# Octopus: A Robust and Privacy-Preserving Scheme for Compressed Gradients in Federated Learning

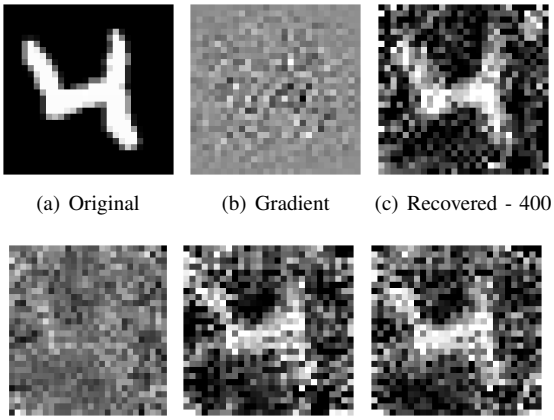
Wenxiu Ding*, *Member, IEEE,* Yuxuan Xiao, Zheng Yan, *Fellow, IEEE,*
Ciwei Chen, Yuxuan Cai, and Xuyang Jing, *Member, IEEE*

***Abstract*—Federated learning is a distributed machine learning framework that allows multiple parties to collaboratively train a shared model without the need to share their original data with a central server. This approach minimizes the risk of data leakage and effectively addresses the challenge of isolated data silos. However, it necessitates multiple rounds of interactions to transmit models or gradients between the client and the server, leading to a significant communication cost and private data leakage. Consequently, some schemes compress the transmitted models or gradients through model pruning or knowledge distillation. However, they often overlook the privacy implications of compressed gradients or models, potentially resulting in privacy breaches. Additionally, they are susceptible to client disconnections, resulting in incomplete or delayed model updates. To address these challenges, this paper proposes Octopus, a robust and privacy-preserving scheme for compressed gradients in federated learning. Octopus employs Sketch to compress gradients and embeds masks for the compressed gradients, thereby safeguarding the gradients while concurrently reducing communication overhead. Moreover, we propose an anti-disconnection strategy to support model updates even in situations where some clients are disconnected. Lastly, we carry out comprehensive security and convergence analyses, along with extensive performance evaluations, demonstrating Octopus's robustness, stability, and efficiency over existing schemes.**

***Index Terms*—federated learning, privacy preservation, anti-disconnection, sketch.**

## I. Introduction

FEDERATED learning (FL) [1], a distributed machine learning paradigm, has emerged as a solution for addressing the privacy and security concerns associated with centralized training on sensitive data. It enables multiple participants such as personal devices and edge servers, to collaborate in training a shared model without outsourcing their original datasets. This decentralized approach minimizes the risk of data exposure, and addresses the predicament of isolated data silos.While FL alleviates direct data exposure by keeping raw data on client devices, its iterative communication mechanism incurs substantial overhead—a problem exacerbated by the growing complexity of modern machine learning models. Numerous studies have focused on reducing the communication overhead in FL [2]–[6]. These approaches employ techniques such as gradient quantization and model compression to mitigate the communication burden. However, these compression and quantization methods are typically implemented under a mutual agreement between clients and the server, a framework that inherently grants the server potential access to client data, thereby posing significant privacy risks. As illustrated in Fig. 1, we apply the gradient matching attack described in [7] to the MNIST dataset to reconstruct the image. The attack is executed on a workstation equipped with a single NVIDIA RTX4080 GPU, and the timing is measured using wall-clock time. After 400 iterations within 2 minutes, this attack successfully generates an approximately restored image from quantized gradients. This result demonstrates that even when gradients are quantized and compressed, the original information remains susceptible to reconstruction. Consequently, this finding highlights that minimizing communication overhead alone is insufficient; it is equally critical to implement robust privacy-preserving mechanisms to safeguard compressed data. However, existing privacy-preserving technologies face substantial challenges when directly applied into communication-efficient FL frameworks.



(a) Original (b) Gradient (c) Recovered - 400

(d) Quantized (e) Recovered - 100 (f) Recovered - 400

Fig. 1: Reconstruction of the Image via Gradient Matching Attack: (a) The original image; (b) Gradient; (c) Recovered image from (b) after 400 iterations; (d): Quantized gradient; (e): Recovered image from (d) after 100 iterations; (f): Recovered image from (d) after 400 iterations

Firstly, there is no suitable solution that ensures the pri-

The work is sponsored by the Natural Science Basic Research Program of Shaanxi under Grant 2023-JC-YB-500; in part by the National Natural Science Foundation of China under Grants 62072351, 62002273, and U23A20300; in part by the Key Research Project of Shaanxi Natural Science Foundation under Grant 2023-JC-ZD-35. (Corresponding author: Wenxiu Ding)

Wenxiu Ding, Yuxuan Xiao, Zheng Yan, Ciwei Chen, and Xuyang Jing are with the School of Cyber Engineering, Xidian University, Xi'an, China
E-mail: wxding@xidian.edu.cn, xyx@stu.xidian.edu.cn, zyan@xidian.edu.cn, ccw@stu.xidian.edu.cn, jingxuyang@xidian.edu.cn

Yuxuan Cai is with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
E-mail: yxcai-kaka@sjtu.edu.cn

vacy of compressed data while achieving an optimal balance between communication efficiency and model accuracy. Homomorphic encryption (HE) [8]–[11] ensures data privacy but incurs substantial storage and communication overhead due to the large size of ciphertext. Moreover, single-key HE requires all clients to encrypt and transmit data using the same encryption key, making it unable to prevent the server from probing and inferring client data. Differential privacy (DP) [12]–[15] introduces noise to safeguard data, often at the cost of reduced model accuracy. Secure multi-party computation (SMC) [16]–[18] offers robust security guarantees but suffers from interaction costs and vulnerability to client disconnections. While Trusted Execution Environments (TEE) [19] mitigate encryption-related time costs, their reliance on hardware trust assumptions raises additional concerns. When these privacy-preserving methods are directly applied to compressed data, their limitations are further amplified. HE's high storage and communication overhead, DP's accuracy degradation, and SMC's susceptibility to client disconnections often outweigh the efficiency gains of compression. These challenges highlight a critical gap in current approaches: the inability to simultaneously achieve both privacy protection and communication efficiency when model updates are compressed before transmission.

Secondly, the robustness of privacy-preserving FL schemes is susceptible to client disconnection during training. Some SMC-based FL schemes [17], [18] employ mask to conceal client information and cope with client disconnection by restoring the client's mask from other clients' sub-secrets. But it incurs new security risk for the client disconnected owing to network congestion. Once the network congestion is resolved and the client's masked message reaches the server, it can use the sub-secrets to reconstruct the original message. In some DP-based FL schemes [12]–[15], to reduce the impact of disconnection, other clients re-adjust the added noise for the remaining clients to ensure that the overall privacy protection level remains unchanged. But it may affect the accuracy performance of the model. Therefore, it is crucial to mitigate the negative effects caused by client disconnections.

To address these challenges, we propose Octopus, a FL framework that leverages Sketch for gradient compression. Based on the combination of Sketch and single-mask mechanisms, Octopus achieves the secure aggregation of masked compressed gradients, thereby diminishing the risk of privacy leakage via approximate recovery of the original gradients. Moreover, it enables approximate reconstruction of the aggregated data on the client side, thereby mitigating the degradation in model accuracy typically caused by compression. To further enhance robustness, Octopus introduces a dropout-resilient strategy based on a two-phase mechanism, which stabilizes model updates, preserves consistent data distribution, and alleviates the aggregation inconsistencies introduced by direct masking.

We also conduct a comprehensive security analysis to prove that Octopus resists up to $n-2$ colluding clients, where $n$ is the number of clients participating in this round of training, and perform a convergence analysis to verify its training stability. To access practical performance, we evaluate Octopus on two standard FL benchmarks [1], [20] in terms of model accuracy, computation cost and communication overhead, and further compare it with HE-based [21] and DP-based [14] FL schemes to demonstrate its superiority. The contributions of this paper can be summarized as follows:

- We propose Octopus, the first FL framework that combines lightweight compressed gradient protection with robustness against both disconnection and collusion attacks.
- We integrate Sketch-based gradient compression with an optimized masking mechanism to achieve efficient and privacy-preserving model training.
- We design a dropout-resilient strategy with a two-phase mechanism to eliminate the influence of the mask on the aggregation result and mitigate the impact of client disconnections.
- We conduct comprehensive security and convergence analyses, along with extensive performance evaluations, demonstrating Octopus's robustness, stability and its advantages over existing HE-based and DP-based FL schemes.

The remainder of this paper is structured as follows. Section II provides an overview of related work. Section III covers the preliminaries, system and security models, and introduces the notations used throughout the paper. In Section IV, we present the detailed design of Octopus. Section V analyzes the security, disconnection, and convergence of Octopus. In Section VI, we present experimental results to demonstrate the superior performance of our proposed scheme. Finally, Section VII concludes the paper.

## II. Related Work

In this section, we assess several existing FL schemes in terms of three key factors: minimizing communication overhead, ensuring privacy, and addressing issues related to client disconnection.

### A. Communication-efficient Federated Learning

Reducing the communication overhead in FL typically involves two approaches. One approach is to compress the transmitted parameters to minimize the communication volume in each round. For instance, FedPAQ [3] proposes a low-precision quantization scheme, in which clients upload quantized gradients to the server. This approach effectively reduces communication overhead. However, by performing direct aggregation of low-precision gradients on the server, FedPAQ introduces a loss of model accuracy, which negatively impacts the performance of federated training. In the case of FetchSGD [4], each participating device computes a gradient vector and maps it to a low-dimensional space, resulting in reduced communication costs. However, the server retains the capability to perform an unsketch operation, which enables the approximate reconstruction of client data and thus poses a risk of privacy leakage. FedKD [22] utilizes an adaptive mutual distillation technique that allows for reciprocal learning between a teacher model and a student model at each client, where the distillation intensity is governed by their prediction

correctness. To reduce communication overhead, FedKD uploads gradients decomposed via Singular Value Decomposition (SVD). While this approach significantly lowers communication costs, it also increases the computational burden on local clients. CMFL [23] excludes irrelevant information and prevents it from being uploaded, thereby reducing overhead. However, it employs a fixed threshold to determine which updates to discard, which may limit its adaptability and effectiveness as training progresses. Moreover, this approach of directly excluding certain client updates can lead to imbalances in the data distribution aggregated by the global model, potentially degrading its generalization performance.

Another approach is to reduce the number of global communication rounds and lower the overall communication volume by accelerating the convergence speed of the training models. The classic solution is FedAvg [1], which performs global aggregation after a certain number of local iterations. This helps reduce communication rounds and achieve faster convergence. Some variants [20], [24] derived from FedAvg speed up the convergence of the global model by increasing the penalty or correction term. FedCS [25] introduces a deadline for the selection of heterogeneous clients. The mobile edge computing operator chooses clients in a way that allows the server to aggregate a maximum number of client updates within a predetermined time frame. By doing so, the overall training process becomes more efficient, resulting in reduced model training time. However, one notable drawback of this approach is that some clients may be persistently excluded from participating in the FL process.

