MASER: Efficient Privacy-Preserving Cross-Silo Federated Learning with Multi-Key Homomorphic Encryption

Abdullah Al Omar, Xin Yang, Euijin Choo*, Omid Ardakanian
Department of Computing Science, University of Alberta
{aomar3,xyang18,euijin,oardakan}@ualberta.ca

***Abstract*—Federated Learning (FL) is susceptible to privacy attacks, such as data reconstruction attacks, in which a semi-honest server or a malicious client infers information about other clients' datasets from their model updates or gradients. To enhance the privacy of FL, recent studies combined Multi-Key Homomorphic Encryption (MKHE) and FL, making it possible to aggregate the encrypted model updates using different keys without decryption. Despite the privacy guarantees of MKHE, existing approaches are not well-suited for real-world deployment due to their high computation and communication overhead. We propose MASER, an efficient MKHE-based privacy-preserving FL framework that leverages consensus-based model pruning and slicing techniques to reduce this overhead. Our experiments show that MASER strikes a good balance between privacy, accuracy, and efficiency.**

***Index Terms*—Multi-key Homomorphic Encryption, Privacy-preserving Federated Learning, Consensus-based Pruning**

I. Introduction

Cross-silo Federated Learning (FL) has emerged as a viable approach enabling multiple institutions, referred to as clients, to collaboratively train a model without sharing their private data [1]. In FL, a central server aggregates model parameter updates made independently by clients to produce a global model, which is then sent to clients for further training with their local data. Although FL protects privacy by avoiding data sharing with the central server, it has been shown that the model updates or gradients can still leak private information [2], [3]. To mitigate such risks in FL, various privacy-preserving techniques have been proposed.

Differential Privacy protects privacy by introducing noise into the model updates, causing degradation of the model performance particularly in complex learning tasks [4]. Secure Multi-Party Computation offers strong privacy guarantees by providing a zero-knowledge framework for aggregating model updates, but this comes at the expense of significant overhead and complex protocols [5]. Homomorphic Encryption (HE), on the other hand, provides strong privacy guarantees without introducing noise or requiring complex, multi-party protocols. HE enables computations directly on encrypted data without decryption. In FL, this allows the central server to aggregate encrypted client updates, protecting the confidentiality of individual client models [6]–[8]. This makes HE well-suited for cross-silo FL scenarios, where institutions require highly accurate models but may not fully trust the central server to refrain from intrusive inferences, such as attempting to recover training data.

Several HE-based FL approaches have been proposed, most of which are single-key HE (SKHE)-based, where all clients share the same public and secret key pair [6]–[9]. This poses a risk, as a malicious client can decrypt other clients' updates using the shared key and perform intrusive inferences on their data. To tackle this issue, recent works have adopted Multi-Key HE (MKHE)-based schemes, where each client uses its own key pair [10]–[12]. This provides stronger privacy protection in that even if one client is malicious, other clients' privacy will not be compromised.

HE-based FL methods, however, impose significant overheads due to computationally expensive cryptographic operations and increased data size for transmission [7], [9]. To reduce overheads, optimization methods have been proposed, such as packing [7], batching [8], and masking [6], [9]. Despite these efforts, the packing and batching methods still incur high overheads for large models [7], [9]. The masking techniques may expose unencrypted data, making it vulnerable to reconstruction attacks [6], [9].

To address these challenges, we propose MASER, an efficient MKHE-based privacy-preserving FL framework that guarantees strong privacy, reduces the overheads of HE, and maintains model performance.The key idea is to effectively and adaptively sparsify the model prior to encryption, lowering computation and communication overhead of MKHE-based FL without compromising model performance.

MASER employs magnitude-based weight pruning [13] to identify a sparse subnetwork consisting of critical weights from models trained by each client, and introduces a consensus method to ensure agreement among clients on how to sparsify their models before encryption. Specifically, clients compute magnitudes and rank the weights after each local training round. Then, each client selects the top-ranked weights based on a threshold, i.e., the most important parameters, and generates a mask to indicate their choices. The *local masks* generated by clients are not necessarily the same, as the important parameters are chosen based on their local dataset. To decide what parameters should be retained and encrypted, the local masks are sent to the server that produces a *global mask* using a majority voting. This approach offers robustness against malicious clients generating *poisoned ranks* [14]. Upon

2025 IEEE International Conference on Big Data (BigData) | 979-8-3315-9447-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/BIGDATA66926.2025.11401805

receiving the global mask, each client applies it to sparsify the model. To further minimize overhead, we incorporate slicing by dividing the sparsified model parameters into smaller chunks tailored to the MKHE's key size. The sliced parameters are encrypted and transmitted to the server for aggregation.

Contribution: Our contribution is threefold:

- We present a practical privacy-preserving cross-silo FL framework utilizing MKHE to effectively counter privacy attacks from an honest-but-curious (HBC) server and malicious clients.
- We propose a novel consensus-based sparsification method, called MASER, that adaptively identifies and updates the important parameters for encryption in each MKHE-based FL round. MASER ensures that sparsification has negligible impacts on model performance. To our knowledge, this is the first work on privacy-preserving FL that achieves significant efficiency gains while maintaining performance and strong privacy.
- We evaluate MASER across multiple datasets and model architectures to show its performance, privacy capability, and robustness. We show that MASER achieves performance comparable to the best baselines with significantly reduced overhead.

Paper Outline: Section 2 presents the key concepts underlying our approach. Section 3 and Section 4 describe our threat model and approach, MASER, respectively. We discuss experimental results in Section 5 and related work in Section 6. Section 7 concludes the paper.

II. Background

A. Cross-Silo Federated Learning

Cross-silo FL offloads model training from a central server to clients [1]. The clients train *local models* using their data, share them with a central server for aggregation into a *global model*, which is sent to clients for the next training round.

Consider a neural network model $\hat{y} = f(x;\theta)$, where x is the input data, θ represents the model parameters, and $\hat{y}$ is the predicted label for the input x. Let t be the index for the current training round, η be the learning rate, and g^t be the gradient of loss with respect to the parameters. The model parameters are updated iteratively as follows:

$$\theta^{t+1} = \theta^t - \eta \cdot g^t, \tag{1}$$

The training starts with the aggregation server initializing a model θ^0 and distributing it to all or a subset of participating clients. Each client i updates the received model by training it on its local dataset $\mathcal{D}_i$. Thus, the local update of client i at global training round t is governed by:

$$g_i^t = \nabla_{\theta_i} \mathcal{L}(\theta_i^t; \mathcal{D}_i). \tag{2}$$

After the local training, each client sends its local model parameters θ_i^t to the server. The server then aggregates θ_i^t, for example using the Federated Averaging (FedAvg) [15]:

$$\theta^{t+1} = \sum_{i=1}^{N} \alpha_i \theta_i^t, \tag{3}$$

where $\alpha_i = \frac{|D_i|}{|D|}$ is the weight assigned to client i based on the size of its local dataset and N is the total number of clients participating in the model training in that round. The aggregated model θ^{t+1} will be distributed to the participating clients for the next round of training until convergence.

