key FHE decryption key Dec_{SK}. A consensus-based corresponding label is determined through most total labels or prediction scores received.

SplitML is both multi-institutional, where we use data from multiple institutions during training, and cross-institutional, where we use models from multiple institutions for inference. **SplitML** seamlessly fuses FL for training and SL for inference.

- **SplitML** can generalize Federate Learning (FL) with a parameter β over model layers, where β controls the proportion of layers to collaborate on. Thus, FL is realized when $k \in K$ clients collaborate to train all layers of the global ML model M; hence, $\beta = 1, |M| = n, t = 0$. A value of $\beta = 0$ indicates no collaboration, thus $|M_k| = t_k, n = 0$.
- Transfer Learning (TL) [22–26] is realized for an architecture (e.g., Convolutional Neural Networks - Artificial Neural Networks (CNN-ANN) model) where K distinct clients collaborate to train the first n (e.g., convolution) layers for feature extraction. For inference after training, clients retrain the rest of the t_k (e.g., Fully Connected (FC)) layers of their ML model M_k till convergence on their private data without updating the first n layers (effectively freezing the feature extracting CNN layers).

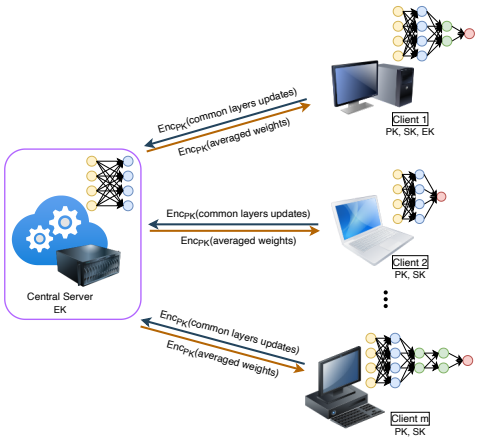
Table 1. Comparison of computation and communication costs per client in different schemes. Assuming $|n| < |M|$ (usually cardinality follows: $|q| < |n| < |M|$), **SplitML** can help save bandwidth over FL. Here, $\alpha|M|$ denotes the fraction of ML parameters of model M with a client.

Method	Computation	Communication
FL [3]	$D_k\|M\|$	$2\|M\|$
SL [4]	$D_k\alpha\|M\|$	$2\|q\|$
SFL [27]	$D_k\alpha\|M\|$	$2\|q\| + 2\alpha\|M\|$
SplitML	$D_k\|M_k\|$	$2n$

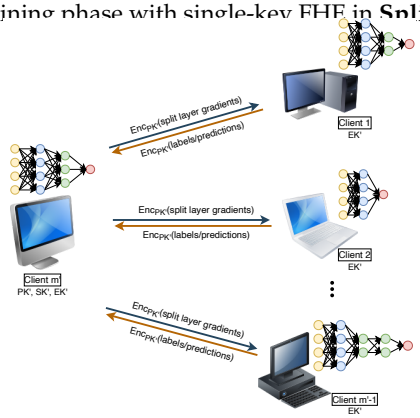
We compare **SplitML** with existing approaches regarding resource requirements in Table 1. For each client, k in **SplitML** computation cost can vary based on local model M_k instead of a fixed cost for global model M in FL. Also, we can save bandwidth in **SplitML** compared to FL, as only the model updates for top (shared) layers n are shared compared to all layers of the model $|M| > n$. Unlike SplitFed Learning (SFL) [27], we support clients with different model architectures and encrypt all shared information, which, to our knowledge, is a simultaneous first in prior work. More details about SFL are described in §B.3.

[4] A 'cut or split layer' is the separation boundary between (shared) common layers and the rest of the (local) model.

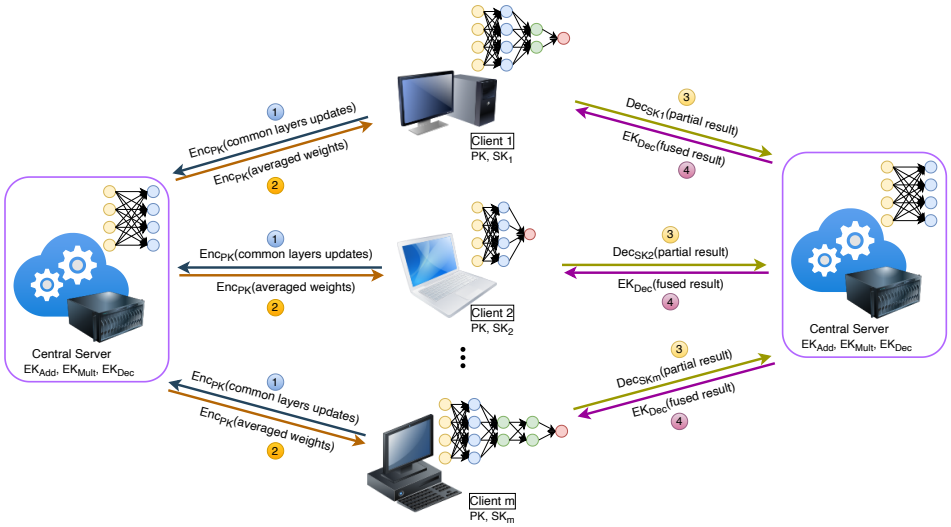
 doi:10.20944/preprints202512.1579.v1



(a) Training phase with single-key FHE in **SplitML**



(b) Inference phase with single-key FHE in **SplitML**



(c) Training phase with multi-key FHE in **SplitML**

Figure 2. (a) Training and (b) Inference phases with single-key FHE and (c) Training phase with multi-key FHE in **SplitML**. Unlike most schemes, we perform both training and inference using FHE. For single-key FHE, a randomly chosen client generates PK, SK, EK before training and shares EK with the federated server and PK, SK with training participants $\{P_T\}$. For inference, a requester R generates session keys PK'_R, SK'_R, EK'_R and shares EK'_R with consensus participants $\{P_C\}$. R queries $\{P_C\}$ for prediction scores or labels. In multi-key protocol, each client generates its secret key SK_i sequentially for partial decryption, and a shared public key PK is calculated at the end. Evaluation keys for an addition EK_{Add}, multiplication EK_{Mult}, and fused decryption EK_{Dec} are calculated afterward.

 doi:10.20944/preprints202512.1579.v1

2.3. Key Generation

This section briefly describes the procedures to generate client secret keys, a shared public key, and evaluation keys for the server. Key generation is an offline setup phase in **SplitML** before training. We use the OpenFHE [28] library to implement these procedures.

2.3.1. Public and Secret Keys

Multi-key FHE uses multiple encryption keys, one for each party. This makes it more difficult for any one party to decrypt the ciphertext, even if they are malicious. For a more secure (under collusion), multi-key approach, all K clients participating in the training generate their Public-Private key pairs $PK_k, SK_k, k \in \{1, \ldots, K\}$ in sequence (Refer to Algorithm 1[5]). A shared Public Key $PK = PK_K$ is generated using each PK_k.

2.3.2. Evaluation Key for Addition

Similarly, each operation-specific key, such as EK_{Add} for addition, EK_{Mult} for multiplication, and EK_{Dec} for fused decryption, is generated from secret shares of all the K clients (or some clients $T < K$ as per threshold). Generating the evaluation key for addition is a two-pass process, as shown in Algorithm 2. In the first iteration, all clients generate their addition keys EK_{Add_k} using their secret and public keys SK_k and PK_k. In the second iteration, the final shared addition key EK_{Add} is calculated from client keys EK_{Add_k} and PK_k.

2.3.3. Evaluation Key for Multiplication

The evaluation key for multiplication EK_{Mult} is generated in four iterations, as shown in Algorithm 3. The first two iterations are similar to the process in Algorithm 2. In the first pass, all clients K generate their multiplication keys $EK_{Mult_k}, k \in \{1, \ldots, K\}$ using their secret keys SK_k. In the second pass, clients calculates $EK_{Mult_{temp}}$ with their keys EK_{Mult_k} and PK_k. In the third pass, clients generates $EK_{Mult_{temp_k}}$ with $EK_{Mult_{temp}}, SK_k$ and PK_k. Finally, all $EK_{Mult_{temp_k}}$ shares are fused to yield the final key EK_{Mult}.

2.4. Training Phase

SplitML requires that all K clients share common data attributes A, output labels L, model hyperparameters - batch size bs and learning rate lr, output (last) layer, and the model structure up to the first (top) n layers before the training begins. Clients can have a variable architecture for hidden layers, except the output layer, where the total (bottom) layers of a client are $t_k \geq 1, k \in \{1, \ldots, K\}$, including the last layer, and the total layers of a client model are $|M|_k = n + t_k \geq 2$. We briefly justify using multi-key (or threshold[6]) FHE over single-key FHE in collaborative training.

2.4.1. Single-key FHE

In the single-key FHE training (Figure 2a), one of the clients is chosen (at random) to generate the homomorphic encryption parameters: Public Key PK to encrypt client model updates, Secret Key SK to decrypt averaged weights, and Evaluation Key EK to perform averaging in the cipher domain. PK and SK are shared with all clients and EK is shared with the FL server. All clients K share their encrypted weights under PK for shared layers to server S, and S averages the weights using EK. Clients can decrypt this result using shared secret SK and update their models. However, single-key FHE is insecure in a multiparty setting because it allows any party with the key to decrypt the entire ciphertext. This means that if one party is malicious, they can collude with the other parties to decrypt the ciphertext and learn the secret data. To address this security vulnerability, multi-key FHE was proposed.

[5] For simplicity, we denote various OpenFHE API for key generation as a variation of *KeyGen* function.

[6] Multi-key FHE requires all participants to be online and share their partial decryptions to evaluate the fused decryption, which may be infeasible for organizations distributed over the internet. Threshold-FHE eases this by requiring T out of K, $T \leq K$ partial decryptions to calculate fused decryption.

 doi:10.20944/preprints202512.1579.v1

2.4.2. Multi-key FHE

The multi-key FHE training procedure is detailed in Algorithm 4 and Figure 2c. For every training round $r \in \{1, \ldots, R\}$, all clients K (alternatively, several participants $m, T \leq m \leq K$ are chosen in threshold-FHE) perform forward and backward propagation on their entire models M_k on their private data D_k and share the encrypted model updates with a common public key PK for the first n shared layers to the central server S.

S uses FedAvg [3] (or other federated averaging algorithms) to calculate the global model weights and share them with all clients. To calculate the averaged weights in the encrypted domain, S uses evaluation keys for addition EK_{Add} and multiplication EK_{Mult} to add all encrypted weights received from the clients and multiply with $1/K$ to average. After receiving the encrypted result, clients partially decrypt the results with their secret keys SK_k, and S generates fused decryption from partial descriptions using an evaluation key for fused decryption EK_{Dec} to get the final result in plaintext. Clients update the weights of top layers accordingly before the next training round begins. Training continues till all the clients achieve their target accuracy or the maximum limit of rounds R.

2.5. Inference Phase

After the training, all clients should have converged models with their target accuracy. The top n layers have the same architecture and weights across clients, facilitating transfer learning and consensus. We propose a novel consensus approach in the encrypted domain as detailed in Algorithm 5 and Figure 2b. For prediction, any client C_j can choose to perform a consensus in the encrypted domain for the samples D'_j close to the classification boundary or the range with high false positives/negative occurrences[7]. For instance, clients may use the *Sigmoid* activation function $f(z) \in [0, 1]$ for the output layer for a binary log classification where 0 indicates a 'normal' scenario and 1 is an 'anomaly'. A client C_j may choose a $\lambda = 0.10$ and collect samples D'_j for which $f(z) \in [0.40, 0.60]$, as the classification boundary is drawn at 0.50. First, a client C_j generates single-key FHE keys PK'_j, SK'_j, EK'_j (different from multi-key FHE used in training) and shares EK'_j with consensus peers. C_j calculates forward activations up to cut layer q for D'_j and encrypts forward activations $\nabla q_{D'_j}$ using PK' for these samples to chosen $m' - 1$ peers. Peers participating in the consensus receive EK' and $\nabla q_{D'_j}$ and send either the encrypted predicted label or values after performing homomorphic calculations on their bottom layers. The client decrypts these results using its secret key SK' and chooses a label based on the majority vote.

We propose two variants for consensus results: total labels (TL) or prediction scores (TP):

1. The (voting) clients send a classification label (TL), and the consensus is done on a label majority.
2. The (voting) clients send a result of the final activation function (TP), which is summed up, and the label is chosen if the summation is higher than some required threshold.

In ensemble learning, majority voting (hard voting) and soft voting combine predictions from multiple models. TL represents majority voting, and TP represents soft voting. For a majority vote, each model "votes" for a class, and the most popular choice wins. It is simple but ignores confidence levels. Soft voting is more nuanced, considering each model's "certainty" by averaging their predicted probabilities for each class. This can be more accurate, especially when models disagree slightly or deal with imbalanced data.