### B. Privacy-preserving Federated Learning

Many technologies have been applied to achieve privacy-preserving FL, such as homomorphic encryption, secure multi-party computation and differential privacy. Hardy et al. [8] used the Paillier encryption scheme [26] to encrypt the model parameters. FedOpt [27] combines homomorphic encryption and differential privacy to protect client privacy while employing sparse compression algorithms to reduce communication overhead. However, the use of identical encryption parameters by all clients introduces a potential risk: the collusion between the clients and the server could lead to private data leakage. Zhu et al. [28] implemented perturbed model compression to safeguard client data and employed model sparsification techniques to reduce communication overhead. However, because the compression matrix is uniformly generated by the server and distributed to each client, the server has the potential to access the clients' private data. Secure multi-party computing ensures the privacy of client data through carefully designed interaction protocols. One such method is pairwise additive masking [17], which allows clients to interact with each other to generate offset masks to hide real information. Each client adds two masks to its data before uploading. However, each client needs to share the generated seeds of the two masks with other clients through multiple rounds of communications, which incurs high communication overhead and is not able to resist collusion attacks. EPPRFL [29] employs sampling techniques to reduce communication overhead, and integrates optimized SMC to protect client privacy. Nonetheless, the framework relies on repeated interactions and computations between two servers and remains susceptible to collusion attacks. SEFL [30] leverages compressed sensing techniques to compress the uploaded models and reduce communication overhead, while employing SMC and functional encryption (FE) to ensure model privacy. However, this framework requires one central cloud server along with two auxiliary cloud servers to function properly. Differential privacy can also be applied to protect uploaded gradients or model parameters by inserting noise perturbations [15], but it leads to the loss of training accuracy.

TABLE I: COMPARISON WITH EXISTING SCHEMES

| Scheme | PT | CC | CO | DI |
|---|---|---|---|---|
| [1] | - | $\mathbb{O}(d^2)$ | $\mathbb{O}(d^2)$ | SS |
| [3] | - | $\mathbb{O}(d^2)$ | $\mathbb{O}(Q \cdot (d^2))$ | SS |
| [4] | - | $\mathbb{O}(nd^2)$ | $\mathbb{O}(mn)$ | SS |
| [15] | DP | $\mathbb{O}(d^2)$ | $\mathbb{O}(d^2)$ | SS |
| [16] | SMC+DP | $\mathbb{O}(k + d^2)$ | $\mathbb{O}(k \cdot d^2)$ | TD |
| [22] | - | $\mathbb{O}(d_s^2 + d_t^2)$ | $\mathbb{O}(h^2 + hd)$ | SS |
| [28] | PMC | $\mathbb{O}(d^2(C + h))$ | $\mathbb{O}(d^2(h + logC))$ | SS |
| [29] | SMC | $\mathbb{O}(dT_{prg} + dT_{mask})$ | $\mathbb{O}(\epsilon d^2 + N^2)$ | SS |
| [30] | SMC+FE | $\mathbb{O}(hd^2 + h^2 + k^3)$ | $\mathbb{O}(hd^2)$ | TD |
| Octopus | SMC | $\mathbb{O}(nd^2 + k^3)$ | $\mathbb{O}(mn)$ | - |

Notes: PT: Privacy Technique; CC: Computation Cost; CO: Communication Overhead; DI: Disconnection Issue; SS: System Stagnation; TD: Time Delay; PMC: Perturbed Model Compression; FE: Functional Encryption; $d$: model dimension; $Q \cdot ()$: quantization; $h$: dimension of dimensionality reduction matrix; $d_s$: student model dimension; $d_t$: teacher model dimension; $k$: bit length of keys; $n$: number of hashes; $m$: number of hash buckets; $C$: number of clusters; $N$: number of clients; $\epsilon$: sampling rate; $T_{prg}$: computation cost of generating a pseudo-random; $T_{mask}$: computation cost of masking a number; -: None

### C. Anti-disconnection Federated Learning

Client disconnection during FL can potentially lead to stagnation of the process, diminshed model generalization ability, or decreased model accuracy. One approach [31] is to enhance feature learning during local training to reduce the deviation of the local model, thus mitigating the impact of client disconnection on the global model. Asynchronous FL schemes [32], [33] afford participants the flexibility to execute model updates at distinct time intervals. However, this temporal variation in update timings may introduce challenges such as heighten model instability and prolong convergence periods.

As shown in Table I, we can see that FedAvg [1] does not perform any additional processing such as compression or privacy protection on the model, and only uploads the original model information. It is clear that some solutions [3], [4], [22] have focused on reducing communication overhead via gradient compression or client selection, yet they have neglected security considerations. Since Octopus prioritizes the protection of client data privacy, its communication performance may be slightly less efficient compared to the aforementioned solutions. Notably, client disconnection has been identified as potential catalysts for system stagnation. System stagnation denotes a critical bottleneck wherein the entire federated

learning process is suspended due to a client being offline, as the server must wait for the client's model update before proceeding. Conversely, time delay refers to the temporary unavailability of a client, which introduces latency but does not disrupt the overall operation of the system. Conversely, certain solutions [15], [16] have prioritized client data protection, which concurrently amplifying communication and computing costs. Several works attempt to balance privacy protection with communication efficiency. For instance, Zhu et al. [28] employs perturbed model compression to safeguard client data while leveraging sparsification to reduce transmission cost; however, the use of a server-generated compression matrix creates potential privacy risks. EPPRFL [29] integrates sampling and optimized SMC to achieve lower communication overhead with privacy guarantees, yet its reliance on repeated interactions between two servers makes it vulnerable to collusion. SEFL [30] combines compressed sensing with SMC and functional encryption to compress updates and protect privacy, but its requirement for a multi-server architecture significantly increases system complexity and deployment costs.

We can find that none of the previous research endeavors have successfully attained a desirable equilibrium across these dimensions, with the exception of our proposed Octopus.

## III. System Model and Preliminary

In this section, we introduce the system and security models, preliminary knowledge, and list the notations used in the paper.

### A. System Model

There are two types of entities in our system as shown in Fig. 2: cloud server and client. Their functions are described as follows:

1) *Cloud Server*: The cloud server as the service provider is responsible for gradient management, such as gradient table storage and global model updating.
2) *Client*: The clients are service consumers and involved in collaborative model training by employing its local dataset to train and update the model. To safeguard privacy, the client applies single mask technology to conceal its own gradient before uploading it to cloud server for subsequent model updates.

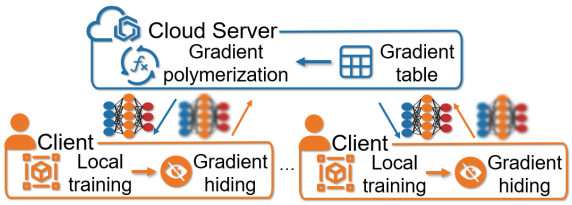



Fig. 2: System Model

### B. Security Model

In other gradient compression schemes [4], [5], there exists a foundational assumption of complete trust in the server and participating clients. However, this assumption represents an idealized state. In practical scenarios, the possibility of server and clients harboring ulterior motives cannot be discounted, introducing potential vulnerabilities and challenges in the application of these schemes.

In our system design, the cloud server is assumed to be honest but curious. According to the design of protocol, it is expected to execute the protocol with integrity. However, it may also engage in covert activities. These activities include the collection of client information and showing interest in accessing private data. Similarly, while most clients are assumed to be honest, we consider the possibility that up to $n-2$ clients may collude with the server to infer the private data of the remaining clients. Under this situation, we formally prove (in V-A) that as long as at least two clients do not collude with the server, their private data remains secure and cannot be reconstructed due to the existence of unknown random masks. Importantly, in our security model, colluding clients are assumed to follow the protocol honestly and upload valid model updates. This assumption is adopted because our work focuses on privacy leakage threats, not on data integrity verification or model poisoning attacks, which constitute a separate category of security challenges requiring orthogonal defense mechanisms. In other words, collusion in our setting refers to the sharing of locally observed information with the server to infer other clients' private data, rather than the submission of incorrect or malicious model updates.

### C. Preliminary Knowledge

The preliminary knowledge is provided here including federated learning and single mask scheme.

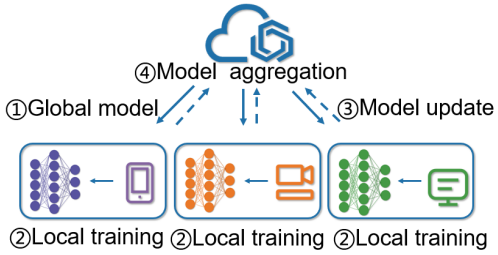



Fig. 3: Federated Learning Process

*1) Federated Learning:* Different from centralized machine learning, federated learning does not need to exchange data between data nodes, and only the parameters of the shared data model are exchanged between nodes. Thus, it enables different devices to learn machine learning models collaboratively without sharing data with centralized servers. The goal of federated learning is to minimize the average loss function:

$$min_\omega f(\omega) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(\omega), F_k(\omega) = \frac{1}{n_k} \sum_{i \in P_k} f_i(\omega) \quad (1)$$

where $F_k$ is the average loss function of all samples on the $k$-th device, $n$ is the sum of all samples, $n_k$ is the number of samples of the $k$-th device, and $P_k$ is the sample set of the $k$-th device. The process of federated learning is as shown in Fig. 3, which generally consists of the following four steps:

① Server selects a subset of existing clients, and each client downloads the current global model.

② Clients in the subset compute model updates based on local datasets.

③ Clients send model update to server.

④ Server aggregates to form a new global model.

FedAvg is a basic algorithm of FL, which takes the average value updated by each client as the global update, as shown in Algorithm 1. Its core idea is to distribute model parameters on local devices, and then perform model aggregation on the central server to update the global model.

**Algorithm 1** FedAvg

**Sever:**
  Initialize the global model $\omega^0$;
  **for** each round $t = 1, 2, ..., T$
    Select $k$ clients from $M$ clients with probability $P_k$ to participate in this round of training;
    Send the global model $\omega^t$ to the $k$ clients;
    Aggregate the model by $\omega^{t+1} = \frac{1}{k}\sum_{i \in k} \omega_i^{t+1}$;
**Client:**
  **for** each round $t = 1, 2, ..., T$
    Accept the global model $\omega^t$ sent by the server;
    Each client $i \in k$ updates $w_i^t$ for $E$ epochs of SGD on their own data with step-size $\eta$ to obtain $w_i^{t+1}$;
    Each client sends $w_i^{t+1}$ to sever;

FedProx, a quintessential algorithm in FL, addresses data distribution heterogeneity, as delineated in Algorithm 2. It accommodates the scenario of partially trained local models, referred to as $\gamma$-inexact solution. Concurrently, a proximal term is incorporated into the original loss function to enhance its effectiveness.