B. Multi-key Homomorphic Encryption

Homomorphic encryption allows computation on encrypted data (usually simple arithmetic operations) without decryption, such that the decrypted data is consistent with the computation on the plaintext data. In the context of FL, the encrypted data are model parameters and the computation performed on encrypted data is model aggregation. MKHE lets clients encrypt their model updates with different keys. The server can still aggregate them without accessing the decryption keys, thus enhancing privacy.

Definition 1 (RLWE [16])**.** *Let $\mathcal{R}_q$ be a polynomial ring and ψ be the error distribution over $\mathcal{R}_q$ with q being a prime integer. Given a secret polynomial $s(x)$ chosen from the dual fractional ideal of $\mathcal{R}_q$, we generate a sample $(a_i(x), s(x)\cdot a_i(x)+e_i(x))$ by choosing $a_i(x)$ from $\mathcal{R}_q$ uniformly at random and sampling $e_i(x)$ from ψ. The problem of Ring Learning with Errors (RLWE) concerns distinguishing arbitrarily many independent pairs of the form*

$$\Big(a_i(x), b_i(x)\Big) = \Big(a_i(x), s(x)\cdot a_i(x) + e_i(x)\Big) \in \mathcal{R}_q^2$$

from uniformly random and independent pairs.

Let $\mathcal{R} = \mathbb{Z}[x]/(x^{\mathrm{n}} + 1)$ be the cyclotomic ring with a power-of-two dimension n, where $\mathbb{Z}[x]$ is a polynomial ring with integer coefficients, and $\mathcal{R}_q = \mathcal{R}/\langle q \rangle$. For this ring and the appropriate choice of error distribution, the RLWE problem is hard [16]. Following [6], [11], we use CKKS, and in particular, its multi-key variant, MKCKKS [17], as our homomorphic encryption algorithm. MKCKKS enables clients to independently encrypt their updates, thereby enhancing privacy by keeping each client's data confidential. Its compatibility with real-number arithmetic is ideal for FL, where numerical precision is critical for model updates. xMKCKKS [11] is an extension of MKCKKS that uses a common public key for encryption, which is the sum of the public keys of all clients. By using public key aggregation, xMKCKKS minimizes privacy risks in the decryption phase, ensuring that decryption accuracy and privacy are maintained even with multiple client keys. We thus adopt xMKCKKS's public key aggregation method in MASER.

Let us denote the security parameter by λ, the secret key distribution by $\mathcal{X}$, and the space of local models by $\mathcal{M}$. The functions of MKHE are defined below:

- `Setup`(1^λ): Given the security parameter λ, this function defines the *public parameters (pp)*=$(n, q, \mathcal{X}, \psi, a)$. These public parameters are the same for all the clients.
- `KeyGen`(pp): Each client i invokes this to generate their public and secret keys. Specifically, the secret key is sampled from $\mathcal{X}$ and the error vector is sampled from ψ^d. Here, d denotes the dimension of the error vector, aligning it with

the structure of the polynomial ring $\mathcal{R}_q^d$ from which a was sampled uniformly at random in `Setup`. The public key pk_i is then calculated using the corresponding secret key sk_i, the error vector e_i, and the random vector a ($pk_i = -sk_i \cdot a + e_i \mod q$ in $\mathcal{R}_q^d$). Clients will share their pk_i with the key manager for aggregation. To aggregate public keys, we use the scheme in xMKCKKS [11]:

$$pk = \sum_{i=1}^{N} pk_i = \sum_{i=1}^{N} (-sk_i) \cdot a + \sum_{i=1}^{N} e_i \pmod q. \quad (4)$$

The aggregated public key, pk, prevents an HBC server from directly decrypting the ciphertext sent by each client. It is distributed to all clients for homomorphic encryption.

- $\texttt{Enc}(\theta_i; pk)$: Let the model parameters of client i be $\theta_i \in \mathcal{R}$. For encryption, we use the aggregated public key pk along with random vectors a and b, where $a = a[0]$ and $b = b[0]$. Here, $a[0]$ and $b[0]$ indicate that we take their first elements to simplify and optimize the encryption process. Errors e_0 and e_1 are sampled from the error distribution ψ. This function returns the encrypted weight parameters $\theta_i = (d_0, d_1) \in \mathcal{R}_q^2$. The values d_0 and d_1 are used to sample the weight parameters in the ring $\mathcal{R}_q^2$, where $d_0 = pk \cdot b + \theta_i + e_0 \pmod q$ and $d_1 = pk \cdot a + e_1 \mod q$. Here, d_0 is generated using the actual update θ_i of client i, while d_1 is generated with only the error component. A level-l multi-key encryption of a client's model parameters θ_i with respect to the secret keys $sk_i = (1, sk_1, sk_2, sk_3, ..., sk_i)$ is a vector in the ciphertext space $\theta_i = (d_0, d_1, d_2, d_3, ..., d_i) \in \mathcal{R}_{ql}^{(k+1)}$ satisfying $\langle \theta_i, sk_i \rangle \approx \theta_i \pmod q_l$. In the case of a homomorphic operation, e.g., the addition of two ciphertext vectors θ_i and θ_j, this operation returns an encrypted θ_k such that $\langle \theta_k, sk_i \rangle_{ql}$ is approximately equal to $\theta_i + \theta_j$.
- $\texttt{PartialDec}(\theta_i, sk_i)$: In partial decryption, each client i will have the secret key $sk_i \in \mathcal{R}$. Given a polynomial d_i and a secret key sk_i, they will sample an error $e_i \leftarrow \psi$ for noise flooding and return $p_i = d_1 \cdot sk_i + e_i \mod q$.
- $\texttt{Merge}(ct_0, \{p_i\}_{1 \le i \le k})$: This function outputs $p = ct_0 + \sum_{i=1}^{k} p_i \mod q$, where ct denotes the encrypted ciphertext. For a multi-key ciphertext at global round $t+1$, $\theta^{t+1} = (ct_0, ct_1, \ldots, ct_k)$, MKHE *merges* by calculating:

$$p = ct_0 + \sum_{i=1}^{k} p_i = ct_0 + \sum_{i=1}^{k} e_i \approx \langle \theta^{t+1}, sk_i \rangle \pmod q.$$

Here, ct_0 is the initial encrypted model update, and p_i is a partial decryption contributed by client i. The merge process approximates the inner product of the model update θ^{t+1} with the secret key sk_i. Then, scaling factors are removed to retrieve the plaintext model update θ^{t+1}.