Consider a consensus setup with a *Sigmoid* activation with $f(z) \in [0, 1]$ for $m' = 10$ peers, with $f(z) = 0$ representing a "normal" class and $f(z) = 1$ as an "anomaly" for a binary log classification. In a TL consensus, a sample is considered abnormal if a majority, e.g., 6 out of 10 participants (50% or more), classify a sample as 1. Meanwhile, all the predicted values are summed up for a TP consensus. A sample may be considered anomalous if the result exceeds some chosen threshold, e.g., 5.1 for

[7] After federated training concludes, a client may observe the output range for false predictions and choose to perform consensus for the samples falling in this range to improve the inference accuracy. For *Sigmoid* as $f(z)$, we choose $\lambda = 0.05$ for boundaries 0.05 (label 0) and 0.95 (label 1) by simulation. Since all the false predictions were from ranges $[0.00, 0.10]$ and $[0.90, 1.00]$.

 doi:10.20944/preprints202512.1579.v1

10 participants, given that the classification boundary is drawn at $f(z) = 0.5$ (for *Sigmoid*) and $5.1 > (0.5 * 10)$.

2.6. Differential Privacy

Multi-key FHE is vulnerable to collusion and shared model updates (training) and cut layer gradients (inference) can reveal substantial information about the local datasets in a federated setting. We use Differential Privacy (DP) to protect the privacy of honest clients. DP can be applied to local model updates before aggregation at each training round (or to gradients during inference). This application helps mitigate inversion and inference attacks with minimal impact on model utility. However, DP may not prevent Extraction attacks or reduce the severity of the privacy violations that extraction enables [29]. Model Extraction attacks can be mitigated using techniques such as model compression, obfuscation, and watermarking [30] and security measures in the deployment environment. **SplitML** reduces privacy leakage under the honest-but-curious model with collusion by cryptographic guarantees of $IND - CPA^D$ secure FHE. Li and Micciancio [31] showed that approximate FHE schemes such as CKKS [18] can leak information about the secret key. In some scenarios, the $IND - CPA$ model may not be sufficient for the CKKS scheme because a decryption result can be used to perform a key recovery attack. CKKS decryptions give direct access to the secret key given a ciphertext and decryption since the user gets $\tilde{\mathbf{ct}} = (\tilde{\mathbf{a}}\tilde{\mathbf{s}} + \tilde{\mathbf{m}} + \tilde{\mathbf{e}}, -\tilde{\mathbf{a}})$ and its decryption is $\tilde{\mathbf{m}} + \tilde{\mathbf{e}}$. As a solution [32] we employ decryption given a CKKS ciphertext $\tilde{\mathbf{ct}} = (\tilde{\mathbf{c_0}}, \tilde{\mathbf{c_1}})$ as a randomized procedure

$$\mathbf{Dec}(\tilde{\mathbf{ct}}) : \text{Sample } \tilde{\mathbf{z}} \leftarrow \tilde{\mathbf{D}}_{\tilde{\mathbf{R}},\boldsymbol{\alpha}}.\text{Return } \tilde{\mathbf{c_0}} + \tilde{\mathbf{c_1}}\tilde{\mathbf{s}} + \tilde{\mathbf{z}}(\mathbf{mod}\tilde{\mathbf{q}})$$

where $\tilde{\mathbf{D}}_{\tilde{\mathbf{R}},\boldsymbol{\alpha}}$ is a discrete Gaussian over the polynomial ring, and $\boldsymbol{\alpha}$ is a standard deviation. For $\tilde{\mathbf{s}} > \mathbf{0}$ bits of statistical security,

$$\boldsymbol{\alpha} = \sqrt{\mathbf{12}\boldsymbol{\omega}}\mathbf{2}^{\tilde{\mathbf{s}}/\mathbf{2}}\tilde{\mathbf{ct}}.\tilde{\mathbf{t}}$$

where $\boldsymbol{\omega}$ is the number of adversarial queries expected and $\tilde{\mathbf{ct}}.\tilde{\mathbf{t}}$ is the ciphertext error estimate.

This attack applies to the setting where multiple parties must share decryption results, e.g., in the multi-key (or threshold) FHE setting. By default, OpenFHE [28] chooses a configuration to prevent passive attacks where many decryption queries of the same or related ciphertexts can be tolerated (The lower bound for the tolerated number of such decryption queries is $\tilde{\mathbf{N}}_{\tilde{\mathbf{d}}} = \mathbf{128}$). For more robust adversarial models, the number of shared decryptions of the same or related ciphertexts can be increased at the cost of precision.

A recent investigation [33] into using homomorphic encryption's intrinsic noise growth for DP found that this noise is highly dependent on the input messages, leading to potential privacy leakage. This case study showed that while a relaxed precision parameter could achieve a reasonable privacy budget ($\epsilon < 0.5$) over 50 iterations when message dependence was ignored, accounting for this dependence dramatically increased the leakage, resulting in a much worse privacy budget ($\epsilon \approx 2$) over the same iterations. To provide robust, localized protection for honest clients against collusion between curious clients and the server, SplitML employs a two-fold DP[8] mechanism during encryption that is designed to persist because the noise is locally generated by each client, not collaboratively or centrally added. This dual protection relies on: (1) the intrinsic errors inherent to the CKKS encryption scheme, and (2) the extra noise added via noise flooding through the default OpenFHE configuration, which is the mechanism used to achieve the stronger $IND - CPA^D$ security against key-recovery attacks.

[8] We do not manually inject additional DP noise into the system; instead, we rely on two inherent features of the CKKS scheme for DP guarantees: the default OpenFHE configuration for noise flooding and the intrinsic noise generated during CKKS operations. Alternatively, for FHE schemes like BGV or BFV, achieving DP would typically involve adding extra noise directly to the model updates before they are encrypted.

 doi:10.20944/preprints202512.1579.v1

3. Security Analysis

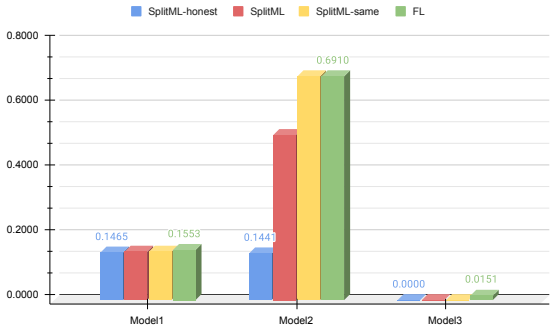
3.1. Model Poisoning Attacks

Byzantine problems occur when some clients are compromised and do not compute or upload weights correctly. As shown in [7], when the average function is used for aggregation, a Byzantine attacker can take over and lead training to an incorrect phase. Model poisoning manipulates the local model to inject backdoors or wrong inputs into the global shared model.

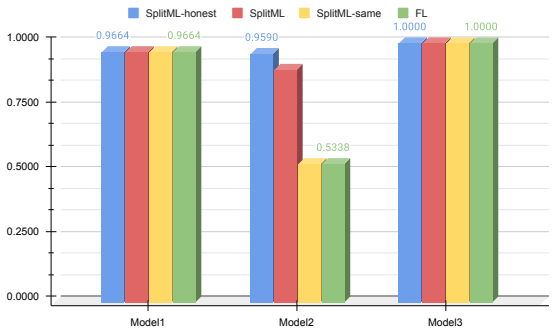
SplitML benefits from split learning in the absence of a global model. An adversary may only influence the top (shared) layers for feature extraction and can not significantly impact the outcome of an honest client model, as the bottom (personalized) layers will compensate for the propagated errors of shared layers. The local model will be trained for multiple rounds (epochs) until the desired accuracy is achieved. In the following, we discuss different experiments and analyze the robustness of our scheme against model poisoning attacks with different settings.

Table 2. Accuracy metrics for clients during training under varying settings. M-1,2,3 refers to model-1,2,3; TA is Training Accuracy; TL is Training Loss; VA is Validation Accuracy; VL is Validation Loss. S1 refers to heterogeneous models of **SplitML** in honest setting; S2 is heterogeneous models in poisonous setting; S3 is homogeneous models of **SplitML** in poisonous setting, and FL is poisonous setting.

Type	S1			S2			S3			FL		
	M1	M2	M3	M1	M2	M3	M1	M2	M3	M1	M2	M3
TA	0.9655	0.9583	1.0000	0.9655	0.8404	1.0000	0.9655	0.5345	1.0000	0.9655	0.5345	0.9994
TL	0.1465	0.1441	0.0000	0.1501	0.5115	0.0000	0.1501	0.6908	0.0000	0.1553	0.6910	0.0151
VA	0.9664	0.9590	1.0000	0.9664	0.8980	1.0000	0.9664	0.5338	1.0000	0.9664	0.5338	1.0000
VL	0.1433	0.1438	0.0000	0.1473	0.4348	0.0000	0.1472	0.6910	0.0000	0.1472	0.6909	0.0008



(a) Training Loss



(b) Validation Accuracy

Figure 3. Comparing training loss (lower is better) and validation accuracy (higher is better) under model poisoning attack for different configurations (all three clients are honest (no poisoning) and only Client-2 is honest (Model-1,3 sends poisonous updates)) of **SplitML** (honest and malicious) and Federated Learning (malicious).

 doi:10.20944/preprints202512.1579.v1

First, we experimented on three small Neural Network (NN) models on a binary log classification problem. All three clients collaborate to update weights for the first layer of 4 neurons with ReLU activation. Clients have the last (output) layer with two neurons and $Softmax$ activation in common, where one neuron corresponds to the 'normal' and another to the 'anomaly' class. Model-1 has a hidden layer of 2 neurons with $ReLU$ activation, model-2 does not have any hidden layers, and model-3 has two hidden layers with two neurons each; the prior hidden layer has $ReLU$, and the later hidden layer has $Sigmoid$ activation. We have used modified Loghub [34] HDFS_1 labeled data from Logpai; refer to §4.1 for details.

We experimented with different configurations of **SplitML** and measured the trained model's accuracy (Figure 3a,3b, and Table 2). In the first scenario, S1, all three clients have heterogeneous models, as described earlier. They perform honestly and collaborate on a shared input layer with four neurons. We achieved 96.64% validation accuracy for model-1, 95.90% accuracy for model-2, and 100% accuracy for model-3. We use this as a benchmark and compare model accuracy for each client in malicious settings S2 and S3, where clients send poisonous weights instead of correct layer weights. In S2 (**SplitML** in adversarial setting with heterogenous models) and S3 (**SplitML** in malicious setting with homogeneous models), only model-2 is honest, and the majority of clients (model-1 and model-3) are poisonous. We chose a considerable (poisonous) update value of 999 for the experiments compared to values in the $[-2, 2]$ in an honest setting. S2 achieved a remarkable 89.80% (only dropped $\approx$ 6%) accuracy with a malicious majority.

In S3, we repeat the malicious majority setting of S2, with all three clients having the same 4-layer architecture as model-3. We observed similar accuracy levels for poisonous clients 1 and 3, honest client 2 suffered heavily while only achieving 53.38% (over 40% loss) accuracy, close to random guessing. We repeated this experiment in an FL setting, where clients collaborate on all layers instead of the top layers in S3. We observed similar accuracy metrics for FL as S3.

Table 3. Accuracy metrics for five clients in a training phase. S1 refers to heterogeneous models in honest **SplitML** setting, S2 is heterogeneous models in poisonous **SplitML** setting, S3 is homogeneous models in poisonous **SplitML**, and FL is performed under the poisonous setting.

Arch	Training Loss					Validation Accuracy				
	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5
S1	0.1352	0.1564	0.2390	0.0000	0.0000	0.9691	0.9623	0.9163	1.0000	1.0000
S2	0.1378	0.1568	5241.3701	0.0000	0.0000	0.9691	0.9623	0.5658	1.0000	1.0000
S3	0.1378	0.1568	0.6839	0.0000	0.0000	0.9691	0.9623	0.5658	1.0000	1.0000
FL	8.48E+15	1.03E+16	1.19E+17	3.20E+17	3.43E+17	0.9691	0.9623	0.5658	0.0000	0.0000

We validated the results by repeating these experiments with five models. All five clients collaborate to train the first 2 (top) layers. The input layer has five neurons, and the second layer has four neurons; both layers use $ReLU$ activation. The output layer is shared with a single neuron and $Sigmoid$ activation. Model-1 has a third (hidden) layer before the output layer, with two neurons and $ReLU$ activation. Similarly, Model-2 has a hidden layer with four neurons, and Model-4 has three neurons; both use $ReLU$ activation. Model-3 does not have any additional layers. Model-5 has two layers, both with two neurons and $ReLU$ activation.

In an honest setting, all five client models, M1 to M5, send correct updates. Under the malicious setting, only clients 2 and 4 are honest, and a majority (3 out of 5) models 1,3, and 5 send poisonous updates (vector of 999 instead of honest values in the range $[-4, 4]$). Since FL architecture collaborates on all the layers, we observed a poor performance (Table 3) as expected with very high losses. In FL, only models 1 and 2 managed to get 96% accuracy, while model 3 observed 56% accuracy, where its performance dropped by 35% compared to the honest setting. **Moreover, models 4 and 5 in FL reported 0% accuracy**.