**Algorithm 2** FedProx

**Sever:**
  **for** each round $t = 1, 2, ..., T$
    Select $k$ clients from $M$ clients with probability $P_k$ to participate in this round of training;
    Send the global model $\omega^t$ to the $k$ clients;
    Aggregate the model by $\omega^{t+1} = \frac{1}{k}\sum_{i \in k} \omega_i^{t+1}$;
**Client:**
  **for** each round $t = 1, 2, ..., T$
    Each chosen device $k \in M$ finds a $\omega_k^{t+1}$ which is a $\gamma_k^t$-inexact minimizer of:
    $\omega_k^{t+1} \approx argmin_w h_k(\omega; \omega^t) = F_k(\omega) + \frac{\mu}{2}||\omega - \omega^t||^2$;
    Each client sends $w_k^{t+1}$ to sever;

In this paper, we take FedAvg [1] and FedProx [20] as examples for scheme presentation and experiment test. Our scheme can also support other algorithms, such as SCAFFOLD [24].

*2) Single Mask Scheme:* The single mask scheme [17] hides real information by adding some random numbers. Clients secretly negotiate a set of random numbers that can offset each other, and these random numbers are called pair-wise additive masks. After a certain mask is separately added to the message to be delivered, the server cannot know the value of the mask, and cannot obtain the real information of the client. Its core idea is described as follows.

A secret communication channel is established between clients $u$ and $v$ using the Diffie-Hellman key exchange protocol. They both share a secret random number $s_{u,v}$. Client $i$ sends an update value to the server, which is computed as $y_i = x_i - (s_{i,1} + s_{i,2} + \cdots + s_{i,i-1}) + (s_{i,i+1} + \cdots + s_{i,n})$. The server aggregates all the received updates, and the masks offset each other after aggregation. Each client $u$ calculates the hidden information $y_u$ using the mask and uploads it to the server.

$$y_u = x_u + \sum_{v \in \mathcal{U}; u<v} s_{u,v} - \sum_{v \in \mathcal{U}; u>v} s_{v,u} (modR) \tag{2}$$

The aggregated message obtained by aggregating the hidden information $y_u$ from all clients on the server side is equivalent to directly aggregating the unhidden message $x_u$.

$$\begin{aligned} \sum_{u \in \mathcal{U}} y_u &= \sum_{u \in \mathcal{U}} (x_u + \sum_{v \in \mathcal{U}; u<v} s_{u,v} - \sum_{v \in \mathcal{U}; u>v} s_{v,u}) \\ &= \sum_{u \in \mathcal{U}} x_u (modR) \end{aligned} \tag{3}$$

### D. Notations

For the sake of brevity, our symbol descriptions are placed in Table II.

TABLE II: NOTATION DESCRIPTION

| Notation | Explanation |
|---|---|
| $G$ | The group of order $q$ |
| $x_i$ | The random number selected by client $i$ |
| $s_{i,j}$ | The random number negotiated between client $i$ and client $j$ |
| $W$ | The weight matrix |
| $H^T$ | The transpose of Sketch matrix |
| $D_i$ | The dataset for client $i$ |
| $g_i^l$ | The gradient of client $i$ at iteration $l$ |
| $h_i^l$ | The hidden gradients of client $i$ at iteration $l$ |
| $A_i$ | The gradient of client $i$ in the gradient table |
| $g_{agg}^t$ | The aggregation gradient at iteration $t$ |
| $\mathcal{L}$ | The loss function |
| $\rho$ | The momentum coefficient |
| $r^t$ | The Error accumulation at iteration $t$ |
| $\mu^t$ | The momentum update at iteration $t$ |
| $\Delta_t$ | The approximate gradient at iteration $t$ |

## IV. Octopus Design

In this section, we introduce the detailed design of Octopus. The complete process is shown in Fig. 4. We divide the whole scheme into two phases, the pre-training phase and the training phase. In the following sections, we first introduce the specific algorithms of each phase, and then give the overall federated learning model update process.

### A. Pre-training Phase

The purpose of the pre-training phase is to obtain the gradient with masks of each client, which will help recover the masks of those disconnected clients during the training phase.

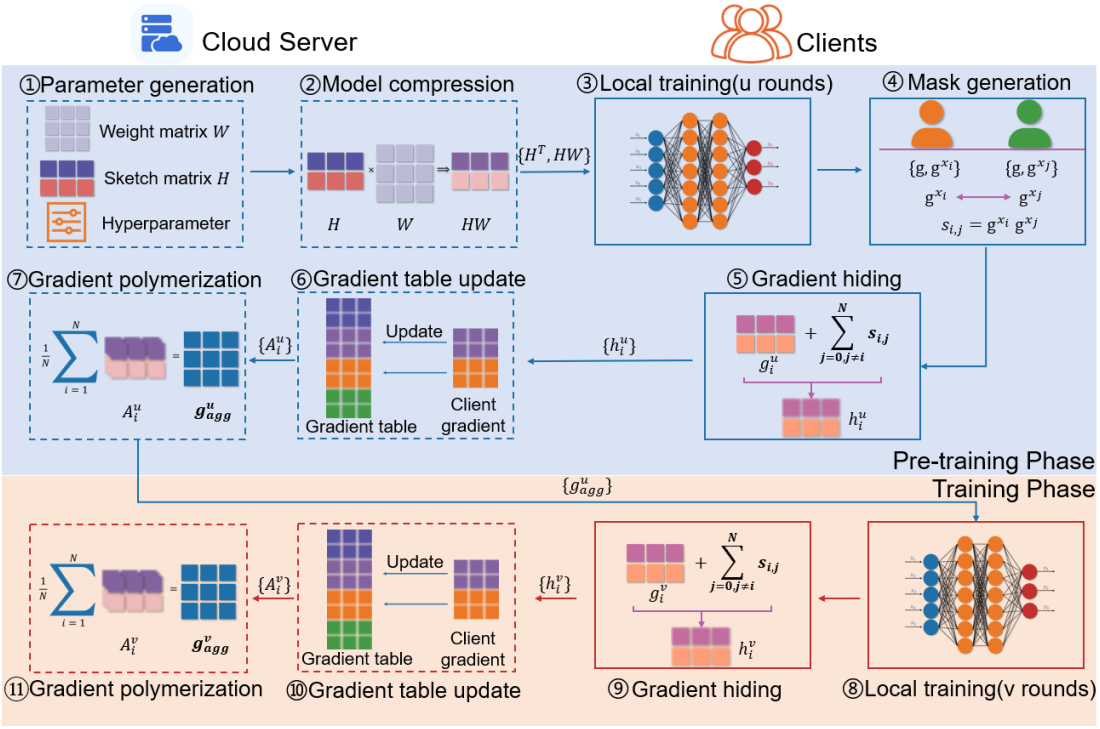



Fig. 4: Overall Training Process

In the pre-training phase, the selected client iterates locally for $u$ rounds, which is less than the local iteration $v$ of the training phase. The blue area in Fig. 4 shows the steps performed in the pre-training phase.

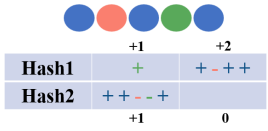
**Pre-training Phase:** There are seven steps in the pre-training phase, and the specific process of each step is as follows.

① *Parameter generation.* The Cloud server initializes the global model $W$, generates the Sketch matrix $H$, the learning rate $\eta$, the security parameter $k$, the number of clients $N$, the global iteration round $T$, the local iteration round $v$, the local test round $u$.

② *Model compression.* Model compression aims to reduce the communication overhead in federated learning. Count Sketch [34] is employed for model compression. It utilizes $k$ hash functions, denoted as $h_i : [d] \rightarrow [c]$ for $i \in k$. Additionally, each of these hash functions has a corresponding sign hash function denoted as $s_i : [d] \rightarrow \{\pm 1\}$ for $i \in k$. The purpose of this is to reduce bias. By randomly assigning +1 or -1, it can be ensured that when merging the values of multiple hash buckets, the positive and negative biases cancel each other out, thereby reducing the bias of the estimated value.

Here, we provide a simple example, illustrated in Fig. 5, to enhance understanding of this technology. Suppose we are required to estimate the frequency of blue data. We can multiply the values $(+2, +1)$ of the mapped positions by their respective sign hash values $(+1, +1)$, and then calculate the sum and average to obtain an estimate of the frequency. Hence, the frequency of the blue data is approximately 2, calculated as $(2*1+1*1)/2$. If there is no sign hash function, the frequency of blue data is approximately 5, calculated as $(4 + 5)/2$, deviating from the true value.

Each of the $k$ sketches is represented as a linear transformation $H_i x \in \mathcal{R}^c$, where $H_i \in \mathcal{R}^{c \times d}$ is referred to as a Count Sketch matrix. The global model is $W \in \mathcal{R}^{d \times d}$, when $c << d$, the communication overhead is reduced. The cloud server sends the transposed matrix $H^T$ of the Sketch matrix and the compressed model $HW$ to the clients.




| | Hash1,Sign1 | Hash2,Sign2 |
|---|---|---|
| | 2,+ | 1,+ |
| | 2,- | 1,- |
| | 1,+ | 1,- |

Fig. 5: The Process of Count Sketch

③ *Local training.* In local training, each client uses a local dataset to train the model. The client first downloads the compressed weight $HW$ from the cloud server, and then uses $H^T$ to approximately restore the original weight matrix $\hat{W}^l$. Then the client can calculate and upload the gradient:

$$g_i^l = \nabla_{H^l w^l} \mathcal{L}(\hat{W}^l, x_i) \tag{4}$$

If the neural network has L layers, then it conducts as follows:

$$x^l = \sigma(H^T(HWx^{l-1})) \tag{5}$$

for $1 \leq l \leq L-1$ and final output is $\hat{y} = a^T x^L$, where $a \in \mathcal{R}^d$ is output layer. After $u$ rounds of iterations, client $i$ gets the gradient $g_i^u$.

④ *Mask generation.* Clients mutually generate masks to protect gradients uploaded to the cloud server, which requires the Diffie-Hellman key exchange protocol [35]. Specifically, $G$ is a group of prime order $q$, along with a generator g; $x_i$ and $x_j$ are random numbers drawn by clients $i$ and $j$ from

$Z_q$; Client $i$ chooses a random integer $x_i$, calculates $X_i = g^{x_i}$ and sends it to client $j$. Client $j$ chooses a random integer $x_j$, calculates $X_j = g^{x_j}$ and sends it to client $i$. Then the random number between clients $i$ and $j$ is $S_{i,j} = (\mathrm{g}^{x_j})^{x_i}$, which helps in hiding the gradients.

⑤ *Gradient hiding.* Once each client has generated a mask, it will be added to the gradient for obfuscating the original gradient. Client $i$ combines the mask negotiated with other clients and adds it to gradient $g_i^u$ to obtain the hidden gradient $h_i^u$. Then the masked gradient will be uploaded to the cloud server.

$$h_i^u = g_i^u + \sum_{j=1}^{j=N} s_{i,j}(i \neq j) \tag{6}$$

⑥ *Gradient table update.* The cloud server receives the hidden gradient $h_i^u$ from client $i$ and puts it into the corresponding positions of the gradient table $A$, where $A_i^u = h_i^u$. The purpose is to save the hidden gradients of each client to prevent the meaningful information from being aggregated after the client disconnects. Since the gradient received by the server is masked, the original gradient cannot be approximately reconstructed.