C. Neural Network Pruning

Model pruning techniques can be classified into two categories according to the timing of pruning. The first category assigns a score to each parameter *after training* and removes those with the lowest scores. Among various methods, Magnitude-based Pruning (MP) is commonly used [18]. MP removes weights with the smallest magnitudes, which has no significant impact on performance. Specifically, given a pruning threshold κ, MP generates a mask M as follows:

$$M_{(j,k)} = \begin{cases} 1, & \text{if } |W_{(j,k)}| \ge \kappa, \\ 0, & \text{otherwise}, \end{cases} \quad (5)$$

where $W_{(j,k)}$ is the k-th element of the model weight vector at the j-th layer. The mask is applied to the model weights through element-wise multiplication. Note that MP does not prune biases in the model parameters and is shown to achieve a good balance between sparsity and accuracy [19].

The second category prunes *at initialization*. They heuristically identify and remove unimportant connections in a randomly initialized neural network without training [20]–[22]. Although the pruning cost are reduced by eliminating model training, the pruned network may not achieve good accuracy achieved by the MP [23]. Since model training occurs in multiple rounds in FL, we are not restricted to single-shot foresight pruning. We thus employ MP in each FL round to find a better sparse subnetwork.

Consensus-Based Model Pruning in FL. Each FL client is responsible for pruning its own model by generating a mask, which cannot be delegated to the server for privacy. The masks generated by different clients may differ significantly, especially in the presence of data heterogeneity. Aggregating models sparsified by different masks would hamper the convergence of FL. Consensus-based model pruning addresses this by reaching a consensus on the mask to be applied to all clients, aligning their model updates for meaningful aggregation [18], [24]. MASER employs majority voting for consensus, requiring only one extra communication round and no data replication. Note that for pruning at initialization, clients need to reach a consensus on the mask only once before FL training, whereas for pruning after training, a new consensus needs to be reached in every training round.

D. Motivation

MKHE and consensus-based pruning have been explored separately in FL, but to our knowledge, no prior work has leveraged consensus-based pruning to enhance the efficiency of MKHE-based FL. Existing consensus-based pruning might be readily applied to SKHE-based FL, where model alignment is straightforward due to the shared key. However, different keys in MKHE complicate model alignment and aggregation. While this can be addressed by fully encrypting the masked model without pruning, it leads to a significant overhead [9]. Thus, a new method is needed to balance privacy, performance, and efficiency.

III. Threat Model

We consider the aggregation server as an HBC adversary, following the literature [25], [26]. This passive adversary performs model aggregation faithfully and adheres to the protocol, but is curious to infer private information from model updates in FL.

We assume that clients may be malicious; however, the number of malicious clients is not enough to compromise

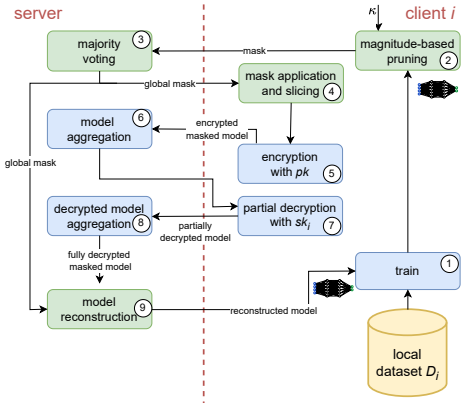

Fig. 1: MASER's components and data flow

our consensus method[1]. Malicious clients are not obligated to follow the protocol and may engage in adversarial actions, such as submitting bogus masks to the server. This model considers the highest level of threat from clients, requiring robust mechanisms to mitigate the impact of these adversarial actions to ensure the integrity of the system and data privacy. Note that MASER is designed for cross-silo FL, where clients (i.e., organizations) may seek to infer competitors' data, increasing the need for privacy-preserving methods. MASER addresses this by providing security measures that ensure client data privacy even in the presence of curious and potentially adversarial clients.

IV. Methodology

A. Overview

We propose a privacy-preserving cross-silo FL framework, MASER (MASking at Each Round). We design a novel consensus-based model pruning method to reduce the number of model parameters for encryption and transmission, while preserving model performance. Figure 1 shows the workflow of MASER consisting of following steps:

0) **Key Generation and Model Initialization.** Before the first FL round, each client generates its key pair and performs public key aggregation as explained in Section II-B. The FL aggregation server randomly initializes the model parameters and sends them to all clients.

1) **Model Training.** At the beginning of each round, each client receives the global model from the aggregation server and trains a model on their local dataset for e local epochs.

2) **Magnitude-based Pruning.** Once a client completes local training, it applies MP to the trained model to generate a local mask for model sparsification. The local mask consists of binary values, with "1"s indicating that the corresponding parameters are important and should be preserved and "0"s indicating that they are not important and can be pruned. Specifically, the weights with a magnitude greater than a pruning threshold κ and all biases are considered important. The local mask is then sent to the server for aggregation.

3) **Mask Aggregation through Majority Voting.** Upon receiving the masks from all participating clients, the server employs a majority voting to generate a global mask, in which "1"s correspond to parameters that are deemed important (i.e., voted for) by at least 50% of the clients. This consensus algorithm offers robustness to malicious clients.

4) **Mask Application and Slicing.** Each client receives the global mask from the server and sparsifies their local model based on the mask. The sparsification filters out the unimportant parameters and the remaining (important) parameters are reshaped into lists, named *slices*. All slices share the same length, determined by the security parameters.

5) **Model Encryption.** Clients encrypt their slices using the aggregated public key pk. These ciphertexts are shared with the server to perform aggregation using homomorphic addition, preventing the server from seeing the plaintext version of the important model parameters.

6) **Encrypted Model Aggregation.** The server receives the encrypted slices from all clients and aggregates models using homomorphic addition in the ciphertext space.

7) **Partial Decryption with Client Secret Keys.** The aggregated slices are sent back to the clients for partial decryption, where each client i uses their secret key sk_i to partially decrypt the aggregated slices. The result is sent to the server.

8) **Model Decryption and Parameter Averaging.** The server receives the partially decrypted ciphertexts for the aggregated slices and combines them to fully decrypt the aggregated slices. The fully decrypted slices contain the summation of all important model weights and biases, which are subsequently divided by the total number of clients participating in model training for averaging in plaintext. This process ensures that neither the server nor any client gains full access to other clients' model updates, because a specific client's update cannot be inferred from the aggregated result.

9) **Model Reconstruction.** The decrypted slices only contain the important weights and biases of the global model. The server should thus convert the decrypted slices into the shape of a full model based on the global mask first. MASER reconstructs the model by placing the elements from the decrypted slices into their corresponding positions in the original model while setting the unimportant weights to zero. The reconstructed global model is then distributed to the clients for the next round. Although model reconstruction on the client side can reduce communication costs, we perform it on the server side to minimize redundant computation and possibly allow a different set of clients to receive the full model and participate in the next round. In Section V-C, we show that this method adds no significant communication cost as the transmission overhead for the reconstructed model is negligible compared to the ciphertexts.