FL leaks information (under malicious setting) as the reported validation accuracy for these five models corresponds to their data distribution. Since the poisonous updates classifies everything as anomalous, validation accuracy is inversely proportional to the dataset size with normal class. Model-1

 doi:10.20944/preprints202512.1579.v1

had 3.08% normal samples, Model-2 had 3.67%, Model-3 had 43.23%, and Model-4 and Model-5 had 100% normal samples (thus 0% accuracy).

SplitML performed exceptionally well, with the same model accuracy in a poisonous setting as honest. Only model-3 performed poorly with 56% accuracy, as it has an output layer right after the two shared layers and did not have any additional hidden layers to compensate for the propagated poisonous values and adjust the weights in the deep hidden layers. While we presented empirical evidence of inherent robustness to poisoning attacks due to our architecture, we do not offer active mitigations to prevent poisonous updates (refer Appendix §C.1), as measures based on the similarity of model updates would not be helpful if the majority of the clients are malicious.

3.2. Inference Attacks

The following subsections discuss membership inference and model inversion attacks in detail.

3.2.1. **Membership Inference**

A Membership Inference (MI) attack [35] is to determine whether a data record is contained in a client's training dataset. A privacy breach is incurred when information (e.g., model weights) is shared between servers. While an MI black-box attack with shadow models (Figure 4) is performed with the help of original labels retrieved/queried for the datasets on a target model, in our case, due to the splitting of a client's local model, we develop attack models both for the (i) top layers and (ii) bottom layers from a split:

1. First, we attack (input) datasets and their gradients from the split layer to determine their membership.
2. We develop another attack model to infer membership from labels or predictions given the gradients from the cut layer.

We attacked a CNN model similar to [36] on the MNIST [37] dataset. The target model has 4 CNN and 2 ANN layers, with the first convolution layer with 3x3 kernel and $ReLU$ activation. It is followed by a max pooling layer with a 2x2 kernel. The third convolution layer has a 3x3 kernel with a $ReLU$ activation followed by a fourth max pooling layer of 2x2. The fifth layer is a dense layer with 128 neurons with $ReLU$ activation, followed by a 10-neuron output layer with $Softmax$ activation. We refer to this full model with six layers as architecture A (Figure 5).

Architecture-B is this full model A's first 4 CNN layers (top split), and C is the last 2 ANN layers (bottom split). We then create more models, keeping the top layers (as in A) the same and measuring MI attack accuracy with shadow models on different bottom layers after the split. Architecture-D has 3 ANN layers: 128 neurons with $ReLU$, 64 with $ReLU$, and 10 with $Softmax$. E has the same number of (ANN) layers (2) as in A with 64 neurons instead of 128 in the first dense layer, keeping the same output layer. F has 2 layers as in A but 256 neurons rather than 128. In G, we remove the 128-neuron layer and keep the output layer. Finally, H has 128 neurons with $TanH$ and 10 with $Sigmoid$ for the output layer. Due to the intrinsics of the shadow models, the developed attack model will have good accuracy either on MI or non-MI. An attacker may develop two attack models complementing each other for high-confidence results. Further, We empirically observed (Table 4) that the top layers, on average, leak more information than the bottom layers for MI. However, on average, splitting helps reduce attack accuracy on partial models compared to performing this attack on a complete model. Though accuracy was observed around 50% for average cases, which is analogous to random guessing, a well-trained model can improve attack performance. While using a Laplacian noise for DP is expected, we suggest adding extra Gaussian noise for encryption to reduce the attack accuracy and achieve $IND - CPA^D$ security for approximate FHE. Scaling parameter Δ can be adjusted for better precision for this extra noise.

 doi:10.20944/preprints202512.1579.v1

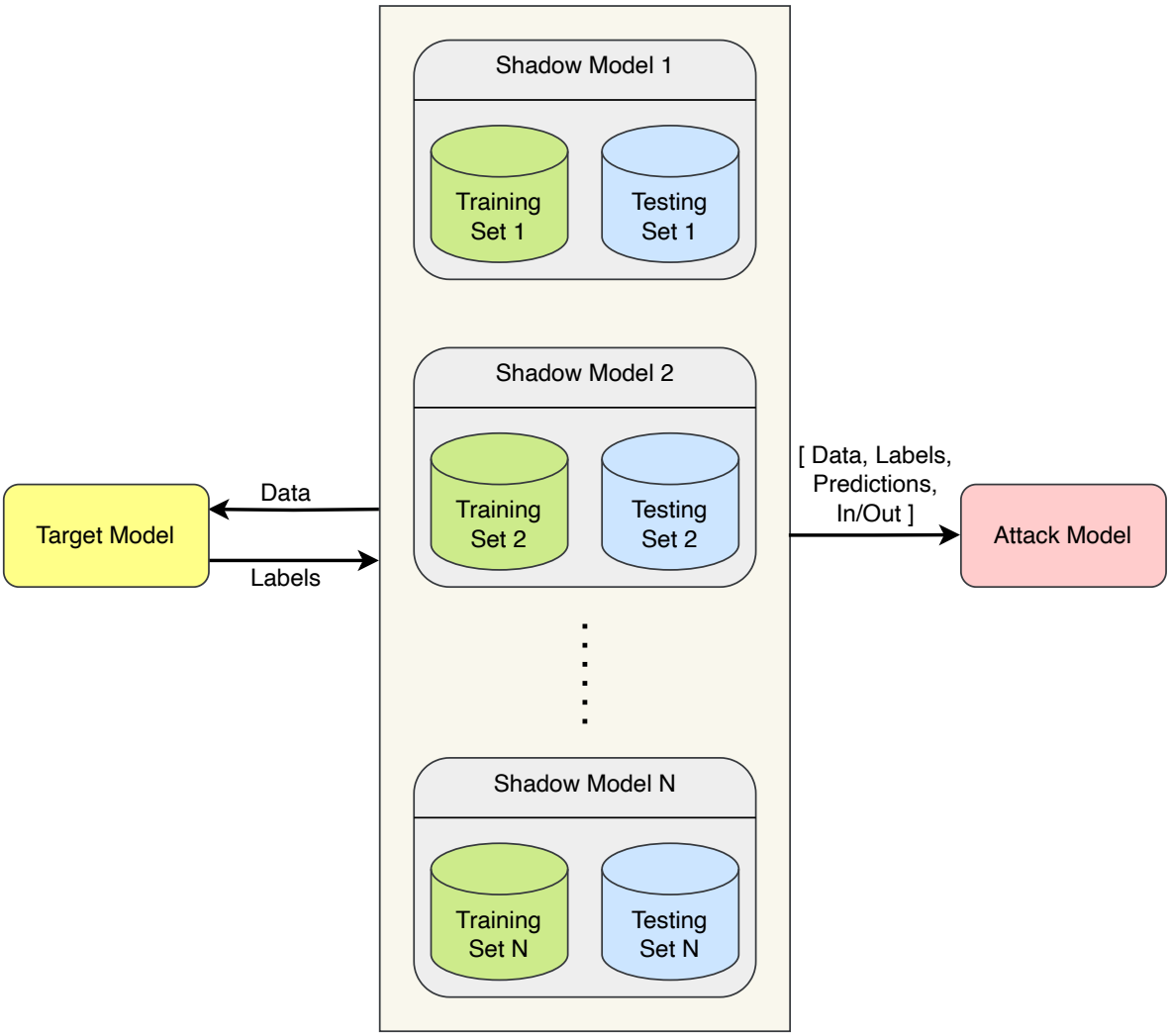
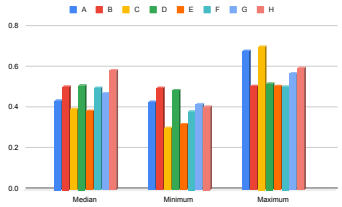
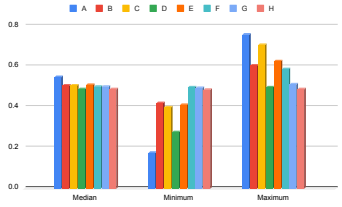


Figure 4. Membership Inference (MI) attack using black-box shadow models deploys multiple shadows with synthetic or captured data for membership (Training set) and non-membership (Testing set) to train the attack model, which can either detect membership (In) or non-membership (Out) of a record with high accuracy.

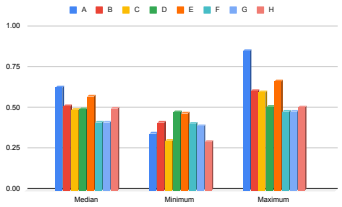
 doi:10.20944/preprints202512.1579.v1



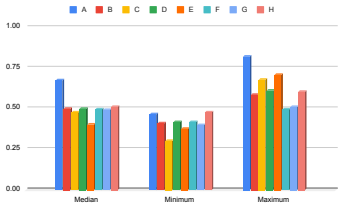
(a) Member-accuracy with one shadow



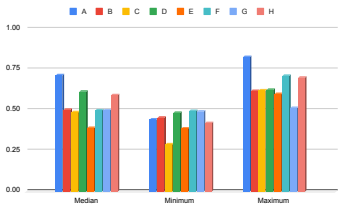
(b) Member-accuracy with two shadows



(c) Member-accuracy with three shadows



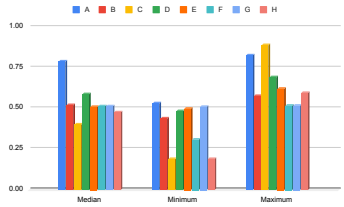
(d) Member-accuracy with four shadows



(e) Member-accuracy with five shadows

Figure 5. *Cont.*

 doi:10.20944/preprints202512.1579.v1



(f) Member-accuracy with ten shadows

Figure 5. Member-Accuracy of Membership Inference attack for different shadow model sizes on different architectures. A = Full model without split, B = Top (common) layers before the split, C = Bottom layers after the split, D = More split layers, E = The same number of split layers but fewer neurons, F = Same layers but more neurons, G = Less number of split layers and H = The Same number of bottom split layers but different activation functions. Attack accuracy on a full model (all layers) was as high as 80%, while splitting the layers reduced the attack accuracy to 0.5 or 50%, equivalent to random guessing.

Table 4. Member-accuracy for different shadow models.

Shadows	Architecture	Median	Minimum	Maximum
1	Full model	0.4374	0.4302	0.6816
	Top layers	0.5072	0.5006	0.5100
	Bottom layers	0.3970	0.3046	0.7036
2	Full model	0.5464	0.1764	0.7552
	Top layers	0.5064	0.4186	0.6028
	Bottom layers	0.5072	0.3994	0.7042
3	Full model	0.6306	0.3482	0.8566
	Top layers	0.5170	0.4154	0.6088
	Bottom layers	0.4958	0.3030	0.6030
4	Full model	0.6750	0.4644	0.8170
	Top layers	0.4994	0.4080	0.5838
	Bottom layers	0.4778	0.3004	0.6784
5	Full model	0.7130	0.4420	0.8260
	Top layers	0.5015	0.4542	0.6166
	Bottom layers	0.4881	0.2892	0.6196
6	Full model	0.7610	0.4730	0.8984
	Top layers	0.5058	0.4352	0.5674
	Bottom layers	0.3934	0.3792	0.5972
7	Full model	0.7572	0.7324	0.8428
	Top layers	0.4766	0.4566	0.5402
	Bottom layers	0.5054	0.5016	0.5856
8	Full model	0.7546	0.5564	0.7862
	Top layers	0.5242	0.3714	0.5594
	Bottom layers	0.4052	0.3768	0.8056
9	Full model	0.8356	0.8176	0.8530
	Top layers	0.4482	0.4374	0.5168
	Bottom layers	0.5008	0.3904	0.6004
10	Full model	0.7886	0.5324	0.8270
	Top layers	0.5206	0.4376	0.5776
	Bottom layers	0.4020	0.1910	0.8900

3.2.2. **Model Inversion**

Model inversion uses the output of a model applied to a hidden input to infer certain features of this input. In [38], model inversion is used to construct an input that produces output 1 for class i and 0 for the rest for face recognition. This input is not an actual member of the training dataset but simply an average of the features that “characterize" the class. The results are semantically meaningless and not recognizable as any specific image from the training dataset. Critically, model inversion does not produce any particular image from the training dataset, which defines Membership Inference (MI).

 doi:10.20944/preprints202512.1579.v1

In summary, model inversion produces the average of the features that, at best, can characterize an entire output class. In log anomaly detection, inverting a model is impractical because an “average" representation of a text log entry is not as semantically accurate as an average representation of a class in a facial recognition task. Moreover, due to multi-key FHE, such attacks are only feasible in a colluding (targeted) setting. Finally, we apply Gaussian noise during encryption, which enhances user privacy. Refer §C for existing solutions against such attacks.