⑦ *Gradient polymerization.* The most recent gradient of each client is stored in the gradient table, from which the cloud server fetches the gradients of each client and offsets the mask through aggregation to obtain the aggregated gradient. The cloud server retrieves the gradient of the current round from the gradient table for the aggregated clients and performs average aggregation to obtain the aggregated gradient $g_{agg}^u$.

$$g_{agg}^u = \frac{1}{n} \sum_{i=1}^{n} A_i^u \tag{7}$$

The cloud server sends aggregated gradients $g_{agg}^u$ to clients that have gradients in the gradient table.

After all the steps, the gradient table in the cloud server completely records all the gradient information of clients, which will be used for further training to cope with client disconnection. The detailed algorithm of Pre-training Phase in Octopus is presented in Algorithm 3.

**Algorithm 3** Pre-training Phase in Octopus

**procedure** MaskGen Algorithm
  **Input:** security parameter $k$, random number $x_i$ for user $i$, random number $x_j$ for user $j$;
  **Output:** mask $s_{i,j}$ between clients $i$ and $j$;
  $(G, \mathrm{g}, q) \leftarrow KA.param(k)$;
  $(x, \mathrm{g}^x) \leftarrow KA.gen(x)$;
  $s_{i,j} \leftarrow KA.agree(x_i, \mathrm{g}^{x_j})$;
**procedure** Pre-training Algorithm
  **Input:** local test round $u$, learning rate $\eta$, security parameter $k$, the number of clients $N$, weight matrix $W$, Sketch matrix $H$;
  **Output:** aggregation gradient $g_{agg}^u$;
  Cloud server randomly selects $n$ clients from $N$ clients;
  **for** all client $c_i \in c_n$ **do in parallel**
    $s_{i,j} \leftarrow$ MaskGen$(k, x_i, x_j)$;
    Download the Sketch matrix and the compressed global model $\{H^T, HW\}$;
    **for** $l = 1$ to $u$
      Calculate gradient $g_i^l = \nabla_{H^l w^l} \mathcal{L}(\hat{W}^l, x_i)$;
    **endfor**
    Hide gradient $h_i^u = g_i^u + \sum_{j=1}^{j=N} s_{i,j}(i \neq j)$;
    Send $h_i^u$ to Sever;
  **endfor**
  Cloud server updates the gradient table $A_i^u$ for client $i$;
  Aggregate gradients from gradient table:
    $g_{agg}^u = \frac{1}{n} \sum_{i=1}^{n} A_i^u$
  **endfor**

**Algorithm 4** Training Phase in Octopus

**procedure** Training Algorithm
  **Input:**local training round $v$, learning rate $\eta$, the aggregation gradient $g_{agg}^u$;
  **Output:** aggregation gradient $g_{agg}^v$;
  **for** all client $c_i \in c_n$ **do in parallel**
    **for** $l = u$ to $v$
      Calculate gradient $g_i^l = \nabla_{H^l w^l} \mathcal{L}(\hat{W}^l, x_i)$;
    **endfor**
    Hide gradient $h_i^v = g_i^v + \sum_{j=1}^{j=N} s_{i,j}(i \neq j)$;
    Send $h_i^v$ to Sever;
  **endfor**
  Cloud server updates the gradient table $A_i^v$ for client $i$;
  Aggregate gradients: $g_{agg}^v = \frac{1}{n} \sum_{i=1}^{n} A_i^v$;

### B. Training Phase

The training phase has more local iterations than the pre-training phase, but client disconnection during this phase will not affect gradient aggregation. The specific steps of the training phase are shown in the orange part in Fig. 4.

**Training Phase:** There are four steps in the training phase, and the specific process of each step is as follows.

⑧ *Local training.* Clients retrieve the latest global model, denoted as $g_{agg}^u$, from the cloud server and subsequently conduct $v$ rounds of training utilizing their local dataset. After $v$ rounds of iterations, client $i$ obtains the gradient, denoted as $g_i^v$.

⑨ *Gradient hiding.* Client $i$ adds the negotiated mask from other clients to its gradient $g_i^v$, resulting the hidden gradient $h_i^v$, which is then uploaded to the cloud server.

$$h_i^v = g_i^v + \sum_{j=1}^{j=N} s_{i,j}(i \neq j) \tag{8}$$

⑩ *Gradient table update.* The cloud server receives the hidden gradient $h_i^v$ from the client, and stores it in the corresponding position of the gradient table based on the client's ID. Two scenarios can occur at this point. First, if the client is not disconnected, the cloud server will utilize $h_i^v$ to update the corresponding value in the gradient table for that client. The second scenario arises when the client is disconnected, in which case the cloud server does not modify the gradient corresponding to that client in the gradient table.

⑪ *Gradient polymerization.* The cloud server retrieves the gradients of the current round from the gradient table for the aggregated clients, and then performs average aggregation to

obtain the aggregated gradient $g_{agg}^v$.

$$g_{agg}^v = \frac{1}{n}\sum_{i=1}^{n} A_i^v \tag{9}$$

The detailed algorithm of Training Phase in Octopus is introduced in Algorithm 4.

### C. Octopus Algorithm

Herein, we introduce the overall Octopus process as shown in Algorithm 5.

**Algorithm 5** Octopus

**Input:** global iteration round $T$, momentum parameter $\rho$, learning rate $\eta$;
**Output:** global weight $\{W_t\}_{t=1}^{T}$;
Init momentum term $\mu_0 = 0$;
Init error accumulation term $r_0 = 0$;
**for** $t = 1$ to $T$
  $\{g_{agg}^u\}^t$ = Pre-training$(u, \eta, k, N, W_t, H)$;
  $\{g_{agg}^v\}^t$ = Training$(v, \eta, \{g_{agg}^u\}^t)$;
  Update momentum: $\mu^t = \rho\mu^{t-1} + \{g_{agg}^v\}^t$;
  Update error feedback: $r^t = \eta\mu^t + r^t$;
  Approximate gradient: $\Delta_t$ = Top-k$(r^t)$;
  Accumulate error: $r^{t+1} = r^t - \Delta_t$;
  Update weight: $W_{t+1} = W_t - \Delta_t$;
**endfor**

Octopus is achieved through basic algorithms. With the basic parameter setting, the cloud server calls Algorithm 3 for pre-training to generate an aggregated gradient $\{g_{agg}^u\}^t$ with a local iteration number $u$. At this time, the cloud server side stores the hidden gradients of each client in gradient table. And then calls Algorithm 4 for training to obtain an aggregated gradient $\{g_{agg}^v\}^t$ with a local iteration number $v$. When the client disconnects during training, the cloud server can get $\{g_{agg}^u\}^t$ from the gradient table instead of $\{g_{agg}^v\}^t$ to weaken the disconnection impact. After the participants' gradients $\{g_{agg}^v\}^t$ is aggregated on the server side, the subsequent model update incorporates an error feedback and momentum scheme, analogous to methodologies found in other compression correction literature [4]. The error feedback term, $r^t$, enables the correction of inaccuracies introduced by our approximate gradient recovery, specifically addressing the discrepancies arising from using $\nabla_{w^l} f(H^T H W^l)$ as an approximation of $\nabla f(W^l)$. Upon constructing the complete error term, $r^t$, we extract Top-k components of $r^t$ to form $\Delta_t$. This results in an error-corrected gradient approximation that inherently includes the momentum term, thereby facilitating the stochastic gradient descent process. Since the Top-k sparsification [36] is not an unbiased estimate of the gradient, error accumulation is required to converge. The detailed convergence analysis can be found in Section V-D. When the global iterative round reaches $T$, the execution of the Octopus ends.

## V. Security Analysis

In this section, we undertake a comprehensive analysis on Octopus, focusing on three key aspects: security, disconnection, and convergence. Firstly, the security analysis is conducted to validate the robustness of the scheme in scenarios where a subset of users colludes with the server. This is followed by a disconnection analysis, which aims to ascertain that partial user disconnection does not result in the aggregation of meaningless information. Finally, a convergence analysis is performed to validate that, through interactions between the server and clients, the scheme achieves convergence.

### A. Security Analysis

In previous studies [7], it has been pointed out that gradient information carries the risk of privacy leakage, and thus it is necessary to protect it when uploading. In this work, we adopt the approach of adding masks to the gradients(as described in III-C2). We propose two theorems to analyze and prove the security of Octopus under potential collusion scenario as follows.

**Theorem 1.** *Based on the pairwise additive property [17], masks are uniformly randomly distributed, so clients' data hidden by masks is also uniformly random. That is, a third party cannot distinguish the mask from the original user data from the hidden data.*

*Proof.* In our proposed Octopus, let $M$ represent our client set containing $n$ clients, where each client $i$ has a value $x_i$. The pairwise mask $S_{i,j}$ is uniformly random and satisfies $S_{i,j} = -S_{j,i}$. The masked data for user $i$ is given by $y_i = x_i + \sum_{j\in M\setminus\{i\}} S_{i,j}$. We aim to prove that when a user's value is combined with uniformly random pairwise masks, the resulting value is uniformly random, provided the sum is preserved. For $n = 1$, the statement is trivially true, as $x_i$ is uniformly random without any additional masking. Assume the conclusion holds for $n = k$; that is, for $|M| = k$, the user data, when masked, remains uniformly random. For $n = k+1$, consider one user $i_{k+1}$ separately, and let the remaining $k$ users form the set $M'$. By the inductive hypothesis, the masked data for the $k$ users in $M'$ is uniformly random, and their sum remains unchanged. User $i_{k+1}$ generates a new mask $s_{i_{k+1},j}$ with each user $j \in M'$, and the new masked value for each user $i \in M'$ is computed as:

$$y_i = x_i + \sum_{j\in M'\setminus\{i\}} S_{i,j} + S_{i,i_{k+1}} \tag{10}$$

The masked value for user $i_{k+1}$ is:

$$y_{i_{k+1}} = x_{i_{k+1}} + \sum_{j\in M\setminus\{i_{k+1}\}} S_{i_{k+1},j} \tag{11}$$

Since the newly generated masks $S_{i_{k+1},j}$ are uniformly random and adhere to $S_{i_{k+1},j} = -S_{j,i_{k+1}}$, the sum of these masked values remains unchanged. Therefore, by induction, the resulting masked data is uniformly random under the condition that the sum is preserved.