B. Model Training and Sparsification

MASER uses MKCKKS-based MKHE [17] to safeguard the model parameters shared with the aggregation server in

[1]More complex consensus protocols might be needed with a majority of malicious clients

Algorithm 1: MASER

Data: Secure parameters $\mathcal{S}$, number of clients m, slice size λ, local training dataset $\mathcal{D}_i$, prunning threshold κ
Result: Decrypted model parameters θ
Client i **Executes:**
1 $sk_i, pk_i \leftarrow \texttt{KeyGen}(\mathcal{S})$ // Generate local key pair
2 Send pk_i to key manager // To aggregate public keys
3 Receive aggregated public key pk: $pk_i \leftarrow pk$
4 **for** *each round* $t = 1, 2, \ldots$ **do**
5 Pull aggregated model parameters θ^t from the server: $\theta_i^t \leftarrow \theta^t$
6 **for** *local epoch* $e = 1, 2, \ldots$ **do**
7 $\theta_i^t \leftarrow \texttt{Train}(\theta_i^t, \mathcal{D}_i)$ // Update local model
8 $M_i^t \leftarrow \texttt{MaskGen}\ (\theta_i^t, \kappa)$ // Generate mask
9 Send M_i^t to server // To aggregate masks
10 Receive aggregated mask and update local mask: $M_i^t \leftarrow M^t$
11 $v_i^t \leftarrow \texttt{Slice}(\theta_i^t \odot M_i^t, \lambda)$ // Apply aggregated mask and prepare slices
12 $ct_i^t \leftarrow \texttt{Enc}(v_i^t, pk)$ // Encrypt slices using public key
13 Send ct_i^t to server
14 Receive ct^t from server // ct^t: aggregated ciphertext
15 $pd_i^t \leftarrow \texttt{PartialDec}(ct^t, sk_i)$ // Partially decrypt ct^t using secret key
16 Send pd_i^t to server
Server Executes :
17 Initialize θ^0
18 **for** *each round* $t = 1, 2, \ldots$ **do**
19 Send model θ^t to client i
20 Receive local mask M_i^t
21 $M^t \leftarrow \texttt{MaskAgg}(M_1^t, \cdots, M_m^t)$ // Perform mask aggregation
22 Send M^t to client i
23 Receive ct_i^t from clients // ct_i^t: ciphertext of slices
24 $ct^t \leftarrow \sum_{i=1}^{m} ct_i^t$ // Homomorphic parameter aggregation
25 Send ct^t to client i
26 Receive pd_i^t from clients // pd_i^t: partially decrypted ciphertext
27 $Q \leftarrow \texttt{Merge}(pd_1^t, \cdots, pd_m^t)$ // Aggregate partially decrypted ciphertext
28 $\theta^{t+1} \leftarrow \texttt{Reshape}(\texttt{Decode}(\frac{Q}{m}))$ // Decode and reshape Q into plaintext model

FL. In particular, each client i in MASER-based FL first generates their unique secret key sk_i and public key pk_i. However, MKCKKS-based MKHE is not directly applicable in FL, because using these public keys for encryption may lead to privacy leakage during decryption [11]. To mitigate this, we adopt the key aggregation strategy in xMKCKKS [11]. Concretely, the public keys are aggregated and each client uses the aggregated key to encrypt. Since sk_i used for decryption is withheld by each client, other clients or the server cannot decrypt the client's model updates. We employ a trusted key manager who aggregates the public keys shared by all clients to generate an aggregated public key pk. The pk is sent to all clients to replace their unique public key pk_i (Line 1 to 3 in Algorithm 1). After the key generation, the server initializes the model θ^0 and establishes a connection with each client to start the FL training.

Training. MASER supports model training when the data held by the clients are IID and non-IID. At the beginning of a training round t, each client i pulls the global model parameters, denoted as θ^t, from the server to update their local model parameters, denoted as θ_i^t. Then, they train their local model using their local dataset $\mathcal{D}_i$ for e local epochs. MASER follows FedAvg [15] and FedProx [27] to train the local model θ_i^t, when the clients' local datasets are IID and non-IID, respectively. FedProx [27] adds a proximal term to the local loss function, which helps to manage heterogeneity in non-IID data by constraining the divergence of each client's local model from the global model, thereby promoting better convergence across diverse datasets [27]. Note that the aggregation of model parameters is performed using homomorphic operations (Line 4 to 7 in Algorithm 1).

Remark. While our proof-of-concept employs FedAvg and FedProx for aggregation, it is important to note that other aggregation rules performing basic arithmetic operations on client updates (e.g., robust aggregation methods [28], [29]) can be easily adopted in MASER by substituting these operations with the equivalent homomorphic operations.

Sparsification. MKHE-based FL performs homomorphic encryption on the trained local model and share the encrypted models with the server for aggregation. However, homomorphic operations are computationally expensive, hence encrypting and transmitting the entire model would make the overhead prohibitive. MASER utilizes model sparsification to reduce the number of parameters for encryption and transmission. Specifically, it uses MP for sparsification to prune weights whose magnitudes are not among the top κ percent of all weights in the model. The not-pruned weights and all bias terms are considered *important parameters*. A local mask M_i^t is generated for the model parameterized by θ_i^t trained by client i, with "1"s indicating important parameters and "0"s meaning the pruned weights. Since our pruning threshold is defined as a percentile, the smallest magnitude required for a parameter to be deemed important is different for each client and each round of pruning.

In MASER, each client i sends its local masks M_i^t to the server to generate a unified global mask M^t, instead of directly applying M_i^t on i's local model θ_i^t. This ensures that model sparsification is consistent across all clients, allowing the server to accurately aggregate the encrypted sparse models. Without a consistent global mask, the aggregated model would be misaligned, with different sparsity patterns across clients, leading to incorrect parameter aggregation. Note that the server receives the binary masks only, and does not know the magnitudes of the important parameters.

For each training round, the server generates a global mask M^t by using a majority voting-based aggregation. The server performs element-wise summation for all local masks, divides the result by the total number of clients, m, and thresholds it against 50%. The mask aggregation rule is:

$$M^t(j,k) = \begin{cases} 1, & \text{if } \frac{1}{m}\sum_{i=1}^{m} M_i^t(j,k) \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where t is the index for the current training round, (j,k) is an index into the local mask M_i^t or the global mask M^t. A model parameter is deemed important, i.e., the global mask for it is "1", when at least 50% of clients had "1" for this parameter

in their local mask. The global mask M^t is shared with all clients to replace their local mask M_i^t. Each client performs element-wise multiplication of their updated local mask and the trained local model, i.e., $\widetilde{\theta}_i^t \leftarrow M_i^t \odot \theta_i^t$, where $\widetilde{\theta}_i^t$ is the sparsified local model for client i at round t (Line 8 to 10, Line 20 to 22 in Algorithm 1). Note that our mask aggregation method helps mitigate a mask poisoning attack by limiting each client's contribution on the global mask generation and enables learning a robust global model that generalizes across clients with varying data characteristics, benefiting even those with limited data in cross-silo FL.