3.3. Model Extraction Attacks

Model Extraction (ME) attacks target the confidentiality of ML models [29]. The adversary aims to obtain a stolen replica that performs similarly to the victim while making a few labeling queries. FL focuses on protecting clients' data but is highly vulnerable to Intellectual Property (IP) threats, whereas SL prevents model leakage by design. The model is split in SL, so IP threat due to directly downloading the model is non-existent. Similarly, in **SplitML**, the clients (attackers) do not have access to the entire models of their peers (victims). Hence, they cannot download their models. A successful ME attack breaches the model IP and also makes the model more vulnerable. ME attack can support transferable adversarial attacks [39], mainly targeted ones [40] against the victim model. A high-fidelity surrogate model can also perform bit-flip attacks [41]. Li et al. [42] expose the vulnerability of SL and show how malicious clients can launch ME attacks by querying the gradient information from the server side. They propose five variants of ME attack, which differ in the gradient usage and the data assumptions. In **SplitML**, peers do not have access to gradients but only the weights of shared layers during training.

While SL is better than FL regarding IP protection, it is still vulnerable. Jagielski et al. [43] shows that a high fidelity and accurate model can be obtained with few model prediction queries. ME is relevant in **SplitML** for inference where an adversary sends abundant queries to a target and observes the results for each crafted input. Potential countermeasures restrict or modify information returned in each query [44]. For example, returning the full vector of probabilities reveals much information. The defender may thus choose to return a variant whose numerical precision is lower or even to produce only the most likely label with or without the associated output probability. The defender could also return a random label and noise. Hence, we endorse collaborating on labels (TL) rather than prediction scores (TP) to reduce information leakage.

4. Experimental Analysis

4.1. Dataset

Generally, log anomaly datasets are skewed and dominated by either ‘normal’ or ‘anomalous’ samples. Hence, we use a balanced dataset to mitigate the problem of achieving ‘pseudo-high’ accuracy of the ML model. To demonstrate the balance of classes in our dataset (Table 5), we used a ‘Return-1 Model’ to always classify the data as ‘anomalous.’ As a result, we observed 49.99% accuracy and a recall of 100%, as the model always returns label-1[9] for an anomaly. We have used Loghub HDFS_1 [34] labeled data from Logpai, which is 1.47 GB of HDFS log data set generated through running Hadoop-based map-reduce jobs on more than 200 Amazon's EC2 nodes for 38.7 hours and labeled by Hadoop domain experts. Among 11,175,629 log entries collected, 2.58% (288,250) data is anomalous. We have used Drain [45] log parser[10] to transform our unstructured log data to a structured format.

We created a smaller and balanced dataset of 576,499 inputs with seven features uniformly distributed among ‘normal’ and ‘anomaly’ classes. We further partition the data as a non-overlapping i.i.d. dataset of 397,366 observations split across three clients. Client-1 had 69,455 normal and 66,780 anomaly samples, Client-2 had 93,032 normal and 73,055 anomaly samples, and Client-3 had 43,666 normal and 51,378 anomaly samples. On each client, we use 20% of data for testing and 80% for training.

[9] We denote normal class as label-0 and anomalous with label-1.
[10] We omit the details of textual log data parsing for brevity.

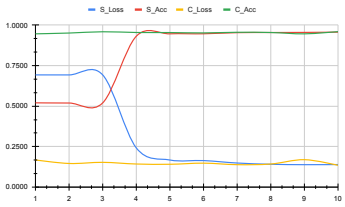
 doi:10.20944/preprints202512.1579.v1

Table 5. Return-1 model shows our dataset's balanced distribution of two classes.

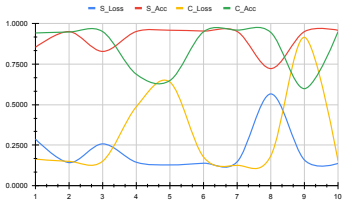
Type	Accuracy	Precision	Recall	F1-Score
Full (100%)	0.4999	0.4999	1.0000	0.6666
Test (20%)	0.5016	0.5016	1.0000	0.6681

4.2. Results

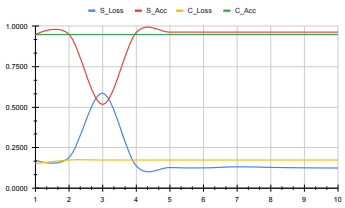
We present experimental results about ML attacks in §3. The computations were performed on a MacBook Pro with a 2.4 GHz Quad-Core Intel Core i5 processor and 8 GB 2133 MHz LPDDR3 memory. We used Python 3.11 [46] with sklearn APIs [47] for binary classifiers. We compared the performance based on the following measures: Precision, Recall, Accuracy, and F1-Score. We created three small NNs, depicting three clients (Client-1, 2, 3) participating in our scheme for both training and inference.



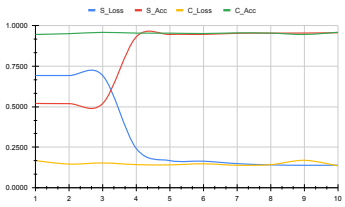
(a) Model-1 with $lr = 0.05$



(b) Model-2 with $lr = 0.05$



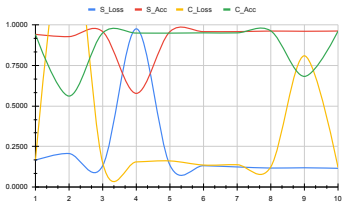
(c) Model-3 with $lr = 0.05$



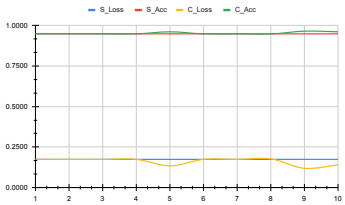
(d) Model-1 with $lr = 0.10$

Figure 6. *Cont.*

 doi:10.20944/preprints202512.1579.v1



(e) Model-2 with $lr = 0.10$



(f) Model-3 with $lr = 0.10$

Figure 6. Accuracy and loss (in standalone and collaborative settings) for training epochs with learning rate $lr \in \{0.05, 0.10\}$.

Table 6. Consensus results with $lr = 0.10$.

Type	Accuracy	Precision	Recall	F1-Score
Model-1	0.8770	0.8749	1.0000	0.9333
Model-2	0.7366	0.9319	0.7485	0.8302
Model-3	0.8604	0.8604	1.0000	0.9249
TL	0.7366	0.9319	0.7485	0.8302
TP	0.8877	0.8845	1.0000	0.9387

The models share the first two Fully Connected (FC) layers, with the first layer having five neurons, 40 parameters, and *ReLU* activation and the second layer having four neurons, 24 parameters, and *ReLU* activation. For simplicity, all the models have a common output layer with a single neuron and *Sigmoid* activation function. Model-1 has a third layer with two neurons, ten parameters, and *ReLU* activation. Model-2 only has the three layers described earlier. In contrast, Model-3 has two additional FC layers, a third layer with three neurons, 15 parameters, and *ReLU* activation, and a fourth layer with two neurons, eight parameters, and *ReLU* activation.

All the clients train their models with plaintext data and only encrypt the weights to the FL server after one round is complete. For each round, all clients train their models for one epoch in parallel. In the first round, clients initialize their model weights with random values, and for the subsequent rounds, the weights for the shared layers are set to the values provided by the server. The server calculates the results in each round using an FL algorithm (like FedAvg) in the encrypted domain and the homomorphic EK. Empirically, we observed that we reduced the epochs required to converge the model on average by half. As shown in Figure 6, for a batch size of 64 and learning rates of 0.05 and 0.10, the standalone training (S_Acc, S_Loss denotes Standalone Accuracy/Loss) required six epochs to converge whereas collaborative learning (C_Acc, C_Loss denotes Collaborative Accuracy and Loss) achieved higher accuracy in only three epochs.

We further experimented with inference using the earlier trained models. For *Sigmoid* activation function $f(z) \in [0, 1]$, we set boundary threshold $\lambda = 0.05$. We performed prediction consensus using both the predicted label TL (§2.5.1) and prediction value TP (§2.5.2) approach for the samples that fall in the prediction range of $[0.45, 0.55]$. We set $TL \geq 2$ and $TP \geq 1.5$ to choose label-1 for three clients in a consensus.

 doi:10.20944/preprints202512.1579.v1

We observed that TL performed better when most samples were classified as normal (prediction scores close to 0), and TP worked better when most were anomalous (scores close to 1). For example, in an observation for $lr = 0.05$ with 88 normal and 0 anomaly samples, Model-1 achieved 18.18%, Model-2 achieved 73.86%, Model-3 achieved 00.00%, TL achieved 75.00%, and TP achieved 31.82% accuracy. TL outperformed all models. In another observation for $lr = 0.10$ (Table 6) with 353 normal and 2175 anomalies, Model-1 had 87.70%, Model-2 had 73.66%, Model-3 had 86.04%, TL had 73.66%, and TP recorded the highest of 88.77% accuracy.

In practice, a client may want to perform a consensus for the predictions most likely to be False Positives (FP) and False Negatives (FN). From our experiments, we observed that for our three clients with *Sigmoid* output, all the false predictions were from $[0.00, 0.10]$ and $[0.90, 1.00]$. For one observation for $lr = 0.05$ with 1388 FN and 23663 FP collected from all three clients, Model-1 achieved 5.54%, Model-2 achieved 41.11%, Model-3 achieved 5.54%, TL achieved 5.54%, and TP achieved 20.80% accuracy. For another observation for $lr = 0.10$ with 3166 FN and 29 FP, Model-1 achieved 0.69%, Model-2 achieved 26.29%, Model-3 achieved 26.26%, TL achieved 32.99% and TP achieved 21.60% accuracy.

Key generation involves interaction between parties and is done before the training process begins. We observed $\sim 5-6$ seconds processing time (zero network communication overhead, as all the clients were simulated locally on a single machine) to generate all private and shared keys. Federation overhead was $\sim 1.2-1.5$ seconds of total training time of $\sim 25-30$ seconds per epoch.

5. Conclusion

This paper presents the **SplitML** framework, a unified approach that builds upon secure and trusted learning to tackle privacy-utility concerns in real-world distributed model training. **SplitML** merges the strengths of FL and SL, enabling personalized models while safeguarding client data and thwarting inference attacks. It leverages multi-key Fully Homomorphic Encryption (FHE) with minimal overhead for secure communication and the protection of sensitive information. Additionally, it incorporates a counseling process allowing encrypted queries and aggregated classification results without compromising privacy. While addressing typical ML attacks through enhanced security and privacy in a collaborative setting, future work will incorporate fairness considerations into the architecture. The current threat model assumes an honest-but-curious server (passively curious, not actively malicious), which may potentially collaborate with malicious clients. A natural next step involves extending the threat model to include an untrusted server by deploying signature verification for clients, distributed servers, or a peer-to-peer scheme, thereby eliminating reliance on a single, central, trusted server.

Author Contributions: For research articles with several authors, a short paragraph specifying their individual contributions must be provided. The following statements should be used "Conceptualization, X.X. and Y.Y.; methodology, X.X.; software, X.X.; validation, X.X., Y.Y. and Z.Z.; formal analysis, X.X.; investigation, X.X.; resources, X.X.; data curation, X.X.; writing—original draft preparation, X.X.; writing—review and editing, X.X.; visualization, X.X.; supervision, X.X.; project administration, X.X.; funding acquisition, Y.Y. All authors have read and agreed to the published version of the manuscript.", please turn to the CRediT taxonomy for the term explanation. Authorship must be limited to those who have contributed substantially to the work reported.

Funding: Please add: "This research received no external funding" or "This research was funded by NAME OF FUNDER grant number XXX." and and "The APC was funded by XXX". Check carefully that the details given are accurate and use the standard spelling of funding agency names at https://search.crossref.org/funding, any errors may affect your future funding.

Institutional Review Board Statement: In this section, you should add the Institutional Review Board Statement and approval number, if relevant to your study. You might choose to exclude this statement if the study did not require ethical approval. Please note that the Editorial Office might ask you for further information. Please add "The study was conducted in accordance with the Declaration of Helsinki, and approved by the Institutional Review Board (or Ethics Committee) of NAME OF INSTITUTE (protocol code XXX and date of approval)." for

 doi:10.20944/preprints202512.1579.v1

studies involving humans. OR "The animal study protocol was approved by the Institutional Review Board (or Ethics Committee) of NAME OF INSTITUTE (protocol code XXX and date of approval)." for studies involving animals. OR "Ethical review and approval were waived for this study due to REASON (please provide a detailed justification)." OR "Not applicable" for studies not involving humans or animals.

Informed Consent Statement: Any research article describing a study involving humans should contain this statement. Please add "Informed consent was obtained from all subjects involved in the study." OR "Patient consent was waived due to REASON (please provide a detailed justification)." OR "Not applicable" for studies not involving humans. You might also choose to exclude this statement if the study did not involve humans.