**Theorem 2.** *When no more than $n-2$ clients collude with the server ($n$ is the number of clients participating in this round of training), the data of the non-colluding clients are not at risk of exposure.*

*Proof.* Consider the following equations representing the masked gradients:

$$\begin{cases} y_1 = x_1 + S_{1,2} + \cdots + S_{1,i-1} + S_{1,i} + \cdots + S_{1,n} \\ y_2 = x_2 - S_{2,1} + \cdots + S_{2,i-1} + S_{2,i} + \cdots + S_{2,n} \\ \quad \vdots \\ y_i = x_i - S_{i,1} - \cdots - S_{i,i-1} + S_{i,i+1} + \cdots + S_{i,n} \\ \quad \vdots \\ y_n = x_n - S_{n,1} - \cdots - S_{n,i-1} - \cdots - S_{n,n-1} \end{cases}$$

When all clients except clients $i$ and $j$, collude with the cloud server to obtain the private information of other clients. Based on the equations, the masked gradients of clients $i$ and $j$ can be written as:

$$y_i = x_i + \sum_{i<k} S_{i,k} - \sum_{i>k} S_{k,i}$$
$$y_j = x_j + \sum_{j<k} S_{j,k} - \sum_{j>k} S_{k,j}$$

When all other clients collude with the cloud server, the cloud server can get:

$$x_i = y_i - \sum_{i<k(k\neq j)} S_{i,k} + \sum_{i>k} S_{k,i} - S_{i,j}$$
$$x_j = y_j - \sum_{j<k} S_{j,k} + \sum_{j>k(k\neq i)} S_{k,j} + S_{i,j}$$

Therefore, the cloud server cannot obtain $x_i$ or $x_j$ since $S_{i,j}$ is unknown. Hence, the privacy of clients $i$ and $j$ remains secure. From the above formula, it is evident that when $n-2$ clients collude with the server, the data obtained by the server is still mixed with the clients' masks. According to Theorem 1, the server cannot retrieve the original data from the uniformly distributed mask information. Consequently, when the number of clients colluding with the server is no more than $n-2$, clients' private data remains secure.

**Remark.** The security guarantee is based on the assumption that no more than $n-2$ clients collude with the server. In real-world federated learning systems, large-scale collusion is typically impractical due to the independence of participants and lack of mutual trust. Hence, the assumption is both reasonable and practically enforceable in federated settings.

### B. Disconnection Analysis

In privacy preservation schemes reliant on a single mask, a significant challenge arises with the issue of client disconnections. Once a client drops out, the cloud server cannot recover meaningful information during aggregation. In our design, during the pre-training phase, clients only need to iterate a few rounds $u$ locally, which does not require significant computing time or extensive computing power, allowing each client to complete the pre-training phase. Consequently, we can assume that no client disconnections will occur. Nevertheless, to enhance the robustness of our design, we also consider potential client disconnections during the pre-training phase. In such cases, we adopt a conservative fallback mechanism: if a client fails to complete and upload its masked gradients within a specified timeout period, the server will discard this client's participation in the current round and reinitialize the pre-training phase with the remaining clients. Since the process involves limited computation, this approach avoids significant delays and prevents incomplete gradient masking from compromising aggregation integrity. Subsequently, the cloud server's gradient table will store the masked gradient of each client. In the event of a client disconnection during the future training phase, the cloud server can still utilize the gradient table to recover meaningful aggregated gradients.

In the previous discussion, issues related to client disconnection due to uncontrollable factors, such as network disruptions, were examined. The focus now shifts to scenarios where clients voluntarily exit the federated learning network. Upon deciding to withdraw, a client is obligated to issue a formal withdrawal notice to the network consortium. Furthermore, it is incumbent upon the client to upload its final gradient set, completely masked, to the cloud server's gradient table. After the current aggregation cycle concludes, the cloud server is responsible for removing this client's gradient data from its database, effectively omitting the client from future iterations of federated learning.

### C. Disconnection Analysis

In privacy preservation schemes reliant on a single mask, a significant challenge arises with the issue of client disconnections. Once a client drops out, the cloud server cannot recover meaningful information during aggregation. In our design, during the pre-training phase, clients only need to iterate a few rounds $u$ locally, which does not require significant computing time or extensive computing power, allowing each client to complete the pre-training phase. Consequently, we can assume that no client disconnections will occur.

### D. Convergence Analysis

Several studies [37]–[39] have demonstrated that when the compression operation satisfies the $\tau$-contraction condition, SGD with biased compression ensures convergence:

$$C(x) \leq (1-\tau)\, x \tag{12}$$

In this section, it is postulated that the contraction property remains consistent along the optimization path. We introduce two critical assumptions. Assumptions 1 and 2 are fundamental for proving convergence properties of SGD-like algorithms. Additionally, various other methodologies, as indicated in [39], [40], have similarly employed these assumptions.

**Assumption 1** The objective function $f(W)$ is L-smooth. That is, for any $x, y \in R^{d\times d}$. We have:

$$||\nabla f(x) - \nabla f(y)|| \leq L||x-y|| \tag{13}$$

**Assumption 2** All stochastic gradients $g_i$ of $f(W_i)$ are unbiased and bounded:

$$\mathbb{E}g_t = \nabla f(W_t^{\ell}), \mathbb{E}||g_t||^2 \leq G^2 \tag{14}$$

Let the weight matrix be $W_t \in R^{d\times d}$, and using the sketch's recovery matrix $\widehat{W_t} = RW_t$, where $R = H^T H$. What we need to prove is that when iterating to round $T$,

$min_{t=1,2,\cdots,T}\mathbb{E}||f(\widehat{W_t})||^2$ is bounded. Applying the chain rule of differentiation, we obtain:

$$\frac{\partial L(\widehat{W_t})}{\partial W_t} = \frac{\partial L\left(RW_t\right)}{\partial RW_t}\frac{\partial RW_t}{\partial W_t} \tag{15}$$

It follows that: $\nabla_{W_t} L\left(\widehat{W_t}\right) = R^T \nabla_{RW_t} L\left(RW_t\right)$. For simplicity, let us denote the loss function as $f$, leading to:

$$\begin{aligned} ||\nabla_{W_t} f(RW_t)|| &= ||R\nabla_{RW_t} f(RW_t)|| \\ &\leq ||R^T||||\nabla_{RW_t} f(RW_t)|| \end{aligned} \tag{16}$$

Given that $R$ contains only $\{-1, 0, 1\}$ and each row has only $\frac{d}{c}$ non-zero elements, thus $R \leq \frac{d^2}{c}$, we derive:

$$||\nabla_{W_t} f(RW_t)|| \leq \frac{d^2}{c}||\nabla_{RW_t} f(RW_t)|| \tag{17}$$

Let $C(x) = Top-k(x)$, the gradient sequence encountered during optimization is $\tilde{W}_t = W_t - r_t - \frac{\eta\rho}{1-\rho}\mu_{t-1}$. Expanding this formula we get:

$$\begin{aligned} \tilde{W}_t &= W_{t-1} - C(\eta(\rho\mu_{t-2} + g_{t-1}) + r_{t-1}) \\ &+ C(\eta(\rho\mu_{t-2} + g_{t-1}) + r_{t-1}) \\ &- \eta(\rho\mu_{t-2} + g_{t-1}) - r_{t-1} - \frac{\eta\rho}{1-\rho}\mu_{t-1} \\ &= W_{t-1} - r_{t-1} - \eta g_{t-1} - \eta\rho\mu_{t-2} \\ &- \frac{\eta\rho}{1-\rho}(\rho\mu_{t-2} + g_{t-1}) \\ &= W_{t-1} - r_{t-1} - \eta(1 + \frac{\rho}{1-\rho})g_{t-1} \\ &- \eta\rho(1 + \frac{\rho}{1-\rho})\mu_{t-2} \\ &= W_{t-1} - r_{t-1} - \frac{\eta\rho}{1-\rho}\mu_{t-2} - \frac{\eta}{1-\rho}g_{t-1} \\ &= \tilde{W}_{t-1} - \frac{\eta}{1-\rho}g_{t-1} \end{aligned} \tag{18}$$

This simplification resembles the update process in SGD. Utilizing the L-smooth property of the loss function, we can deduce:

$$\begin{aligned} \mathbb{E}f(\tilde{W}_{t+1}) &\leq f(\tilde{W}_t) + \langle \nabla f(\tilde{W}_t), \tilde{W}_{t+1} - \tilde{W}_t \rangle \\ &+ \frac{L}{2}\left\|\tilde{W}_{t+1} - \tilde{W}_t\right\|^2 \\ &\leq f(\tilde{W}_t) - \frac{\eta}{(1-\rho)}\mathbb{E}\langle \nabla f(\tilde{W}_t), \mathbf{g}_t \rangle \\ &+ \frac{L\eta^2}{2(1-\rho)^2}\mathbb{E}\|\mathbf{g}_t\|^2 \\ &\leq f(\tilde{W}_t) - \frac{\eta}{(1-\rho)}\left\|\nabla f(W_t)\right\|^2 + \frac{\eta}{2(1-\rho)} \\ &\left(\left\|\nabla f(W_t)\right\|^2 + \mathbb{E}\left\|\nabla f(\tilde{W}_t) - \nabla f(W_t)\right\|^2\right) \\ &+ \frac{L\eta^2 G^2}{2(1-\rho)^2} \\ &\leq f(\tilde{W}_t) - \frac{\eta}{2(1-\rho)}\left\|\nabla f(W_t)\right\|^2 \\ &+ \frac{\eta L^2}{2(1-\rho)}\mathbb{E}\|\tilde{W}_t - W_t\|^2 + L\eta^2\sigma^2 \\ &= f(\tilde{W}_t) - \frac{\eta}{2(1-\rho)}\left\|\nabla f(W_t)\right\|^2 \\ &+ \frac{\eta L^2}{2(1-\rho)}\mathbb{E}\left\|\mathbf{r}_t + \frac{\eta\rho}{1-\rho}\mu_{t-1}\right\|^2 + L\eta^2\sigma^2 \end{aligned} \tag{19}$$

Next, we aim to prove that $\mathbb{E}\left\|\mathbf{r}_t\right\|^2$ and $\mathbb{E}\left\|\mu_{t-1}\right\|^2$ are bounded. Given:

$$\mathbb{E}||\mu_t||^2 = \mathbb{E}||\sum_{i=1}^{t}\rho^i g_i||^2 \leq \mathbb{E}||\sum_{i=1}^{\infty}\rho^i g_i||^2 \leq \left(\frac{d^2 G}{c(1-\rho)}\right)^2 \tag{20}$$

Concurrently, we also derive the following equation:

$$\begin{aligned} \mathbb{E}||r_{t+1}||^2 &= \mathbb{E}||\eta(\rho\mu_t + g_t) + r_t - C(\eta(\rho\mu_t + g_t) + r_t)||^2 \\ &\leq (1-\tau)\mathbb{E}||\eta(\rho\mu_t + g_t) + r_t||^2 \\ &\leq (1-\tau)\Big((1+\gamma)||r_t||^2 + (1+1/\gamma)\eta^2||\mu_t||^2\Big) \\ &\leq (1-\tau)\left((1+\gamma)||r_{t-1}||^2 + \frac{(1+1/\gamma)d^4\eta^2 G^2}{c^2(1-\rho)^2}\right) \\ &\leq \sum_{i=0}^{\infty}\frac{((1-\tau)(1+\gamma))^i(1+1/\gamma)d^4\eta^2 G^2}{c^2(1-\rho)^2} \\ &\leq \frac{\frac{d^4}{c^2}(1-\tau)(1+1/\gamma)\eta^2 G^2)}{1-((1-\tau)(1+\gamma))} \\ &\leq \frac{4d^4(1-\tau)\eta^2 G^2}{c^2\tau^2(1-\rho)^2} \end{aligned} \tag{21}$$

where $\gamma = \frac{\tau}{2(1-\tau)}$.