C. Parameter Slicing for Efficient Homomorphic Encryption and Model Aggregation

The amount of data that can be encrypted by the CKKS-based algorithm (i.e., the number of slots) is determined by the degree of the polynomial modulus n, where n is a power of 2. The number of slots is $\lambda = \frac{n}{2}$. Therefore, to efficiently encrypt parameters of the sparsified model, we propose reshaping and dividing that model into multiple lists that share the same length of λ. We call these lists as *slices*, and perform homomorphic encryption and model aggregation in the ciphertext space per slice. In particular, we prepare the slices by starting with the model parameters in the first layer. The number of slices required by the first layer of model parameters is determined by $\lceil \frac{|\widetilde{\theta}_{(1)}|}{\lambda} \rceil$, where $|\widetilde{\theta}_{(1)}|$ is the number of important parameters in the first layer. If the important parameters in the first layer are unable to fully fill the last slice, the remaining slots in the last slice will be used to fill the important parameters in the second layer. Hence a total of $\lceil \frac{|\widetilde{\theta}|}{\lambda} \rceil$ slices will be required to store all important parameters. Should there be any remaining slots available in the last slice after filling in all important model parameters, they will be set to zero. This slicing technique allows efficient utilization of limited slots and encrypting the sparsified model with a minimum number of homomorphic operations. After slice preparation, each client performs homomorphic encryption using their aggregated public key pk for every slice and shares the encrypted ciphertext for every slice with the server to perform model aggregation in the ciphertext space. ct_i^t denotes the ciphertext encrypted by client i at round t (Line 11 to 13 in Algorithm 1).

Once the server receives the encrypted slices from all clients, it aggregates models by using homomorphic addition to sum up the ciphertext for each corresponding slice sent by all clients. The summed ciphertexts $ct^t = \sum_{i=1}^{m} ct_i^t$ are sent back to all clients for partial decryption. Note that we perform summation in the ciphertext space and defer the averaging to after the global model is fully decrypted. The averaging step multiplies the summation of model parameters ct^t by $\frac{1}{m}$, where m is the number of clients in this round. We assume the server knows m. Performing averaging in plaintext reduces the computational cost of homomorphic multiplication (Line 23 to 25 of Algorithm 1).

D. Decryption and Model Reconstruction

The decryption of the ciphertext requires clients to perform partial decryption before the server can fully decrypt the aggregated ciphertext to plaintext [17]. In the partial decryption phase, each client i receives the aggregated ciphertext ct^t from the server and uses their own secret key sk_i to decrypt their own portion in ct^t. Let pd_i^t denote the ciphertext partially decrypted by client i at round t. pd_i^t is sent back to the server to fully decrypt the ciphertext (Line 14 to 16 in Algorithm 1). The results after full decryption are the slices summed over all clients in plaintext. We then divide each element in these slices by m to obtain the aggregated model consisting of all important model parameters. The server then reshapes the important parameters in the slices to obtain the original shape of the neural network by referring to the position of the corresponding important parameters in the global mask. This step is named model reconstruction. Lastly, the fully decrypted and reshaped model parameters are sent to all clients to update their local models and to begin the next training round (Line 26 to 28 in Algorithm 1). Note that the server is unable to access individual clients' model updates because the aggregation process combines all clients' updates into a single model update in a way that renders individual contributions indistinguishable.

V. Experimental Analysis

A. Experimental Setup

Environment. We deployed MASER on two physically separated machines located in two datacenters on the same campus to simulate a real-world FL with network latency. We host all clients on one machine with an Intel Core i9-9940X CPU, 128 GB RAM, and an NVIDIA GeForce RTX 2080 Ti GPU, and the server and key manager on the other with an AMD EPYC 7313 CPU and 512 GB RAM. Implementation details are in Appendix A.

Dataset. We evaluate MASER on two commonly used datasets, MNIST and CIFAR-10 [7], [14]. We thereby show its effectiveness across different levels of task complexity. We simulated a cross-silo FL scenario with 5 clients and perform 25 global training rounds. We divide the training sets into non-overlapping subsets distributed among clients. For the IID, we evenly split them among all clients. For the non-IID, we divide them according to the Dirichlet distribution (α = 1.0) following prior research [7], [14].

Models. We use a modified LeNet-5 [30] and a modified Conv8 model (named "ConvNet") [31] for the MNIST and CIFAR-10, respectively. We set the modulus degree λ for our MKHE to 8192 and the learning rate to 0.01. We evaluate the aggregated model on the server side, using the test sets of MNIST and CIFAR-10 to measure the performance.

Baselines.

- **Vanilla FL.** We use FedAvg [15] and FedProx [27] in the IID and non-IID cases for model aggregation, respectively. Vanilla FL sets the upper bound for model performance with no privacy-preserving measures.