Written informed consent for publication must be obtained from participating patients who can be identified (including by the patients themselves). Please state "Written informed consent has been obtained from the patient(s) to publish this paper" if applicable.

Data Availability Statement: We encourage all authors of articles published in MDPI journals to share their research data. In this section, please provide details regarding where data supporting reported results can be found, including links to publicly archived datasets analyzed or generated during the study. Where no new data were created, or where data is unavailable due to privacy or ethical restrictions, a statement is still required. Suggested Data Availability Statements are available in section "MDPI Research Data Policies" at https://www.mdpi.com/ethics.

Acknowledgments: In this section you can acknowledge any support given which is not covered by the author contribution or funding sections. This may include administrative and technical support, or donations in kind (e.g., materials used for experiments). Where GenAI has been used for purposes such as generating text, data, or graphics, or for study design, data collection, analysis, or interpretation of data, please add "During the preparation of this manuscript/study, the author(s) used [tool name, version information] for the purposes of [description of use]. The authors have reviewed and edited the output and take full responsibility for the content of this publication."

Conflicts of Interest: Declare conflicts of interest or state "The authors declare no conflicts of interest." Authors must identify and declare any personal circumstances or interest that may be perceived as inappropriately influencing the representation or interpretation of reported research results. Any role of the funders in the design of the study; in the collection, analyses or interpretation of data; in the writing of the manuscript; or in the decision to publish the results must be declared in this section. If there is no role, please state "The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results".

Appendix A Background

We denote the notations used in this paper in Table A1, then briefly describe Fully Homomorphic Encryption (FHE) and Differential Privacy (DP) in the following subsections.

 doi:10.20944/preprints202512.1579.v1

Table A1. Notations used in this paper.

Symbol	Name
Enc	Encryption
Dec	Decryption
Add	Addition
$Mult$	Multiplication
PK	Public Key
SK	Secret Key
EK_{Add}	Evaluation Key for Addition
EK_{Mult}	Evaluation Key for Multiplication
EK_{Dec}	Evaluation Key for Fused Decryption
Δ	Scaling Factor for FHE
λ	Classification Threshold
$f(z)$	Activation Function f on Input z
S	Central (Federation) Server
K	(Total) Number of Clients
k	Client Index ($k \in \{1, \ldots, K\}$)
T	(Threshold) Number of Clients required for Fused Decryption ($T \leq K$)
T'	Number of Colluding Clients
C_k	k-th Client
D_k	Dataset of k-th Client
o	Observation (Record) in a Dataset ($o \in D_k$)
A	Shared Attributes (Features)
L	Shared Labels
n	Number of Shared Layers
t_k	Number of Personalized Layers of k-th Client
M_k	ML model of k-th Client
$\|M_k\|$	Number of Total Layers of k-th Client ($\|M_k\| = n + t_k$)
q	Cut (Split) Layer
$\|q\|$	Size of the Cut Layer
∇q	Gradient from Cut Layer q
$\tilde{\mathbf{N}}_{\tilde{\mathbf{d}}}$	Number of Decryption Queries
m	Number of Participants in Training
p	Training Participant Index ($p \in \{1, \ldots, m\}$)
P_p	p-th Participant
m'	Number of Participants in Consensus
h	Consensus Participant Index ($h \in \{1, \ldots, m'\}$)
R	Number of Training Rounds
r	Round Index ($r \in \{1, \ldots, R\}$)
α	Fraction of ML parameters with a Client
$1 - \alpha$	Fraction of ML parameters with the Server
bs	Batch Size
lr	Learning Rate

Appendix A.1 Fully Homomorphic Encryption (FHE)

FHE is a cryptographic primitive that can perform arithmetic computations directly on encrypted data. This makes FHE a preferred candidate for privacy-preserving computation and storage [9,10]. FHE has received significant attention worldwide, which yielded many improvements since Gentry's scheme in 2009 [8]. As a result, FHE is used in many applications [11–17]. FHE can be classified as word-wise [48–51] and bit-wise [52,53] schemes as per the supported operations.

FHE allows arbitrary computations to be performed on encrypted data without decrypting by using three keys: the public key (PK), the secret key (SK), and the evaluation key (EK). The public key

 doi:10.20944/preprints202512.1579.v1

can be used to encrypt data. The secret key can be used to decrypt data. The evaluation key can be used to evaluate circuits on encrypted data. It is typically generated from the secret key but can also be generated from the public and secret keys together.

This work utilizes the CKKS [18] scheme as a FHE scheme. CKKS varies from other FHE schemes (such as BFV [54,55], BGV [56], and TFHE [57]) in the way in which it interprets encryption noise. Indeed, CKKS treats encryption noise as part of the message, similar to how floating-point arithmetic approximates real numbers. This means the encryption noise does not eliminate the Most Significant Bits (MSBs) of the plaintext $\tilde{\mathbf{m}}$ as long as it stays small enough. CKKS decrypts the encryption of message $\tilde{\mathbf{m}}$ as an approximated value $\tilde{\mathbf{m}} + \tilde{\mathbf{e}}$, where $\tilde{\mathbf{e}}$ is a slight noise. The authors of CKKS suggested multiplying plaintexts by a scaling factor Δ before encryption to lessen precision loss after adding noise during encryption. CKKS also sustains batching, a process for encoding many plaintexts within a single ciphertext in a Single Instruction Multiple Data (SIMD) fashion. CKKS is a set of probabilistic polynomial-time algorithms regarding the security parameter. The algorithms are:

- *CKKS.KeyGen*: generates a key pair.
- *CKKS.Enc*: encrypts a plaintext.
- *CKKS.Dec*: decrypts a ciphertext.
- *CKKS.Eval*: evaluates an arithmetic operation on ciphertexts (encrypted data).

Multi-key CKKS is a tuple of five probabilistic polynomial-time algorithms. Given a security parameter μ, the maximal multiplicative depth of evaluatable circuits $\tilde{L}$, the number of clients K, and access structure $\tilde{A}$, *CKKS.KeyGen* returns a public key PK and K secret keys SK_1 to SK_K and evaluation key EK. Given a public key PK and a message $\tilde{m}$, *CKKS.Enc* returns a ciphertext $\tilde{c}$. Given a secret key SK_k and a ciphertext $\tilde{c}$, *CKKS.PDec* returns a partial decryption ψ_k. Given an evaluation key EK, a circuit f, and ciphertexts $\tilde{c}_1$ to $\tilde{c}_k$, *CKKS.Eval* returns a ciphertext $\tilde{c}_f$. Given a set of partial decryptions $\{\psi_k\}$ where $k \in \tilde{A}$, and a ciphertext $\tilde{c}$, *CKKS.Combine* returns a message $\tilde{m}'$ approximate of $\tilde{m}$.

- $PK, \{SK_1, \ldots, SK_K\}, EK \leftarrow CKKS.KeyGen(1^\mu, \tilde{L}, K, \tilde{A})$
- $\tilde{c} \leftarrow CKKS.Enc(\tilde{m}, PK)$
- $\psi_k \leftarrow CKKS.PDec(\tilde{c}, SK_k)$
- $\tilde{c}_f \leftarrow CKKS.Eval(f, EK, \tilde{c}_1, \ldots, \tilde{c}_K)$
- $\tilde{m}' \simeq \tilde{m} \leftarrow CKKS.Combine(\{\psi_k\}_{k \in \tilde{A}}, \tilde{c})$

Appendix A.2 Differential Privacy (DP)

DP [58] can guarantee privacy by allowing data to be analyzed without revealing sensitive information about any item in the dataset. This is done by adding random noise to the data, making distinguishing between the original and noisy data impossible.

The amount of noise added to the data is determined by a parameter called ϵ, known as the privacy budget. The higher the value of ϵ, the more noise is added to the data and the stronger the privacy guarantee. A mechanism $\mathcal{M}$ is considered (ϵ, δ)-DP if, $\forall$ adjacent datasets, x and y, and $\forall$ possible subsets of results, $\mathcal{R}$ of the mechanism, the following holds:

$$\mathbb{P}[\mathcal{M}(x) \in \mathcal{R}] \leq e^\epsilon * \mathbb{P}[\mathcal{M}(y) \in \mathcal{R}] + \delta.$$

DP adds random noise to the data, obscuring private information. The amount of noise can be carefully specified to achieve DP [58,59] and its variants [60] for protecting privacy, in particular, of ML training data [61]. DP learning has been applied to regressions [62,63], Support Vector Machines (SVM) [64], Decision Trees (DT) [65], and Neural Networks (NN) [66]. A client may add noise to their input data [67] or the local updates before sending them to the federated server [68]. The noise can also be added on the server side [69]. A probability bound on the information leakage can be computed based on the amount of noise added, indicating how likely an adversary can extract private information from the data. The goal is to prevent an attacker from extracting private information from the trained model by controlling how much each sample can influence the parameters during training [70].

 doi:10.20944/preprints202512.1579.v1

Multiple parties can also use it to train a shared model on distributed data without disclosing private data to other participants [71]. Bad actors may collude in a collaborative learning process in an honest-but-curious model. The server may not be trusted to generate the noise to protect the data by DP because it may communicate it to clients, thus annihilating the DP guarantees. In that case, a common practice is to make the participants generate the noise in a distributed way [72,73]. This is especially practical for the resulting noise to follow a Gaussian distribution since this distribution is stable by addition.

Dong et al. [74] experimented with Gaussian and Laplacian noises and observed that the trade-off between utility and defense mainly depends on the magnitude of noise levels and is less related to the noise types. In addition, both types of noises will introduce a non-negligible accuracy drop for the target model.

Titcombe et al. [75] introduced a noise defense in which additive Laplacian noise is applied to the intermediate data representation on the data owners' side before sending it to the computational server. This obscures the data communicated between model segments and makes it harder for the attacker to learn the mapping from the intermediate representation to the input data. The noise defense can be applied unilaterally by the data holder. This is useful when a data holder does not trust the computational server.

Appendix B Related Work

In this section, we present formal definitions for Federated Learning (Definition A1) and Split Learning (Definition A2) and discuss their inherent vulnerabilities.

Appendix B.1 Federated Learning

Federated Learning (FL) developed by Google [3] distributively trains ML models in devices having privacy-sensitive local training samples to solve the data islanding problem. At the starting round $r = 0$ of the FL training, central (federation) server S initializes a global model M_0 and sends it to m participants (out of k clients) of the current round of FL. After receiving the initial model M_0, each participant starts on-device training and updates the model using local samples. Then, each participant returns the updated model M'_{p_0} to the server. The server aggregates all the received models to generate the updated version M_1 of the global model. These rounds of computation-communication continue ($r = R$) until the server acknowledges the global model M_R to be converged. FL can be classified [76] into three types:

1. Horizontal Federated Learning (HFL), where organizations share partial features.
2. Vertical Federated Learning (VFL), where organizations share partial samples.
3. Federated Transfer Learning (FTL), where neither samples nor features have much in common.

Definition A1. *FL is a privacy-preserving collaborative distributed learning scheme for training a globally shared ML model M with $K > 1$ clients and an FL server S. For each training round $r \in \{1, \ldots, R\}$, participants $1 < m \leq K$ train all the $n > 0$ shared layers of M_{r-1} with their private dataset $D_k, k \in \{1, \ldots, K\}$ with common attributes A and common labels L. After each training round, S averages the weights of M'_{r-1} from all participants with some averaging algorithm and sends the updated model weights M_r to participants of the next round. Training continues till M converges or fixed rounds R.*

The main disadvantage of FL is that each client needs to run the full ML model, and resource-constrained clients, such as those available in the Internet of Things (IoT) devices, can only afford to run part of the model. In **SplitML** clients runs full models locally, but communication cost is lower, as weights for only shared layers are sent. Also, the central server requires less computation in **SplitML** to average the weights of shared layers compared to all layers in FL. Hence, **SplitML** can help lower the communication costs on clients and computation costs on the server.

 doi:10.20944/preprints202512.1579.v1

Model privacy vanishes in FL for others if one of the clients is compromised. The heterogeneity of models in **SplitML** can help protect privacy even though some clients are compromised.

FL is also vulnerable to Inference attacks. Truex et al. [77] proposed a feasible black-box membership inference attack in FL. Zhu et al. [68] proposed a deep leakage method to retrieve training data from publicly shared gradients on computer vision and Natural Language Processing (NLP) tasks. Wang et al. [78] used a Generative Adversarial Network (GAN) based method called Multi-task GAN in FL to precisely recover the private data from a specific client, which causes user-level privacy leakage. We provide empirical evidence to show that **SplitML** offers robust protection against these attacks. The heterogeneity of data and models and the fact that in **SplitML**, only top layers weights are shared provides extensive protection against inference.