Thus, $min_{t=1,2,\cdots,T}\mathbb{E}||f(\widehat{W_t})||^2$ is bounded. This section presents a basic analysis of convergence. For an in-depth rigorous exploration, we recommend reading [4], [5]. The conceptual framework of our proof is closely aligned with these referenced works.

## VI. Performance Evaluation

In this section, experiments were conducted to validate Octopus in reducing communication overhead, compensating

for client disconnections, and ensuring privacy preservation. The experiments were structured as follows: Firstly, compression tests were executed on three distinct datasets, varying the degree of compression. Secondly, simulated client disconnection tests were conducted across three datasets to assess Octopus's ability to compensate for disconnection. Finally, we tested and compared the performance of Octopus with a federated learning scheme based on CKKS (CKKS-FL) [21] and one based on differential privacy (DP-FL) [14] to show its superiority. The resultant experimental findings unequivocally affirm the efficacy of Octopus in addressing these multifaceted challenges.The specific details of the experimental settings and performance are outlined as follows.

### A. Experiment Environment

Prior to the initiation of the experiment, we establish clear evaluation criteria. The principal criterion is accuracy, considered imperative for augmenting model precision in the context of federated learning. This is pivotal in illustrating Octopus's superior performance in maintaining data integrity and reliability under diverse operational conditions. The secondary criterion is efficiency, focusing on sustaining high accuracy while concurrently reducing parameter transmission overhead. This aspect is critical for demonstrating Octopus's enhanced capability in optimizing resource utilization while ensuring robust model performance.

For a comprehensive experimental analysis, we select two benchmark solutions: FedAvg and FedProx. As the basic solution for federated learning and one of the most commonly used federated learning solutions, FedAvg serves as a key benchmark. FedProx, with its unique capabilities in processing heterogeneous data, is also chosen as a benchmark solution. This selection aims to substantiate Octopus's versatility in adapting to diverse solutions in federated learning framework.

**Hardware and Software Settings.** We use the Pytorch [41] deep learning development framework and Python as the development language. Both the client and server components are emulated within a singular host machine. The experiment was conducted on a desktop with Intel(R) Core(TM) i7-14700K @ 3.40 GHz processor, and 48GB of running memory. The GPU is NVIDIA GeForce RTX4080.

**Hyperparameter Settings.** In the context of federated training, we configured a total of $N = 30$ clients to expedite the experimental process. Simultaneously, we set the global iteration number $T$ to 30. In each round, 50% of the clients are randomly selected to participate in training, ensuring that 15 clients are involved per round. In the subsequent experiments, we configured the pre-training and training rounds for $u$ as 5 and 10, respectively, and for $v$ as 15 and 10, respectively. We maintained a constant local iteration count of 20. This configuration aims to examine the impact of the pre-training rounds on disconnection. The learning rate was set at $\eta = 0.01$ to minimize the risk of convergence to local minima. We opted for a local batch size of 16, a decision aimed at accommodating clients with limited computational capabilities in real-world scenarios, ensuring that the video memory is not exceeded during local iterations.

**Dateset and Model Settings.** We chose MNIST [42], Fashion-MNIST [43] and CIFAR-10 [44]. These datasets vary in complexity, from straightforward single-channel images to more intricate multi-channel configurations, and are widely recognized in the field of image classification. Our hypothesis is that achieving strong performance on these varied datasets would be indicative of Octopus's potential efficacy in other image classification tasks. The model used for MNIST and Fashion-MNIST is a simple convolutional neural network, which contains two convolutional layers and two fully connected layers, and the convolutional kernel is 5*5. For the CIFAR-10 dataset, noted for its complexity, we employed the ResNet18 [45]. This architecture is renowned in the field for its effectiveness within the domain of residual networks, making it a suitable choice for our training purposes. To better simulate real-world heterogeneous data scenarios, all datasets were partitioned in a non-IID manner. Specifically, we adopt the Dirichlet distribution-based data partitioning strategy [46], which is commonly used in federated learning settings to control the degree of data heterogeneity among clients. For each dataset, we sample label distributions from a Dirichlet distribution with a concentration parameter $\alpha = 0.8$, ensuring that each client holds data samples from a limited number of classes and that the class distributions across clients are imbalanced.

### B. Performance Evaluation of Octopus

In order to verify the efficiency and robustness of Octopus, we conducted experiments from four perspectives. Firstly, we conducted a theoretical analysis of the communication overhead. Secondly, we tested the accuracy of model under different compression ratios. Thirdly, we compared the accuracy of Octopus across different disconnection ratios. Finally, we evaluated and compared the time and space overhead of Octopus with existing privacy-preserving FL schemes.

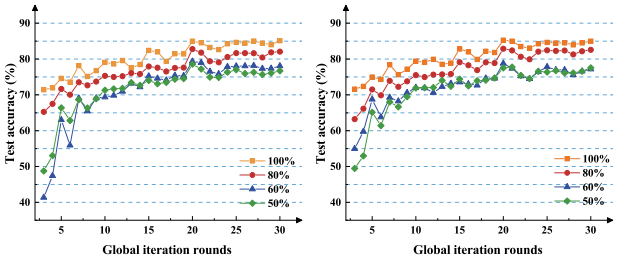
TABLE III: COMMUNICATION COMPARISON

| Compression Ratio | Communication Overhead | |
|---|---|---|
| | MNIST&Fashion-MNIST | CIFAR-10 |
| 100% | 174.72KB | 89.45MB |
| 80% | 139.78KB | 71.56MB |
| 60% | 104.83KB | 53.67MB |
| 50% | 87.36KB | 44.73MB |

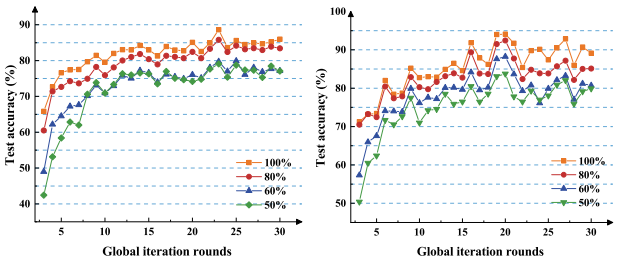
Notes: 100%: means no compression

**Communication Overhead.** The communication overhead includes the cost of model transmission and mask negotiation. Therefore, we initially conducted a theoretical evaluation of the model transmission under varying compression ratios. Subsequently, we perform a theoretical analysis of the communication overhead incurred by the mask in Octopus.

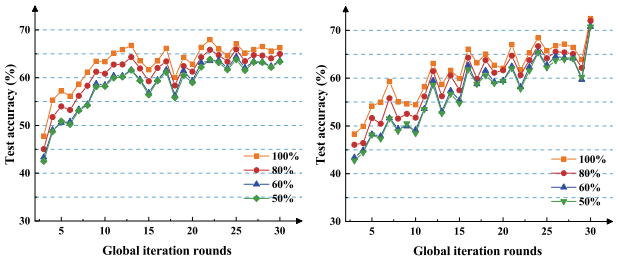
First, we provided the theoretical communication overhead under different compression ratios. Our scheme used 3 Sketches for compression, as [5]. We analyzed the communication cost of MNIST&Fashion-MNIST and CIFAR-10 as shown in Table III. In the context of the MNIST dataset, the chosen model comprises 21840 parameters, with each




(a) FedAvg-MNIST (b) FedProx-MNIST

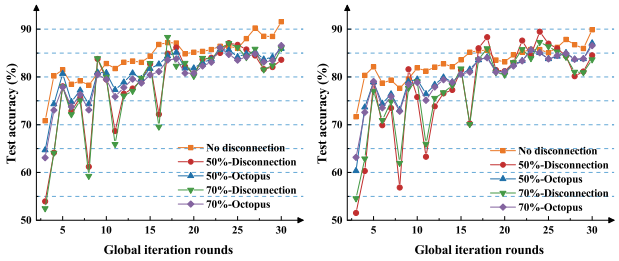



(c) FedAvg-FMNIST (d) FedProx-FMNIST

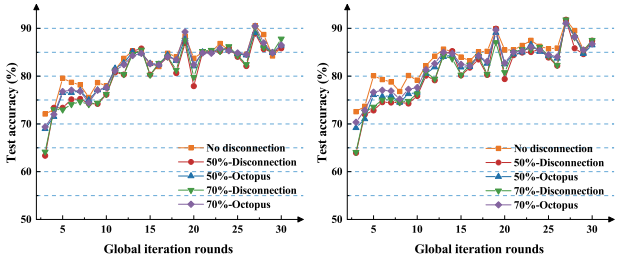



(e) FedAvg-CIFAR-10 (f) FedProx-CIFAR-10

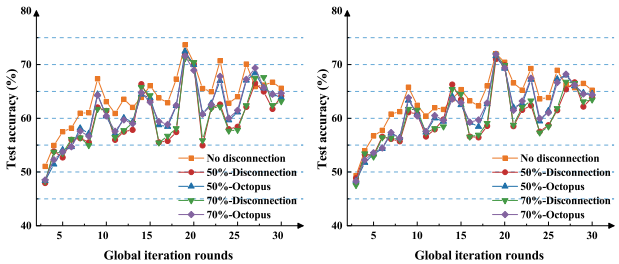
Fig. 6: Accuracy of FedAvg and FedProx under Different Compression Ratios




(a) FedAvg-MNIST (b) FedProx-MNIST




(c) FedAvg-FMNIST (d) FedProx-FMNIST




(e) FedAvg-CIFAR-10 (f) FedProx-CIFAR-10

Fig. 7: Accuracy of FedAvg and FedProx under Different Disconnection Ratios

parameter represented as a 64-bit floating-point number. This configuration results in a total model size of 174720 bytes. In a similar vein, the ResNet model employed for the CIFAR-10 dataset encompasses 11181642 parameters, culminating in a model size of 89453136 bytes. Upon applying a compression ratio of 80%, the transmitted parameter count is reduced to 17472, which corresponds to a communication overhead of 139776 bytes.

Next, we examined and analyzed the incremental communication overhead incurred during mask negotiation. There are $m$ clients, each engaging in the negotiation of a secret mask $S$, characterized as a 64-bit floating-point number. Consequently, client $i$ would need to send $(m-1) \times 8$ bytes of data to other clients as a mask. When the number of clients is 30, the additional communication overhead per client for the mask can be estimated at 232 bytes. This overhead is undoubtedly minimal compared to the data volume of the transmitted gradient information. Drawing from the preceding theoretical analysis, it can be deduced that our scheme results in a reduced communication overhead.