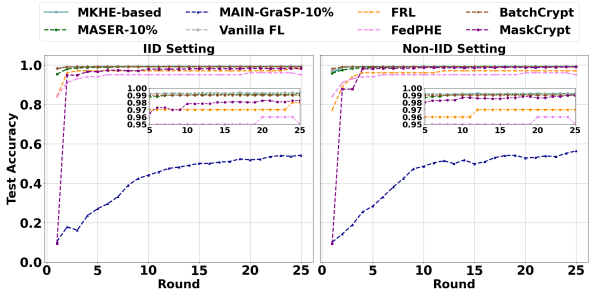

Fig. 2: Test accuracy across 25 FL rounds on MNIST

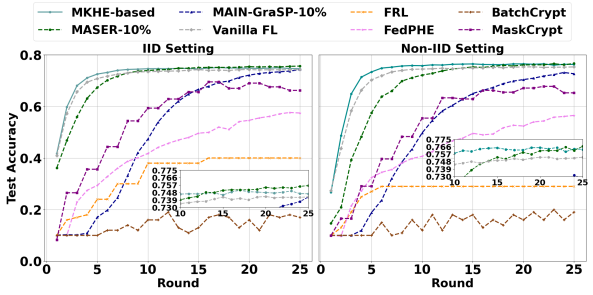

Fig. 3: Test accuracy across 25 FL rounds on CIFAR10

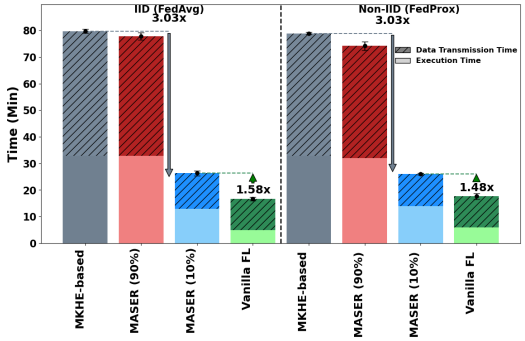

Fig. 4: Time overhead for 25 FL rounds on MNIST

- **SKHE-based FL.** We choose 3 state-of-art SKHE-based approaches with different optimizing techniques: FedPHE [7], BatchCrypt [8], and MaskCrypt [6]. SKHE-based methods risk privacy due to the shared key among clients unlike the strong privacy guarantee in MASER.
- **FRL [14].** FRL protects privacy by not sharing model parameters, but is slow to reach comparable accuracy in complex learning tasks. By comparing with FRL, we demonstrate the effectiveness of MASER in balancing privacy, model accuracy, and efficiency.
- **MKHE-based FL.** This baseline excludes MASER's sparsification. By comparing with this, we show that we significantly reduce the overhead of MKHE-based FL.
- **Masking at Initialization (MAIN)-GraSP.** MAIN-GraSP replaces MASER's sparsification with GraSP [21]. MAIN-GraSP computes and updates the global mask once at initialization. By comparing with this, we show that MASER's approach of updating the global mask in every round is necessary for enhanced performance.

B. Performance Evaluation

Figures 2 and 3 show the test accuracy measured across FL rounds for MASER and the baselines in IID and non-IID settings on MNIST and CIFAR-10, respectively. MASER and MAIN-GraSP choose the most important weights based on a predefined threshold κ. MASER-10% (MAIN-GraSP-10%) means that 10% of the most important weights are retained (before majority voting). We selected 10% to show MASER's competitive performance by using only 10% of the parameters. Importantly, we observe that threshold exceeding 5% does not significantly affect the accuracy. Less aggressive pruning slightly increases the accuracy with higher overhead (Figure 8 in Appendix B).

Figure 2 shows that all methods except MAIN-GraSP-10% converge to 95-99% accuracy by round 5 in the IID and by round 10 in the non-IID on the MNIST. In the IID, vanilla FL performs the best, reaching 99.37% and MASER-10% achieves 99.18% accuracy. In the non-IID setting, vanilla FL and MKHE-based FL perform the best reaching 99.20%, while MASER-10% reaches 99.10%. MAIN-GraSP-10% shows slower convergence, not reaching comparable performance in the IID and the non-IID by round 25. These results suggest that MASER-10% shows comparable performance to the best baselines despite using only a subset of the model weights for aggregation in both IID and non-IID settings. The minor difference can be attributed to MASER's pruning strategy, where less than 10% (due to majority voting after keeping 10% of parameters) of the most important parameters are used for aggregation.

Figure 3 shows that MASER-10% achieves the best accuracy among all baselines by round 25, 75.63% in the IID and 76.64% in the non-IID on CIFAR-10. MaskCrypt, FedPHE, FRL, and BatchCrypt perform significantly worse. This indicates limitations in their ability to handle heterogeneous data distribution in CIFAR-10, when utilizing relatively simple models such as ConvNet. MAIN-GraSP-10% reaches 74.25% accuracy in the IID and 72.65% accuracy in the non-IID by round 25. However, similar to the results on MNIST, MAIN-GraSP-10% converges much slower than MASER-10%. This is because MAIN-GraSP prunes a large portion of parameters at initialization, some of which would be among the important ones if pruning were delayed, leading to reduced learning capacity and model performance. In contrast, MASER recalculates the pruning mask in each round, adapting to the evolving model. This adaptive approach in MASER allows for more effective parameter selection and retention, contributing to its superior performance. Furthermore, these results confirm that MASER-10% better accommodates changing data patterns, converges faster, and attains higher accuracy.

C. Overhead Analysis

We restrict our analysis to the MKHE-based FL, considering their computation and communication costs, as SKHE-based methods are not directly comparable to the MKHE-based

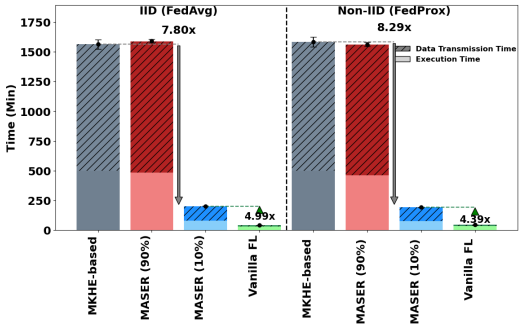

Fig. 5: Time overhead for 25 FL rounds on CIFAR-10

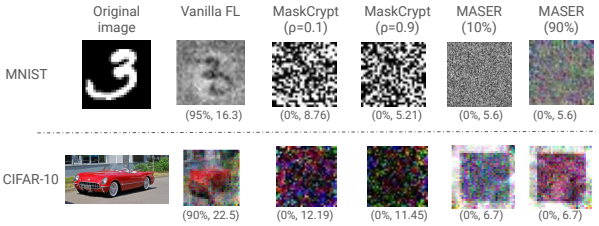

Fig. 6: Original and reconstructed images via the data reconstruction attack on the FL model

FL and MASER. For example, most of them simulate the server and clients as processes on the same machine, while we deploy them on two separate machines, incurring extra communication latency. We run experiments 5 times and report the average transmission and execution time. The error bar shows the standard deviation over the 5 runs.

Figures 4 and 5 show the total overhead, measured in minutes, incurred by various FL strategies in IID and non-IID settings on MNIST and CIFAR-10, respectively. We break down the total running time into two parts: data transmission (hatched) and execution (solid).

The figures show that vanilla FL has the fastest running time, which is expected as it does not employ encryption, being vulnerable to privacy attacks. Figure 4 shows that MKHE-based FL has the highest time overhead on average on MNIST, $\sim$ 58% of which is for data transmission in both settings. By pruning only 10% of the model weights, MASER-90% slightly reduces the total run time. Figure 5 shows that MKHE-based FL and MASER-90% yield the highest total run time in both settings on CIFAR-10. These results indicate that homomorphic operations are expensive computationally, and the encrypted ciphertext inflates data size, drastically increasing both the execution and data transmission time. In contrast, MASER-10% drastically reduces the total overhead. Compared to MKHE-based FL, MASER-10% achieves 3.03$\times$ reduction in the IID and non-IID settings on MNIST; $\sim$ 8 $\times$ reductions in the IID and in the non-IID settings, respectively on CIFAR-10. Importantly, MASER-10% only requires $\sim$ 1.5$\times$ and $\sim$ 5$\times$ the total running time of vanilla FL on MNIST and CIFAR-10, respectively. These results with analysis in Section V-B show that MASER-10% can maintain model performance and provide a strong privacy guarantee at the cost of a slight increase in the communication and computation overhead compared to vanilla FL. Note that the total overhead of MASER-10% is almost evenly divided by the transmission and execution time, meaning that our sparsification technique successfully minimizes both communication and computation overhead.