In an FL setting [79–81], the heterogeneity of client data makes the (Model) Poisoning attacks easier and detection harder. A Byzantine adversary launches a Model Poisoning attack, manipulating an arbitrary proportion of malicious users to deviate from a correct trend by submitting poisonous local model updates. The adversarial objective of malicious users is to cause the federated model to yield attacker-chosen target labels for specific samples (targeted attacks [79,80]), or to misclassify all testing samples indiscriminately (untargeted attacks [81]). Privacy-Preserving Federated Learning (PPFL) is vulnerable to Model Poisoning attacks launched by a Byzantine adversary, who crafts malicious local gradients to harm the accuracy of the federated model. To mitigate these attacks, the central server must distinguish the information uploaded by honest clients from malicious ones. Our experiments show that **SplitML** by-design protects against malicious clients performing Model Poisoning attacks to lower the accuracy of honest clients, even when the majority are sending malicious updates.

HeteroFL [82] introduces a novel framework to address the challenge of possibly heterogeneous clients such as mobile phones and IoT devices equipped with different computation and communication capabilities in FL. HeteroFL tackles inefficiencies by allowing clients to train local models with varying complexity levels. HeteroFL proposes to allocate subsets of global model parameters adaptively according to the corresponding capabilities of local clients. Clients with higher computational capabilities can train more complex models, while those with limited resources can use simpler models. Despite these differences, all models belong to the same model class. This approach departs from traditional FL, where local models typically share the same architecture as the global model.

Helios [83] is a heterogeneity-aware framework to address the straggler (devices with weak computational capacities) issue in FL. It identifies the different training capabilities of individual devices and assigns them appropriate workloads. Helios proposes a ”soft-training" method that dynamically compresses the model training workload for stragglers. This is achieved through a rotating neuron training approach, where only a subset of the model's neurons are trained at each step. Helios aims to accelerate the training of stragglers while maintaining the accuracy and convergence of the overall FL process.

Heterogenous FL [82,83] enables the training of heterogeneous local models while producing a shared global inference model. While these frameworks consider heterogeneous clients in terms of computational resources to collaboratively train a shared global model, **SplitML** considers heterogeneity in terms of model architecture itself. Unlike the model partitions calculated by algorithms in these approaches, clients in **SplitML** choose their own local models and don't have a shared global model.

Appendix B.2 Split Learning

In the (vanilla) Split Learning (SL) algorithm [4], from one specific layer, called the 'split layer' or 'cut layer,' the Neural Network (NN) is split into two sub-networks. The client performs forward propagation on local training data and computes the output of its sub-network up to the cut layer, which is sent to the server to compute the final output until the last layer of the network. At the server's sub-network, the gradients are backpropagated from the last layer to the split layer, and the gradient of the split layer is sent back to the client. The client performs the rest of the backward

 doi:10.20944/preprints202512.1579.v1

propagation process from the split to the first layer of the network. This process continues until the client has new training data. The server has no direct access to clients' raw data, and complete model parameters are not sent to the server. The only information being communicated is the output of the cut layer from clients to the server and the cut layer gradient from the server to clients. Compared with other approaches, such as FL, SL requires consistently fewer resources from the participating clients, enabling lightweight and scalable distributed training solutions.

Definition A2. *(Vanilla/Horizontal) SL is a privacy-preserving collaborative distributed learning scheme for training a globally shared ML model M with $K > 1$ clients and an FL server S. For each training round $r \in \{1, \ldots, R\}$, a participant $P_p, p \in \{1, \ldots, m\}, 1 < m \leq K$ trains the first $n > 0$ shared layers ($|M| = n + t, t > 0$) up to the 'cut/split layer' q of the ML model with its private dataset D_k with common attributes A and common labels L. In each round, the participant sends its gradients from the cut layer ∇q to S, where the training continues for the last t layers, and later S sends the gradients back to the participant for backward-propagation of M. Training continues till M converges or fixed rounds R.*

SL splits the full ML model $M = M_S(M_C(\cdot))$ into multiple smaller network portions and trains them separately on a server (M_S) and (distributed) clients (M_C) with their local data. The relay-based training in SL makes the clients' resources idle because only one client engages with the server at one instance, causing a significant increase in the training overhead with many clients. SL can become inefficient with many clients; unlike SL, training in FL and **SplitML** is parallel.

SplitNN [84], a distributed deep learning method, does not share raw data or model details with collaborating institutions. The proposed configurations of SplitNN (a) Simple vanilla SL, (b) SL without label sharing, and (c) SL for vertically partitioned data cater to practical health settings [84]. A drawback of the (a) vanilla SL is that the output labels L of training samples must be transmitted from clients to the server. In (b), a U-shaped configuration for SL is presented to alleviate the problem of label sharing in SL, where four sub-networks are used to make the model training possible without label sharing. The (a) and (b) approaches are suitable for horizontal data where training samples in different clients share the same feature space but do not share the sample space.

In contrast, vertical SL schemes deal with data structures in which different features of the same training samples are available to different clients. In (c), a vertical SL configuration is presented in which two clients containing different modalities of the same training samples train their specific sub-networks up to the cut layer q. Then, the outputs of their sub-networks are concatenated and sent to the server. The server performs forward and backward propagations and sends the gradient to each client to complete the backward propagation and train the overall network. However, SL introduces many security risks like data leakage and model theft.

Like FL and **SplitML**, the SL scheme protects clients' data by not sending it directly to the server. However, model inversion attacks can compromise data protection in SL. Model inversion is a class of attacks that attempts to recreate data fed through a predictive model, either at inference or training [38,85–87].

It has been shown that model inversion attacks work better on earlier hidden layers of a neural network due to the increased structural similarity to input data [88]. This makes SL a prime target for model inversion attacks. In **SplitML** training, we do not share gradients from cut layer ∇q but collaborate on encrypted model weights. In **SplitML**, inference is done locally after a model is converged in training, and if a client chooses to perform consensus with peers, it encrypts ∇q with FHE. These measures enhance defenses against inference attacks.

In a backdoor attack, a malicious party could introduce a backdoor into the model trained by the SL participants. This would allow the malicious party to control the model's output, even if they cannot access it. In **SplitML**, these attacks are more challenging due to the heterogeneity of models and the fact that models are not partitioned between clients and servers.

In Sybil attacks [89,90], a malicious party could create multiple fake identities to influence the model's training. This could be done to skew the model's output in the desired direction. In **SplitML**,

 doi:10.20944/preprints202512.1579.v1

the setup phase requires generating collaborative evaluation keys EK and a secret key SK shares for fused decryption. Hence, it is computationally infeasible for a PPT adversary to set up fake clients and participate in encrypted collaborative training.

Pasquini et al. [91] demonstrated that an honest-but-curious server could obtain the clients' data during training. They propose a Feature-space Hijacking Attack (FSHA) by adapting and extending the inference attack in [92] to make it work in SL. FSHA assumes an attacker with access to a public dataset that follows a similar distribution to the client's private training dataset. Since the decoder essentially knows how to invert the values belonging to that latent space, it can invert values received from the client and obtain the original inputs. During the attack, the server exploits its control on the training process and steers it towards a specific target feature-space that is appositely crafted. Such an attack encompasses two phases: (1) a setup phase where the server hijacks the learning process, and (2) a subsequent inference phase where the server can freely recover the smashed data sent from the clients.

UnSplit [93] proposed a novel symbiotic combination of model stealing and model inversion in a limited threat model within the context of SL. UnSplit assumes an honest-but-curious attacker, a much weaker form than a powerful malicious attacker, who knows the (global) model architecture but not the parameters. The attacker aims to recover any input given to the network and obtain a functionally similar (i.e., similar performance on unseen data) clone of the client network.

Unlike FSHA [91], UnSplit [93] has no assumptions about the attacker's knowledge of a public data set related to the original task. Unlike SL, the server (adversary) does not control the model after the split layer and does not influence the learning process in **SplitML**. Moreover, encrypted layer weights are shared in **SplitML**, not the gradients, making such attacks infeasible. In **SplitML**, the attacker may not know the client's local model due to heterogeneity. Additionally, **SplitML** deploys multi-key FHE with (ϵ, δ)-DP (refer §A for details) to provide provable security guarantees.

Appendix B.3 Integrating FL with SL

Thapa et al. [27] proposed SplitFed Learning (SFL), combining FL and SL to train models on horizontally partitioned data. SFL considers the advantages of FL and SL while emphasizing data privacy and the robustness of the model by incorporating Differential Privacy (DP). In FL, the central server has access to the entire model. In contrast, the central server has access only to the server-side portion of the model and the smashed data (i.e., activation vectors of the cut layer) from the client in SL and SFL. In SFL, all clients perform forward-propagation on their client-side model in parallel and pass their smashed data to the central server. With each client's smashed data, the central server processes forward-propagation and backward-propagation on its server-side model. It then sends the gradients of the smashed data to the respective clients for backward propagation. Afterward, the server updates its model by federated averaging algorithm (FedAvg [3]), and each client performs the backward propagation on their client-side local model and computes its gradients. A DP mechanism makes gradients private and sends them to the fed server. The fed server conducts the FedAvg of the client-side local updates and sends them back to all participants.

SFL proposes two variants. In $SFLV1(splitfedv1)$, the server-side models of all clients are executed separately in parallel and then aggregated to obtain the global server-side model at each global epoch. $SFLV2(splitfedv2)$ processes the forward-backward-propagations of the server-side model sequentially concerning the client's smashed data (no FedAvg of the server-side models). SFLV2 is motivated by the intuition of the possibility of increasing the model accuracy by removing the model aggregation executed separately in parallel and then aggregated to obtain the global server-side model at each epoch. When the number of clients increases, the training time cost increases in the order: $SFLV2 < SFLV1 < SL$.

While SFL takes advantage of FL with SL, it also inherits its security challenges. For instance, the global model is shared across all participants, making inference and poisoning attacks easier. In our scheme **SplitML**, clients may have different bottom layers, making the attacks difficult. Another

 doi:10.20944/preprints202512.1579.v1

advantage of our scheme is consensus after training, where a client may request its peers to vote on some of the samples during inference.

Existing Federated and Split Learning approaches work on vertically or horizontally partitioned data and cannot handle sequentially partitioned data where multiple sequential data segments are distributed across clients. The most common ML models for training on sequential data are Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM). Existing SL approaches work on feed-forward networks and cannot be used for recurrent networks. This data distribution is different from vertical and horizontal partitioned data. FedSL [94] proposed a novel federated split learning framework to train models on distributed sequential data. RNNs are split into sub-networks, and each sub-network is trained on a client containing single segments of multiple-segment training sequences.

Previous SL methods split feed-forward NN while FedSL split RNN. SL is performed between clients, and FL is performed between clients and the server. In the SL step of SplitFed [27], label sharing is required among clients and the split server. However, in FedSL [94], the complete model is not shared, and label sharing is not required between clients or clients and servers. This line of work involving sequential partition is parallel to ours.

Appendix C Defenses

In this section, we present existing solutions to prominent machine-learning attacks discussed in §3.

Appendix C.1 Model Poisoning

Yin et al. [95] presents the median statistics against Byzantine failures, which computes the median of local gradients as the global update in FL. Krum [96] adopts the Euclidean distance to determine outliers. The gradient with the closest distance to its neighboring gradients is selected as the global gradient. Bulyan [97] selects specific local gradients using Krum [96] and then aggregates for the global gradient. Auror [98] uses K-means to cluster local model updates and identify the outliers in FL.

Sybils [99] use cosine similarity to identify the closest gradients as the outliers against model poisoning in non-IID (Independently Identically Distribution), meaning malicious gradients have similar variations distinct from benign gradients. As local updates trained on non-IID data have relatively distinct distributions [100,101], these schemes in [97] and [98] may have a high probability of misjudging benign local updates trained on non-IID data as outliers. Sybils [99] also demonstrate a significant accuracy drop in an IID scenario, where IID gradients with similar distributions are quickly regarded as malicious gradients.

FL trust [102] is a Byzantine-tolerance mechanism against model poisoning based on the assumption of a clean validation dataset held by the server, which identifies the outliers by measuring the cosine similarity of each local gradient to the benign gradient trained on the validation data. With the same assumption, the trimmed-means method [81] is also proposed with a validation dataset.

Unfortunately, these schemes are not applicable in the PPFL setting since the server in PPFL is prohibited from collecting and operating training and validation data directly in compliance with privacy policies [103,104].

Liu et al. [105] proposed the Privacy-Enhanced FL (PEFL) against poisoning attacks with IID data, which uses the Pearson correlation coefficient to identify the outliers. It adopts the two-server model for secure computation, in which a trusted server holds the secret key to decrypt intermediate encrypted parameters. This defense strategy takes the strong security assumption that the confidentiality of the secret key cannot be compromised.

Once the adversary corrupts the server, it is easy to leak the secret key, resulting in privacy leakage. Ma et al. [106] designed a privacy-preserving defense strategy using two-trapdoor homomorphic encryption (ShieldFL), which can resist encrypted model poisoning without compromising privacy

 doi:10.20944/preprints202512.1579.v1

in PPFL. They also propose the Byzantine-tolerance aggregation using cosine similarity, which can achieve robustness for both IID and non-IID data.