**Compression Ratio.** Different compression ratios lead to varying degrees of information loss. In order to evaluate the impact on model accuracy, we implemented four distinct compression ratios. The FedAvg and FedProx schemes were employed as baseline references for this analysis. Fig. 6 demonstrates that with ongoing training, the model achieves convergence and the accuracy stabilizes. At a compression ratio of 50%, the accuracy differs by less than 10% compared to the scenario without compression. This denotes that, even with a 50% reduction in communication overhead, the decline in accuracy is maintained within 10%. Notably, at an 80% compression ratio, this reduction in accuracy can be further limited to within 5%. Fig. 6 reveals that with FedProx employed as the federated learning algorithm, there is fluctuation in the accuracy curve. However, it is observed that at a 50% compression rate, despite a decrease in accuracy, this reduction remains below 10%. When utilizing the CIFAR-10 dataset, the observed decrease in accuracy is even more marginal. The experimental findings lead to the conclusion that our scheme achieves a 50% reduction in communication overhead while maintaining an accuracy decrease of no more than 10%, a threshold deemed acceptable. There is little difference between the two baseline solutions on the MNIST dataset because the MNIST dataset is relatively simple. It can be seen that FedProx performs better on the FMNIST and CIFAR-10 datasets because these two datasets are relatively complex and FedProx is better at addressing data distribution heterogeneity. FedProx needs to upload a proximal term, which requires higher precision. However, the compression operation result

TABLE IV: AVERAGE ACCURACY UNDER DIFFERENT CONDITIONS

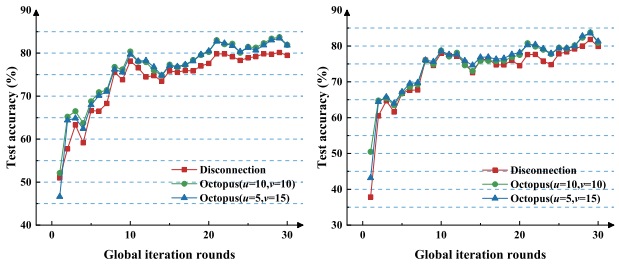
| Algorithm | Data Set | No Disconnection | 50% Disconnection Ratio | | | | 70% Disconnection Ratio | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Original | ***Octopus*** | Difference | $\mathcal{W}_r$ | Original | ***Octopus*** | Difference | $\mathcal{W}_r$ |
| FedAvg | MNIST | 81.60 | 74.56 | 78.19 | 3.63 | 51.5% | 74.39 | 77.09 | 2.70 | 37.5% |
| | Fashion-MNIST | 81.07 | 78.81 | 80.22 | 1.41 | 62.5% | 79.38 | 80.41 | 1.03 | 61.0% |
| | CIFAR10 | 62.40 | 58.40 | 59.81 | 1.41 | 35.2% | 58.61 | 59.87 | 1.26 | 33.2% |
| FedProx | MNIST | 80.61 | 73.76 | 77.13 | 3.37 | 49.1% | 73.78 | 76.88 | 3.1 | 45.3% |
| | Fashion-MNIST | 82.21 | 79.30 | 79.73 | 0.43 | 14.8% | 79.55 | 80.48 | 0.93 | 34.9% |
| | CIFAR10 | 61.85 | 58.63 | 59.74 | 1.44 | 34.3% | 58.78 | 59.81 | 1.03 | 33.5% |

Notes: $\mathcal{W}_r$: Weakening rate quantifies the effectiveness of the Octopus in mitigating disconnection impacts; Difference: The difference between Octopus and original scheme

in a loss of precision, so the negative impact of model compression on FedProx is slightly greater.

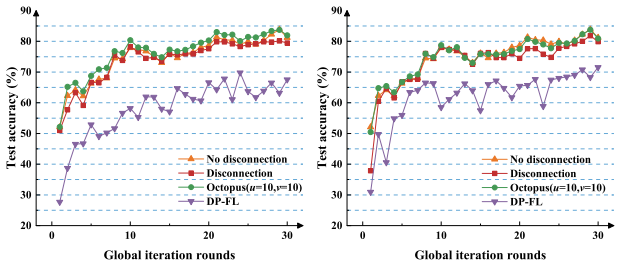
**Anti-disconnection Mechanism.** We conducted four sets of experiments to verify the performance of Octopus when clients are disconnected under different circumstances. In these experiments, no disconnection denotes the baseline solution under standard conditions, while disconnection refers to the scenario where the baseline solution is subjected to direct disconnection. The first set of experiments employed two distinct baseline solutions: FedAvg and FedProx, evaluated them across three datasets: MNIST, FMNIST, and CIFAR-10. The subsequent sets of experiments focused solely on FedAvg as the baseline solution and utilized MNIST dataset.

The first set aims to verify the accuracy of Octopus under various disconnection ratios (50%, 70%) to confirm that Octopus effectively mitigates the impact of disconnection issues. The number of pre-training rounds $u$ was set as 5, and the number of local training rounds $v$ was 15. Fig. 7 reveals that upon reaching a specific number of global iterations, there is a pronounced decline in accuracy. This phenomenon can be attributed to the disconnection of critical clients, those possessing pivotal model data. Octopus is implemented for anti-disconnection. It significantly mitigates the accuracy reduction. This outcome suggests that our solution effectively safeguards the integrity of the overall federated learning process. We conducted detailed statistical analyses to understand how the Octopus lessens the impact of disconnections. Initially, we tested the average accuracy. Herein, we define the disconnection weakening rate to measure the influence of anti-disconnection mechanism as below: $\mathcal{W}_r = (a - b)/a$ where $a$ denotes the discrepancy in accuracy between scenarios with no disconnection and with disconnection and $b$ is the variance between scenarios with no disconnection and those employing the Octopus scheme. This coefficient quantifies the effectiveness of the Octopus in mitigating disconnection impacts. Table IV reveals that Octopus significantly mitigates the negative impact on model accuracy under different disconnection ratios (50% and 70%). A consistent pattern is observed across FedAvg and FedProx algorithms applied to MNIST, Fashion-MNIST, and CIFAR10 datasets: the highest accuracy is noted with no disconnections, while a decline in accuracy following disconnections is effectively alleviated by the application of Octopus. Additionally, the weakening rate ($\mathcal{W}_r$) serves as a crucial metric, quantifying the effectiveness of Octopus across various scenarios, with most cases showing




(a) 20%-Disconnection (b) 50%-Disconnection

Fig. 8: Accuracy of Octopus under Different Pre-training Rounds



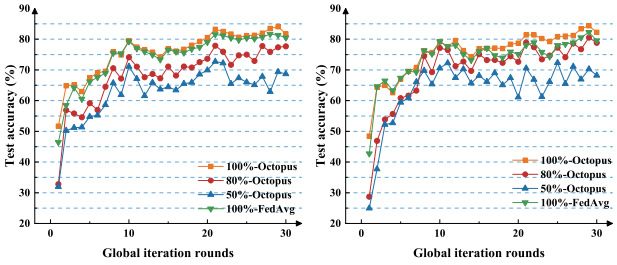

(a) 20%-Disconnection (b) 50%-Disconnection

Fig. 9: Accuracy Comparison under Different Disconnection Ratios without Compression

a weakening rate between 35% and 65%. Overall, these results underscore the potential value of Octopus in federated learning against disconnection.

The second set of experiments examined the influence of different hyperparameter settings (pre-training rounds $u$ and local training rounds $v$) on Octopus's performance. As illustrated in Fig. 8, Octopus consistently outperforms scenarios with disconnections irrespective of the values of $u$ and $v$ or the disconnection rates of 20% and 50%. On this simple MNIST dataset, different settings of $u$ and $v$ result in only minor differences.

The third set of experiments compared the performance of Octopus with DP-FL [14] under different disconnection ratios (20% and 50%) without compression. From Fig. 9, we can find that Octopus outperforms direct disconnection and, in some cases, even surpasses the performance when no disconnection occurs. We also observe that disconnection has a negative impact on DP-FL, with the accuracy rate differing




(a) 20%-Disconnection (b) 50%-Disconnection

Fig. 10: Accuracy Comparison under Different Disconnection Ratios with Gradient Compression

by approximately 10% compared to the scenarios without disconnections. Generally, the experiment results demonstrate that Octopus effectively avoid the accuracy degradation in DP-FL caused by disconnection clients.

The fourth set of experiments tested the impact of different compression rates (80% and 50%) on Octopus under different disconnection ratios (20% and 50%). From Fig. 10, it can be observed that Octopus outperforms the baseline solution, FedAvg, at disconnection rates of 20% and 50%. Despite achieving an 80% compression rate, Octopus maintains an accuracy gap close to 5% compared with the baseline solution. At a 50% disconnection ratio, its performance is on par with the baseline solution. These experiments validate the efficacy of Octopus across varying disconnection and compression ratios.

**Privacy Preservation.** We compared Octopus with CKKS-FL [21], DP-FL [14] and SEFL [30] in terms of the accuracy, training time and communication overhead. In a homomorphic encryption setting, the polynomial degree is set to 32768, with four coefficients modulo $\{60, 50, 50, 50, 60\}$, and a scaling factor of $2^{40}$, as recommended by the TenSEAL [47]. In a differential privacy setting, we incorporate Gaussian noise and allocate three distinct privacy budgets: $\epsilon = 1, 5, 10$. The failure probability of the privacy budget, denoted as $\delta$, is set to $1 \times 10^{-5}$. The maximum gradient norm is constrained to 10, following the recommendations of the Opacus [48]. In the SEFL setting, we use the Fernet module from the Cryptography library to perform symmetric encryption, simulating the lightweight encryption design adopted in SEFL.

As shown in Fig. 11, we tested the three privacy-preserving federated learning schemes on the MNIST dataset. In terms of accuracy, Octopus achieves the best performance, as it does not introduce approximate modifications to the model. The accuracy of DP-FL varies depending on the privacy budget settings. Specifically, when $\epsilon = 10$, the accuracy decreases by approximately 5%. Meanwhile, the accuracy of CKKS-FL is comparable to that of Octopus. The use of a larger polynomial modulus allows for higher-precision computations compared to basic configurations such as 8192. However, this setting is not without drawbacks, as it significantly increases storage overhead. In contrast, SEFL shows the largest fluctuation in accuracy among all schemes. This instability mainly results from the decoding error introduced by the All-Or-Nothing Transform (AONT) [49] and the approximate reconstruction of encrypted components. Although SEFL is lightweight in communication, these approximations can accumulate during training and lead to less stable convergence than other schemes. For the extra privacy overhead, DP-FL performs better in terms of time since DP-FL only needs to add Gaussian noise to the gradient. Octopus requires mask negotiation between clients. CKKS-FL requires additional operations such as key generation, encryption, and ciphertext aggregation, so it takes the longest time. Moreover, different privacy budget settings do not impact the time, so only the case of $\epsilon = 10$ is considered for comparison. Due to its lightweight encryption design, SEFL incurs slightly lower additional privacy overhead compared to Octopus; however, its overhead remains higher than that of DP-FL. Regarding communication overhead, DP-FL exhibits the smallest overhead. Octopus requires more communication overhead than DP-FL because it needs to upload a pre-trained model and negotiate random numbers with other clients. CKKS-FL, however, incurs the highest communication overhead because homomorphic encryption leads to substantial ciphertext expansion. Since SEFL only needs to encrypt the last element of the uploaded vector, its communication overhead is only slightly higher than that of DP-FL, and lower than that of both Octopus and CKKS-FL. The experimental results above show that Octopus converges faster than SEFL in fewer aggregation rounds while achieving higher accuracy. Compared to CKKS-FL, Octopus reduces both computational and communication overhead, and it delivers better model performance than DP-FL. Overall, Octopus achieves a better balance among model accuracy, computational cost, and communication overhead.