To show how MASER minimizes the overhead, we report the transmission data size after serialization in megabytes (MB) at different FL stages for MASER-10% and MASER-90% in Table I. We consider the size of the serialized local mask, aggregated global mask, encrypted slices, aggregated slices, and the decrypted global model parameters. By serialization, we mean the process of converting complex data structures, such as client masks, model parameters, and encrypted ciphertexts, into a format suitable for transmission. In MNIST with the modified LeNet-5, the impact of the pruning threshold on the encrypted slice size is minimal due to the model's compact size. For both 10% and 90% thresholds, all the parameters can fit into one slice, resulting in the same size of the encrypted slice and the aggregated ciphertext. In contrast, CIFAR-10 with the ConvNet shows a huge difference in the encrypted data size between the 10% and 90% thresholds. At the 10% threshold, only one slice is created, resulting in an encrypted slice size of 62.85 MB and an aggregated ciphertext size of 188.56 MB. However, at the 90% threshold, the larger number of parameters requires 10 slices, raising the encrypted slice size to 633.92 MB and the aggregated ciphertext size to 1901.8 MB. This proportional increase demonstrates how the number of slices (and thus the encrypted data size) expands with higher thresholds.

D. Privacy Leakage Analysis

We perform the reconstruction attack in [3] to assess the privacy leakage from the shared model updates in FL. We measure the Attack Success Ratio (ASR) and Peak Signal-to-Noise Ratio (PSNR) of reconstructed images, with higher values indicating greater privacy leakage [3]. Figure 6 shows results for MNIST and CIFAR-10 across different FL methods, including vanilla FL, MaskCrypt with varying encryption levels ρ, and MASER with different pruning thresholds. The tuple below each reconstructed image indicates the ASR (%) and PSNR value (in dB).

Vanilla FL shows significant privacy leakage with the high ASR (95% for MNIST and 90% for CIFAR-10) and high PSNR (16.3dB for MNIST and 22.5dB for CIFAR-10). In contrast, MASER-10% and MASER-90% yield strong privacy protection, having near-zero ASR and low PSNR (5.6dB for MNIST and 6.7dB for CIFAR-10), leaving no discernible pattern in the reconstructed image. Although MaskCrypt shows no significant privacy leakage at high encryption levels (e.g., $\rho = 0.9$), many unencrypted parameters are exposed at low encryption levels (e.g., $\rho = 0.04$ shown in Figure 6 of [6]). MASER, however, fully encrypts the sparsified model parameters, ensuring robustness to the reconstruction attack. Furthermore, MaskCrypt relies on SKHE, allowing a malicious client to decrypt the model updates of other clients with a

TABLE I: Size of the data after serialization (in MB)

Dataset	Threshold	Per-client mask	Global mask	Encrypted slices	Aggregated slices (enc.)	Global model
CIFAR10 (ConvNet)	10%	11.67	23.34	62.85	188.56	23.33
CIFAR10 (ConvNet)	90%	11.67	23.34	633.92	1901.8	23.33
MNIST (modified LeNet-5)	10%, 90%	0.45	0.89	25.14	75.43	0.89

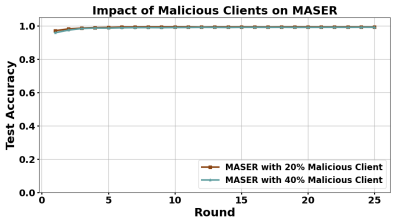

Fig. 7: Test accuracy on MNIST with malicious clients

shared key. In contrast, MASER ensures security due to each client's unique keys.

E. Robustness Analysis

Figure 7 shows the robustness of MASER with 20% and 40% of malicious clients, who sends bogus masks to the server to compromise the training process. It shows that MASER consistently maintains high and stable test accuracy throughout all rounds, showing negligible performance degradation even when 40% of clients are malicious.

VI. Related Work

We focus on HE-based FL as they are directly related. HE-based FL protects privacy by allowing the server to compute directly on encrypted data without decryption [32], [33]. SKHE has been widely used in privacy-preserving FL [6]–[9]. However, clients share the same keys in SKHE that can be exploited by malicious clients, making the system vulnerable [10]. MKHE has recently gained popularity in FL due to each client's unique keys enhancing security [10], [11]. However, the existing MKHE-based FL methods lack mechanisms to reduce the overhead. This is particularly important in FL, where maintaining an optimal balance between security and efficiency is crucial. We navigate this trade-off by ensuring that the key size remains large enough to guarantee security while still managing overhead.

Efficient HE. To address the high overhead in HE-based FL, several optimization methods have been proposed. BatchCrypt [8] quantizes the model parameters, packs them into smaller ciphertexts, and processes parameters in groups (or batches). FedPHE packs and encrypts sparsified model parameters [7]. However, they still incur high costs for large models. MaskCrypt partially encrypts sensitive model parameters based on the masks [6]. While it reduces the computational overhead, the partial encryption may expose certain model components, leading to potential privacy risks depending on the chosen thresholds [6]. Importantly, all of these rely on SKHE, which is vulnerable to a data reconstruction attack by malicious clients. MASER addresses these issues by combining MKHE with a consensus-based pruning strategy and a slicing technique. Unlike the prior work training a small model having a limited learning capability, we prune the model parameters before encryption, focusing only on the critical parameters, and reconstruct the original model after aggregation and decryption. This ensures a balance between accuracy, efficiency, and privacy.

Communication-Efficient FL. FedMask [34] employs structured sparse binary masks to improve the efficiency of FL. While effective in reducing the overhead, such approaches assume a shared parameter space and overlook privacy concerns. In FRL [14], server and clients exchange only important rankings of model parameters, which significantly reduces the communication overhead and improves privacy. However, FRL is slow to achieve comparable accuracy in complex learning tasks (Section V-B).

VII. Conclusion

MASER is a novel privacy-preserving, cross-silo FL framework integrating MKHE with consensus-based model pruning, selective encryption, and slicing to safeguard client data from an HBC server and malicious clients. Our experiments show that MASER achieves high accuracy, while significantly reducing overhead introduced by privacy-preserving techniques. These results highlight its potential for real-world deployment, where both privacy and efficiency are critical. While MASER was evaluated with the relatively small models due to limited resources, future work will focus on evaluating across various datasets and larger architectures to show its generalizability. Another potential direction includes developing more robust and adaptive consensus protocols to handle a majority of malicious clients.

Acknowledgment

This work was supported by the MITACS Accelerate program (No. IT36083) in partnership with HCLTech.