Appendix C.2 Membership Inference

Wei et al. [107] proposed an algorithm called NbAFL, which adds artificial noise before aggregation. Geyer et al. [108] considered this problem from the client-side perspective, and they used random sub-sampling and the Gaussian mechanism to distort the sum of all updates. Hao et al. [109] integrated additively homomorphic encryption [110] with DP. Their method provides more substantial protection to prevent privacy leakage when multiple nodes or central servers collude.

Appendix C.3 Model Inversion

Abuadbba et al. [111] apply noise to the intermediate tensors in an SL to defend against model inversion attacks on one-dimensional ECG data. The authors frame this defense as a DP [112] mechanism. However, in their work, adding noise significantly impacts the model's accuracy for modest ϵ values. A similar method, Shredder [113], adaptively generates a noise mask to minimize mutual information between input and intermediate data.

NoPeekNN [114] limits data reconstruction in SL by minimizing the distance correlation between the input data and the intermediate tensors during model training. NoPeekNN optimizes the model by a weighted combination of the task's loss and a distance correlation loss, which measures the similarity between the input and intermediate data. NoPeekNN's loss weighting is governed by a hyperparameter $\alpha \in [0, \infty]$. While NoPeekNN was shown to reduce an autoencoder's ability to reconstruct input data, it has not been applied to an adversarial model inversion attack.

Titcombe et al. [75] extend NoPeekNN [114] and propose a simple additive noise method to defend against model inversion. Their method can significantly reduce attack efficacy at an acceptable accuracy trade-off on MNIST. Their work defines a threat model for SL in which training and inference data are stolen in an FL system. They examine the practical limitations on attack efficacy, such as the amount of data available to an attacker and their prior knowledge of the target model. They introduce a method for protecting data, consisting of random noise added to the intermediate data representation of SL. However, they only consider the susceptibility of inference-time data to model inversion and do not investigate the efficacy of the attack on data collected during training.

As noise defense and NoPeekNN are introduced at the intersection between model segments, they do not protect against white box model inversion, which extracts training data memorized by a data owner's model segment. Other works [115,116] provide practical ways to mitigate model inversion attacks. However, they cannot achieve satisfactory mitigation when the client-side model has few layers (less than 3 in a VGG-11 model). A training-time DP mechanism, such as DP-SGD or PATE [70,71,117] could offer protection against other attack types (e.g., MI [35], Sybil attacks [89,90]).

Appendix C.4 Model Extraction

Model owners can use watermarks to detect and claim ownership of their stolen machine-learning models. Watermarking can be considered a form of poisoning by leveraging unused model capacity to overfit outlier input-output pairs known only to the defender to claim ownership. Authors in [118,119] introduced watermarking algorithms that rely on data poisoning [120]. Jia et al. [121] proposed Entangled Watermark Embedding (EWE) using Soft Nearest Neighbors Loss (SNNL), which forces the model to entangle representations for legitimate task data and watermarks.

References

1. de Montjoye, Y.A.; Radaelli, L.; Singh, V.K.; Pentland, A.S. Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science* **2015**, *347*, 536–539, [https://www.science.org/doi/pdf/10.1126/science.1256297]. https://doi.org/10.1126/science.1256297.
2. Zhang, J.; Li, C.; Qi, J.; He, J. A Survey on Class Imbalance in Federated Learning. *arXiv preprint arXiv:2303.11673* **2023**.

 doi:10.20944/preprints202512.1579.v1

3. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial intelligence and statistics. PMLR, 2017, pp. 1273–1282.
4. Gupta, O.; Raskar, R. Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications* **2018**, *116*, 1–8.
5. Yin, X.; Zhu, Y.; Hu, J. A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)* **2021**, *54*, 1–36.
6. Nasr, M.; Shokri, R.; Houmansadr, A. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In Proceedings of the 2019 IEEE symposium on security and privacy (SP). IEEE, 2019, pp. 739–753.
7. Xia, Q.; Tao, Z.; Hao, Z.; Li, Q. FABA: an algorithm for fast aggregation against byzantine attacks in distributed neural networks. In Proceedings of the IJCAI, 2019.
8. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the Proceedings of the forty-first annual ACM symposium on Theory of computing, 2009, pp. 169–178.
9. Trivedi, D. Privacy-Preserving Security Analytics, 2023.
10. Trivedi, D. The Future of Cryptography: Performing Computations on Encrypted Data. *ISACA Journal* **2023**, *1*.
11. Angel, S.; Chen, H.; Laine, K.; Setty, S. PIR with compressed queries and amortized query processing. In Proceedings of the 2018 IEEE symposium on security and privacy (SP). IEEE, 2018, pp. 962–979.
12. Bos, J.W.; Castryck, W.; Iliashenko, I.; Vercauteren, F. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In Proceedings of the Progress in Cryptology-AFRICACRYPT 2017: 9th International Conference on Cryptology in Africa, Dakar, Senegal, May 24-26, 2017, Proceedings. Springer, 2017, pp. 184–201.
13. Boudguiga, A.; Stan, O.; Sedjelmaci, H.; Carpov, S. Homomorphic Encryption at Work for Private Analysis of Security Logs. In Proceedings of the ICISSP, 2020, pp. 515–523.
14. Bourse, F.; Minelli, M.; Minihold, M.; Paillier, P. Fast homomorphic evaluation of deep discretized neural networks. In Proceedings of the Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38. Springer, 2018, pp. 483–512.
15. Kim, M.; Lauter, K. Private genome analysis through homomorphic encryption. In Proceedings of the BMC medical informatics and decision making. BioMed Central, 2015, Vol. 15, pp. 1–12.
16. Trama, D.; Clet, P.E.; Boudguiga, A.; Sirdey, R. Building Blocks for LSTM Homomorphic Evaluation with TFHE. In Proceedings of the International Symposium on Cyber Security, Cryptology, and Machine Learning. Springer, 2023, pp. 117–134.
17. Trivedi, D.; Boudguiga, A.; Triandopoulos, N. SigML: Supervised Log Anomaly with Fully Homomorphic Encryption. In Proceedings of the International Symposium on Cyber Security, Cryptology, and Machine Learning. Springer, 2023, pp. 372–388.
18. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. Cryptology ePrint Archive, Report 2016/421, 2016. https://eprint.iacr.org/2016/421.
19. Badawi, A.A.; Alexandru, A.; Bates, J.; Bergamaschi, F.; Cousins, D.B.; Erabelli, S.; Genise, N.; Halevi, S.; Hunt, H.; Kim, A.; et al. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915.
20. Al Badawi, A.; Bates, J.; Bergamaschi, F.; Cousins, D.B.; Erabelli, S.; Genise, N.; Halevi, S.; Hunt, H.; Kim, A.; Lee, Y.; et al. OpenFHE: Open-Source Fully Homomorphic Encryption Library. In Proceedings of the Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, New York, NY, USA, 2022; WAHC'22, pp. 53–63. https://doi.org/10.1145/3560827.3563379.
21. Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H.B.; Patel, S.; Ramage, D.; Segal, A.; Seth, K. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In Proceedings of the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 2017; CCS '17, p. 1175–1191. https://doi.org/10.1145/3133956.3133982.
22. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* **2009**, *22*, 1345–1359.
23. Tan, C.; Sun, F.; Kong, T.; Zhang, W.; Yang, C.; Liu, C. A survey on deep transfer learning. In Proceedings of the Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on

 doi:10.20944/preprints202512.1579.v1

Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27. Springer, 2018, pp. 270–279.
24. Ring, M.B. CHILD: A first step towards continual learning. *Machine Learning* **1997**, *28*, 77–104.
25. Yang, Q.; Ling, C.; Chai, X.; Pan, R. Test-cost sensitive classification on data with missing values. *IEEE Transactions on Knowledge and Data Engineering* **2006**, *18*, 626–638.
26. Zhu, X.; Wu, X. Class noise handling for effective cost-sensitive learning by cost-guided iterative classification filtering. *IEEE Transactions on Knowledge and Data Engineering* **2006**, *18*, 1435–1440.
27. Thapa, C.; Arachchige, P.C.M.; Camtepe, S.; Sun, L. Splitfed: When federated learning meets split learning. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, 2022, Vol. 36, pp. 8485–8493.
28. Security Notes for Homomorphic Encryption — OpenFHE documentation, 2022. https://openfhe-development.readthedocs.io/en/latest/sphinx_rsts/intro/security.html.
29. Tramèr, F.; Zhang, F.; Juels, A.; Reiter, M.K.; Ristenpart, T. Stealing Machine Learning Models via Prediction APIs. In Proceedings of the USENIX security symposium, 2016, Vol. 16, pp. 601–618.
30. Juuti, M.; Szyller, S.; Marchal, S.; Asokan, N. PRADA: protecting against DNN model stealing attacks. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2019, pp. 512–527.
31. Li, B.; Micciancio, D. On the security of homomorphic encryption on approximate numbers. In Proceedings of the Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40. Springer, 2021, pp. 648–677.
32. openfhe-development/src/pke/examples/CKKS_ NOISE_FLOODING.md at main · openfheorg/openfhe-development, 2022. https://github.com/openfheorg/openfhe-development/blob/main/src/pke/examples/CKKS_NOISE_FLOODING.md.
33. Ogilvie, T. Differential Privacy for Free? Harnessing the Noise in Approximate Homomorphic Encryption. *Cryptology ePrint Archive* **2023**.
34. He, S.; Zhu, J.; He, P.; Lyu, M.R. Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics, 2020. https://doi.org/10.48550/ARXIV.2008.06448.
35. Shokri, R.; Stronati, M.; Song, C.; Shmatikov, V. Membership inference attacks against machine learning models. In Proceedings of the 2017 IEEE symposium on security and privacy (SP). IEEE, 2017, pp. 3–18.
36. Nicolae, M.I.; Sinn, M.; Tran, M.N.; Buesser, B.; Rawat, A.; Wistuba, M.; Zantedeschi, V.; Baracaldo, N.; Chen, B.; Ludwig, H.; et al. Adversarial Robustness Toolbox v1.2.0. *CoRR* **2018**, *1807.01069*.
37. LeCun, Y.; Cortes, C. MNIST handwritten digit database **2010**.
38. Fredrikson, M.; Jha, S.; Ristenpart, T. Model inversion attacks that exploit confidence information and basic countermeasures. In Proceedings of the Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 1322–1333.
39. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* **2014**.
40. Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* **2017**.
41. Rakin, A.S.; He, Z.; Fan, D. Bit-flip attack: Crushing neural network with progressive bit search. In Proceedings of the Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1211–1220.
42. Li, J.; Rakin, A.S.; Chen, X.; Yang, L.; He, Z.; Fan, D.; Chakrabarti, C. Model Extraction Attacks on Split Federated Learning. *arXiv preprint arXiv:2303.08581* **2023**.
43. Jagielski, M.; Carlini, N.; Berthelot, D.; Kurakin, A.; Papernot, N. High accuracy and high fidelity extraction of neural networks. In Proceedings of the Proceedings of the 29th USENIX Conference on Security Symposium, 2020, pp. 1345–1362.
44. Jagielski, M.; Carlini, N.; Berthelot, D.; Kurakin, A.; Papernot, N. High-fidelity extraction of neural network models. *arXiv preprint arXiv:1909.01838* **2019**.
45. He, P.; Zhu, J.; Zheng, Z.; Lyu, M.R. Drain: An online log parsing approach with fixed depth tree. In Proceedings of the 2017 IEEE international conference on web services (ICWS). IEEE, 2017, pp. 33–40.
46. Foundation, P.S. Python 3.11, 2023.
47. Buitinck, L.; Louppe, G.; Blondel, M.; Pedregosa, F.; Mueller, A.; Grisel, O.; Niculae, V.; Prettenhofer, P.; Gramfort, A.; Grobler, J.; et al. API design for machine learning software: experiences from the scikit-learn