## VII. Conclusion

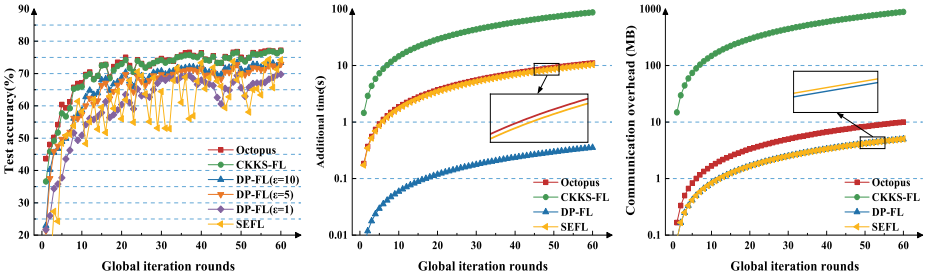
This paper proposed Octopus, a federated learning scheme that resists client disconnection, and used a mask to protect the compressed gradient. To the best of our knowledge, this is the first scheme that combines resistance against client disconnection with the preservation of compressed gradients. Concurrently, security analysis established that the scheme remained secure with at least two non-malicious clients. The experiments demonstrated the effectiveness of Octopus, reducing communication overhead and mitigating the effects of client disconnections, and showed its superiority in performance compared with existing federated learning schemes based on HE, DP or SMC. In the future, we will explore how to apply this framework to asynchronous federated learning to bring higher model accuracy with less communication overhead while protecting client privacy.

## References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[2] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, 2017.

[3] A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2021–2031.




(a) Accuracy (b) Extra privacy overhead (c) Communication overhead

Fig. 11: Performance Comparison of Octopus with SEFL, CKKS-FL, and DP-FL


[4] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8253–8265.

[5] T. Rabbani, B. Feng, M. Bornstein, K. Rui Sang, Y. Yang, A. Rajkumar, A. Varshney, and F. Huang, "Comfetch: Federated Learning of Large Networks on Constrained Clients via Sketching," *arXiv e-prints*, p. arXiv:2109.08346, Sep. 2021.

[6] Y. Chen, H. Vikalo, and C. Wang, "Fed-qssl: A framework for personalized federated learning under bitwidth and data heterogeneity," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 10, 2024, pp. 11 443–11 452.

[7] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," *Advances in neural information processing systems*, vol. 32, 2019.

[8] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv e-prints*, p. arXiv:1711.10677, Nov. 2017.

[9] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 2020)*, 2020.

[10] A. Madi, O. Stan, A. Mayoue, A. Grivet-Sébert, C. Gouy-Pailler, and R. Sirdey, "A secure federated learning framework using homomorphic encryption and verifiable computing," in *2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, 2021, pp. 1–8.

[11] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[12] P. Sun, X. Chen, G. Liao, and J. Huang, "A profit-maximizing model marketplace with differentially private federated learning," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1439–1448.

[13] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[14] R. C. Geyer, T. Klein, and M. Nabi, "Differentially Private Federated Learning: A Client Level Perspective," *arXiv e-prints*, p. arXiv:1712.07557, Dec. 2017.

[15] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[16] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proceedings of the 12th ACM workshop on artificial intelligence and security*, 2019, pp. 13–23.

[17] 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 *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[18] V. Mugunthan, A. Polychroniadou, D. Byrd, and T. H. Balch, "Smpai: Secure multi-party computation for federated learning," in *Proceedings of the NeurIPS 2019 Workshop on Robust AI in Financial Services*, vol. 21. MIT Press Cambridge, MA, USA, 2019.

[19] 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 Transactions on Dependable and Secure Computing*, 2023.

[20] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.

[21] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *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.

[22] C. Wu, F. Wu, L. Lyu, Y. Huang, and X. Xie, "Communication-efficient federated learning via knowledge distillation," *Nature communications*, vol. 13, no. 1, p. 2032, 2022.

[23] W. Luping, W. Wei, and L. Bo, "Cmfl: Mitigating communication overhead for federated learning," in *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 2019, pp. 954–964.

[24] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5132–5143.

[25] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE international conference on communications (ICC)*. IEEE, 2019, pp. 1–7.

[26] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer, 1999, pp. 223–238.

[27] M. Asad, A. Moustafa, and T. Ito, "Fedopt: Towards communication efficiency and privacy preservation in federated learning," *Applied Sciences*, vol. 10, no. 8, p. 2864, 2020.

[28] X. Zhu, J. Wang, W. Chen, and K. Sato, "Model compression and privacy preserving framework for federated learning," *Future Generation Computer Systems*, vol. 140, pp. 376–389, 2023.

[29] X. Li, X. Yang, Z. Zhou, and R. Lu, "Efficiently achieving privacy preservation and poisoning attack resistance in federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4358–4373, 2024.

[30] L. Chen, D. Xiao, Z. Yu, and M. Zhang, "Secure and efficient federated learning via novel multi-party computation and compressed sensing," *Information Sciences*, vol. 667, p. 120481, 2024.

[31] J. Xu, M. Yang, W. Ding, and S.-L. Huang, "Stabilizing and Improving Federated Learning with Non-IID Data and Client Dropout," *arXiv e-prints*, p. arXiv:2303.06314, Mar. 2023.

[32] J. Á. Morell and E. Alba, "Dynamic and adaptive fault-tolerant asynchronous federated learning using volunteer edge devices," *Future Generation Computer Systems*, vol. 133, pp. 53–67, 2022.

[33] J. Hao, Y. Zhao, and J. Zhang, "Time efficient federated learning with semi-asynchronous communication," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2020, pp. 156–163.

[34] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Automata, Languages and Programming: 29th*

*International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002 Proceedings 29*. Springer, 2002, pp. 693–703.
[35] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, 2022, pp. 365–390.
[36] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training," *arXiv e-prints*, p. arXiv:1712.01887, Dec. 2017.
[37] S. P. Karimireddy, Q. Rebjock, S. Stich, and M. Jaggi, "Error feedback fixes signsgd and other gradient compression schemes," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3252–3261.
[38] S. Zheng, Z. Huang, and J. Kwok, "Communication-efficient distributed blockwise momentum sgd with error-feedback," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[39] N. Ivkin, D. Rothchild, E. Ullah, I. Stoica, R. Arora *et al.*, "Communication-efficient distributed sgd with sketching," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[40] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro, "Robust stochastic approximation approach to stochastic programming," *SIAM Journal on optimization*, vol. 19, no. 4, pp. 1574–1609, 2009.
[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.
[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[43] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv e-prints*, p. arXiv:1708.07747, Aug. 2017.
[44] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.
[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[46] A. Archetti, E. Lomurno, F. Lattari, A. Martin, and M. Matteucci, "Heterogeneous Datasets for Federated Survival Analysis Simulation," *arXiv e-prints*, p. arXiv:2301.12166, Jan. 2023.
[47] A. Benaissa, B. Retiat, B. Cebere, and A. Eddine Belfedhal, "TenSEAL: A Library for Encrypted Tensor Operations Using Homomorphic Encryption," *arXiv e-prints*, p. arXiv:2104.03152, Apr. 2021.
[48] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov, "Opacus: User-Friendly Differential Privacy Library in PyTorch," *arXiv e-prints*, p. arXiv:2109.12298, Sep. 2021.
[49] R. L. Rivest, "All-or-nothing encryption and the package transform," in *International Workshop on Fast Software Encryption*. Springer, 1997, pp. 210–218.

**Wenxiu Ding** received her B.Eng. degree and PhD degree in information security from Xidian University, Xi'an, China in 2012 and 2017. She was a research assistant at the School of Information Systems, Singapore Management University from 2015 to 2016, and ever visited Aalto University in 2018. Now she is an associate professor in the School of Cyber Engineering at Xidian University. Her research interests include privacy preservation, access control and trust management.

**Yuxuan Xiao** received the B.Eng. degree in information security from Xidian University, Xi'an, China in June 2022. He is currently pursuing a Master's degree in the School of Cyber Engineering at Xidian University, with his main research focus being efficient and secure Federated learning.

**Zheng Yan** (Fellow, IEEE) received the D.Sc. degree in technology from the Helsinki University of Technology, Espoo, Finland, in 2007. She is currently a distinguished professor at the Xidian University, China. Her research interests are in trust, security, privacy, and data analytics. Dr. Yan is a Stanford World Top 2% Scientist, a ScholarGPS World Top 0.05% Highly Ranked Scholar, and an Elsevier Most Cited Chinese Researcher. She is currently a Co-EiC of Information Sciences and serves as an area/associate/guest editor for over 60 journals, e.g., ACM Computing Surveys, IEEE IoT Journal, Information Fusion, and IEEE Network Magazine, etc. She served as a General/Program Chair for over 50 international conferences. She is a Founder Steering Committee Co-Chair of IEEE Blockchain conference. She received 60 awards and honours, including Distinguished Leadership Award of IEEE TEMS TC on Blockchain and DLT, IEEE TCSC Award for Excellence in Scalable Computing, N2Women: Stars in Computer Networking and Communications, Distinguished Inventor Award of Nokia, IEEE Andrew P. Sage Best Transactions Paper, Best Journal Paper Award of IEEE ComSoc TCBD, etc. She is an External Member of Finnish Academy of Science and Letter, and a Fellow of IEEE, IET, AAIA, and AIIA.

**Ciwei Chen** is currently pursuing the B.Eng. degree in the School of Cyber Engineering at Xidian University, with his main research focus being federated learning.

**Yuxuan Cai** received the B.Eng. degree in computer science and technology from Chongqing Technology and Business University, Chongqing, China, in June 2021, and the M.Eng. degree in cyberspace security from Xidian University in 2024. He is currently pursuing the Ph.D. degree at Shanghai Jiao Tong University. His research interests include privacy-preserving federated learning, homomorphic encryption, SGX, and privacy protection for large language models.

**Xuyang Jing** received the PhD degree in cyberspace security from Xidian University, Xi'an, China, in 2020. He is currently n associate professor with the School of Cyber Engineering, Xidian University, Xi'an, China. His research interests include network measurements and sketches.