References

[1] C. Huang, J. Huang, and X. Liu, "Cross-silo federated learning: Challenges and opportunities," *arXiv preprint arXiv:2206.12949*, 2022.
[2] H. Yin, A. Mallya, A. Vahdat, J. M. Alvarez, J. Kautz, and P. Molchanov, "See through gradients: Image batch recovery via gradinversion," in *IEEE/CVF CVPR*, 2021, pp. 16 337–16 346.
[3] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *NeurIPs*, vol. 33, pp. 16 937–16 947, 2020.
[4] F. Tramer and D. Boneh, "Differentially private learning needs better features (or much more data)," in *ICLR*, 2021.
[5] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.
[6] C. Hu and B. Li, "Maskcrypt: Federated learning with selective homomorphic encryption," *IEEE TDSC*, 2024.
[7] N. Yan, Y. Li, J. Chen, X. Wang, J. Hong, K. He, and W. Wang, "Efficient and straggler-resistant homomorphic encryption for heterogeneous federated learning," in *IEEE INFOCOM*, 2024.

[8] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proc. of the 2020 USENIX Annual Technical Conference*, 2020.
[9] W. Jin, Y. Yao, S. Han, C. Joe-Wong, S. Ravi, S. Avestimehr, and C. He, "FedML-HE: An efficient homomorphic-encryption-based privacy-preserving federated learning system," in *Int'l Workshop on Federated Learning in the Age of Foundation Models*, 2023.
[10] Y. Cai, W. Ding, Y. Xiao, Z. Yan, X. Liu, and Z. Wan, "Secfed: A secure and efficient federated learning based on multi-key homomorphic encryption," *IEEE Trans. on Dependable and Secure Computing (TDSC)*, vol. 21, no. 04, pp. 3817–3833, jul 2024.
[11] J. Ma, S.-A. Naas, S. Sigg, and X. Lyu, "Privacy-preserving federated learning based on multi-key homomorphic encryption," *Int'l Journal of Intelligent Systems*, vol. 37, no. 9, pp. 5880–5901, 2022.
[12] J. Park, N. Y. Yu, and H. Lim, "Privacy-preserving federated learning using homomorphic encryption with different encryption keys," in *2022 13th ICTC*. IEEE, 2022, pp. 1869–1871.
[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NeurIPs*, vol. 28, 2015.
[14] H. Mozaffari, V. Shejwalkar, and A. Houmansadr, "Every vote counts: Ranking-Based training of federated learning to resist poisoning attacks," in *32nd USENIX Security Symposium*, 2023, pp. 1721–1738.
[15] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *PMLR*, 2017, pp. 1273–1282.
[16] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *Journal of the ACM (JACM)*, vol. 60, no. 6, pp. 1–35, 2013.
[17] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *2019 ACM SIGSAC Conf. on Computer and Communications Security*, 2019, pp. 395–412.
[18] S. Babakniya, S. Kundu, S. Prakash, Y. Niu, and S. Avestimehr, "Revisiting sparsity hunting in federated learning: Why does sparsity consensus matter?" *Trans. on Machine Learning Research*, 2023.
[19] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv e-prints*, vol. arXiv:1902.09574, 2019.
[20] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2018.
[21] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," *arXiv preprint arXiv:2002.07376*, 2020.
[22] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *NeurIPs*, vol. 33, pp. 6377–6389, 2020.
[23] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2024.
[24] T. L. Gez and K. Cohen, "A masked pruning approach for dimensionality reduction in communication-efficient federated learning systems," *arXiv preprint arXiv:2312.03889*, 2023.
[25] Y. Aono *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Trans. on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.
[26] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *2017 ACM CCS*, 2017, pp. 1175–1191.
[27] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proc. of Machine learning and systems*, vol. 2, pp. 429–450, 2020.
[28] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *Int'l conf. on machine learning*. PMLR, 2020, pp. 5132–5143.
[29] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," *NeurIPs*, vol. 33, pp. 7611–7623, 2020.
[30] M. Wortsman, V. Ramanujan, R. Liu, A. Kembhavi, M. Rastegari, J. Yosinski, and A. Farhadi, "Supermasks in superposition," *NeurIPs*, vol. 33, pp. 15 173–15 184, 2020.
[31] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, "What's hidden in a randomly weighted neural network?" in *IEEE/CVF CVPR*, 2020, pp. 11 893–11 902.
[32] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *41st ACM Symp. on Theory of computing*, 2009, pp. 169–178.
[33] N. P. Smart and F. Vercauteren, "Fully homomorphic encryption with relatively small key and ciphertext sizes," in *Int'l Conf. on Practice and Theory in Public Key Cryptography*, 2010, pp. 420–443.
[34] A. Li, J. Sun, X. Zeng, M. Zhang, H. Li, and Y. Chen, "Fedmask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking," in *Proc. of the 19th ACM conf. on embedded networked sensor systems*, 2021, pp. 42–55.
[35] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão *et al.*, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.
[36] T. Kim, H. Kwak, D. Lee, J. Seo, and Y. Song, "Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition," in *2023 ACM CCS*, 2023, pp. 726–740.

APPENDIX A

Implementation Details: We implemented MASER on top of the Flower [35] FL framework, and built the machine learning models using PyTorch. MASER utilizes MKCKKS as the MKHE solution [36], but we only found a Golang-based implementation of MKCKKS[2]. Since a Python-based MKCKKS library is necessary to integrate with Flower framework and PyTorch, we implemented a Python wrapper for the Golang-based MKCKKS library. Specifically, we designed a set of functions and data structures for the HE-related operations and used the *cgo* library to export these functions into a C-style dynamic link library (DLL). We imported the DLL into Python and used the *ctypes* library to convert the corresponding data structures and functions to Python. Note that *cgo* does not support the map data structure used in the Golang implementation, so we substituted the map data structure with Golang arrays. Finally, we implemented the public key aggregation in xMKCKKS [11]. We release our Python-based xMKCKKS library and MASER code base in the following repository: https://github.com/sustainable-computing/MASER.

APPENDIX B

Figure 8 shows the test accuracy of MASER across different pruning levels in IID and Non-IID. Despite the varying levels of sparsity, the curves remain close, meaning that our pruning strategy, combined with how we update important parameters, effectively preserves the important information needed for learning, even at high levels of sparsity.

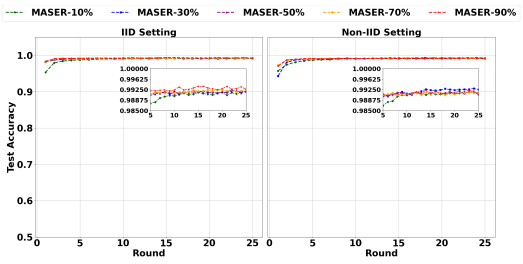

Fig. 8: Test accuracy on MNIST with different thresholds

[2]https://github.com/SNUCP/MKHE-KKLSS