 doi:10.20944/preprints202512.1579.v1

project. In Proceedings of the ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013, pp. 108–122.
48. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* **2014**, *6*, 1–36.
49. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In Proceedings of the Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23. Springer, 2017, pp. 409–437.
50. Fan, J.; Vercauteren, F. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* **2012**.
51. Gentry, C.; Sahai, A.; Waters, B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Proceedings of the Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I. Springer, 2013, pp. 75–92.
52. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachene, M. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Proceedings of the Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22. Springer, 2016, pp. 3–33.
53. Ducas, L.; Micciancio, D. FHEW: bootstrapping homomorphic encryption in less than a second. In Proceedings of the Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I 34. Springer, 2015, pp. 617–640.
54. Brakerski, Z. Fully Homomorphic Encryption Without Modulus Switching from Classical GapSVP. In Proceedings of the Proceedings of the 32Nd Annual Cryptology Conference on Advances in Cryptology — CRYPTO 2012 - Volume 7417, New York, NY, USA, 2012; pp. 868–886. http://dx.doi.org/10.1007/978-3-642-32009-5_50, https://doi.org/10.1007/978-3-642-32009-5_50.
55. Fan, J.; Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.
56. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. Fully Homomorphic Encryption without Bootstrapping. Cryptology ePrint Archive, Paper 2011/277, 2011. https://eprint.iacr.org/2011/277.
57. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In Proceedings of the Advances in Cryptology – ASIACRYPT 2016; Cheon, J.H.; Takagi, T., Eds., Berlin, Heidelberg, 2016; pp. 3–33.
58. Dwork, C. Differential privacy. In Proceedings of the International colloquium on automata, languages, and programming. Springer, 2006, pp. 1–12.
59. Dwork, C.; Roth, A.; et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* **2014**, *9*, 211–407.
60. Li, N.; Qardaji, W.; Su, D.; Wu, Y.; Yang, W. Membership privacy: A unifying framework for privacy definitions. In Proceedings of the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 889–900.
61. Vinterbo, S.A. Differentially private projected histograms: Construction and use for prediction. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2012, pp. 19–34.
62. Chaudhuri, K.; Monteleoni, C. Privacy-preserving logistic regression. *Advances in neural information processing systems* **2008**, *21*.
63. Zhang, J.; Zhang, Z.; Xiao, X.; Yang, Y.; Winslett, M. Functional mechanism: Regression analysis under differential privacy. *arXiv preprint arXiv:1208.0219* **2012**.
64. Rubinstein, B.I.; Bartlett, P.L.; Huang, L.; Taft, N. Learning in a large function space: Privacy-preserving mechanisms for SVM learning. *arXiv preprint arXiv:0911.5708* **2009**.
65. Jagannathan, G.; Pillaipakkamnatt, K.; Wright, R.N. A practical differentially private random decision tree classifier. In Proceedings of the 2009 IEEE International Conference on Data Mining Workshops. IEEE, 2009, pp. 114–121.
66. Shokri, R.; Shmatikov, V. Privacy-preserving deep learning. In Proceedings of the Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, 2015, pp. 1310–1321.
67. Pustozerova, A.; Mayer, R. Information leaks in federated learning. In Proceedings of the Proceedings of the Network and Distributed System Security Symposium, 2020, Vol. 10, p. 122.

 doi:10.20944/preprints202512.1579.v1

68. Zhu, L.; Liu, Z.; Han, S. Deep leakage from gradients. *Advances in neural information processing systems* **2019**, *32*.
69. McMahan, H.B.; Ramage, D.; Talwar, K.; Zhang, L. Learning differentially private language models without losing accuracy. CoRR abs/1710.06963 (2017). *arXiv preprint arxiv:1710.06963* **2017**.
70. Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H.B.; Mironov, I.; Talwar, K.; Zhang, L. Deep learning with differential privacy. In Proceedings of the Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 308–318.
71. Papernot, N.; Song, S.; Mironov, I.; Raghunathan, A.; Talwar, K.; Erlingsson, Ú. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908* **2018**.
72. Sabater, C.; Bellet, A.; Ramon, J. Distributed differentially private averaging with improved utility and robustness to malicious parties **2020**.
73. Grivet Sébert, A.; Pinot, R.; Zuber, M.; Gouy-Pailler, C.; Sirdey, R. SPEED: secure, PrivatE, and efficient deep learning. *Machine Learning* **2021**, *110*, 675–694.
74. Dong, X.; Yin, H.; Alvarez, J.M.; Kautz, J.; Molchanov, P.; Kung, H. Privacy Vulnerability of Split Computing to Data-Free Model Inversion Attacks. *arXiv preprint arXiv:2107.06304* **2021**.
75. Titcombe, T.; Hall, A.J.; Papadopoulos, P.; Romanini, D. Practical defences against model inversion attacks for split neural networks. *arXiv preprint arXiv:2104.05743* **2021**.
76. Yang, Q.; Liu, Y.; Chen, T.; Tong, Y. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* **2019**, *10*, 1–19.
77. Truex, S.; Liu, L.; Gursoy, M.E.; Yu, L.; Wei, W. Demystifying membership inference attacks in machine learning as a service. *IEEE Transactions on Services Computing* **2019**, *14*, 2073–2089.
78. Wang, Z.; Song, M.; Zhang, Z.; Song, Y.; Wang, Q.; Qi, H. Beyond inferring class representatives: User-level privacy leakage from federated learning. In Proceedings of the IEEE INFOCOM 2019-IEEE conference on computer communications. IEEE, 2019, pp. 2512–2520.
79. Bagdasaryan, E.; Veit, A.; Hua, Y.; Estrin, D.; Shmatikov, V. How to backdoor federated learning. In Proceedings of the International Conference on Artificial Intelligence and Statistics. PMLR, 2020, pp. 2938–2948.
80. Hou, B.; Gao, J.; Guo, X.; Baker, T.; Zhang, Y.; Wen, Y.; Liu, Z. Mitigating the backdoor attack by federated filters for industrial IoT applications. *IEEE Transactions on Industrial Informatics* **2021**, *18*, 3562–3571.
81. Fang, M.; Cao, X.; Jia, J.; Gong, N.Z. Local model poisoning attacks to byzantine-robust federated learning. In Proceedings of the Proceedings of the 29th USENIX Conference on Security Symposium, 2020, pp. 1623–1640.
82. Diao, E.; Ding, J.; Tarokh, V. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264* **2020**.
83. Xu, Z.; Yu, F.; Xiong, J.; Chen, X. Helios: Heterogeneity-aware federated learning with dynamically balanced collaboration. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC). IEEE, 2021, pp. 997–1002.
84. Vepakomma, P.; Gupta, O.; Swedish, T.; Raskar, R. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* **2018**.
85. Chen, S.; Jia, R.; Qi, G.J. Improved techniques for model inversion attacks, 2020.
86. Zhang, Y.; Jia, R.; Pei, H.; Wang, W.; Li, B.; Song, D. The secret revealer: Generative model-inversion attacks against deep neural networks. In Proceedings of the Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 253–261.
87. Wu, M.; Zhang, X.; Ding, J.; Nguyen, H.; Yu, R.; Pan, M.; Wong, S.T. Evaluation of inference attack models for deep learning on medical data. *arXiv preprint arXiv:2011.00177* **2020**.
88. He, Z.; Zhang, T.; Lee, R.B. Model inversion attacks against collaborative inference. In Proceedings of the Proceedings of the 35th Annual Computer Security Applications Conference, 2019, pp. 148–162.
89. Douceur, J.R. The sybil attack. In Proceedings of the Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1. Springer, 2002, pp. 251–260.
90. Kairouz, P.; McMahan, H.B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A.N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* **2021**, *14*, 1–210.
91. Pasquini, D.; Ateniese, G.; Bernaschi, M. Unleashing the tiger: Inference attacks on split learning. In Proceedings of the Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, 2021, pp. 2113–2129.

 doi:10.20944/preprints202512.1579.v1

92. Hitaj, B.; Ateniese, G.; Perez-Cruz, F. Deep models under the GAN: information leakage from collaborative deep learning. In Proceedings of the Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, 2017, pp. 603–618.
93. Erdoğan, E.; Küpçü, A.; Çiçek, A.E. Unsplit: Data-oblivious model inversion, model stealing, and label inference attacks against split learning. In Proceedings of the Proceedings of the 21st Workshop on Privacy in the Electronic Society, 2022, pp. 115–124.
94. Abedi, A.; Khan, S.S. Fedsl: Federated split learning on distributed sequential data in recurrent neural networks. *arXiv preprint arXiv:2011.03180* **2020**.
95. Yin, D.; Chen, Y.; Kannan, R.; Bartlett, P. Byzantine-robust distributed learning: Towards optimal statistical rates. In Proceedings of the International Conference on Machine Learning. PMLR, 2018, pp. 5650–5659.
96. Blanchard, P.; El Mhamdi, E.M.; Guerraoui, R.; Stainer, J. Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in neural information processing systems* **2017**, *30*.
97. Guerraoui, R.; Rouault, S.; et al. The hidden vulnerability of distributed learning in byzantium. In Proceedings of the International Conference on Machine Learning. PMLR, 2018, pp. 3521–3530.
98. Shen, S.; Tople, S.; Saxena, P. Auror: Defending against poisoning attacks in collaborative deep learning systems. In Proceedings of the Proceedings of the 32nd Annual Conference on Computer Security Applications, 2016, pp. 508–519.
99. Fung, C.; Yoon, C.J.; Beschastnikh, I. The Limitations of Federated Learning in Sybil Settings. In Proceedings of the RAID, 2020, pp. 301–316.
100. Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; Chandra, V. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* **2018**.
101. Wang, H.; Kaplan, Z.; Niu, D.; Li, B. Optimizing federated learning on non-iid data with reinforcement learning. In Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE, 2020, pp. 1698–1707.
102. Cao, X.; Fang, M.; Liu, J.; Gong, N.Z. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995* **2020**.
103. Melis, L.; Song, C.; De Cristofaro, E.; Shmatikov, V. Exploiting unintended feature leakage in collaborative learning. In Proceedings of the 2019 IEEE symposium on security and privacy (SP). IEEE, 2019, pp. 691–706.
104. Goddard, M. The EU General Data Protection Regulation (GDPR): European regulation that has a global impact. *International Journal of Market Research* **2017**, *59*, 703–705.
105. Liu, X.; Li, H.; Xu, G.; Chen, Z.; Huang, X.; Lu, R. Privacy-enhanced federated learning against poisoning adversaries. *IEEE Transactions on Information Forensics and Security* **2021**, *16*, 4574–4588.
106. Ma, Z.; Ma, J.; Miao, Y.; Li, Y.; Deng, R.H. ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and Security* **2022**, *17*, 1639–1654.
107. Wei, K.; Li, J.; Ding, M.; Ma, C.; Yang, H.H.; Farokhi, F.; Jin, S.; Quek, T.Q.; Poor, H.V. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security* **2020**, *15*, 3454–3469.
108. Geyer, R.C.; Klein, T.; Nabi, M. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* **2017**.
109. Hao, M.; Li, H.; Luo, X.; Xu, G.; Yang, H.; Liu, S. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics* **2019**, *16*, 6532–6542.
110. Phong, L.T.; Aono, Y.; Hayashi, T.; Wang, L.; Moriai, S. Privacy-Preserving Deep Learning via Additively Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security* **2018**, *13*, 1333–1345.
111. Abuadbba, S.; Kim, K.; Kim, M.; Thapa, C.; Camtepe, S.A.; Gao, Y.; Kim, H.; Nepal, S. Can we use split learning on 1d cnn models for privacy preserving training? In Proceedings of the Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, 2020, pp. 305–318.
112. Dwork, C. Differential privacy: A survey of results. In Proceedings of the Theory and Applications of Models of Computation: 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings 5. Springer, 2008, pp. 1–19.
113. Mireshghallah, F.; Taram, M.; Ramrakhyani, P.; Jalali, A.; Tullsen, D.; Esmaeilzadeh, H. Shredder: Learning noise distributions to protect inference privacy. In Proceedings of the Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 2020, pp. 3–18.
114. Vepakomma, P.; Gupta, O.; Dubey, A.; Raskar, R. Reducing leakage in distributed deep learning for sensitive health data. *arXiv preprint arXiv:1812.00564* **2019**, *2*.

 doi:10.20944/preprints202512.1579.v1

115. Li, J.; Rakin, A.S.; Chen, X.; He, Z.; Fan, D.; Chakrabarti, C. Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning. In Proceedings of the Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10194–10202.
116. Vepakomma, P.; Singh, A.; Gupta, O.; Raskar, R. NoPeek: Information leakage reduction to share activations in distributed deep learning. In Proceedings of the 2020 International Conference on Data Mining Workshops (ICDMW). IEEE, 2020, pp. 933–942.
117. Papernot, N.; Abadi, M.; Erlingsson, U.; Goodfellow, I.; Talwar, K. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755* **2016**.
118. Zhang, J.; Gu, Z.; Jang, J.; Wu, H.; Stoecklin, M.P.; Huang, H.; Molloy, I. Protecting intellectual property of deep neural networks with watermarking. In Proceedings of the Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 2018, pp. 159–172.
119. Nagai, Y.; Uchida, Y.; Sakazawa, S.; Satoh, S. Digital watermarking for deep neural networks. *International Journal of Multimedia Information Retrieval* **2018**, *7*, 3–16.
120. Jagielski, M.; Oprea, A.; Biggio, B.; Liu, C.; Nita-Rotaru, C.; Li, B. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In Proceedings of the 2018 IEEE symposium on security and privacy (SP). IEEE, 2018, pp. 19–35.
121. Jia, H.; Choquette-Choo, C.A.; Chandrasekaran, V.; Papernot, N. Entangled Watermarks as a Defense against Model Extraction. In Proceedings of the USENIX Security Symposium, 2021, pp. 1937–1954.