Privacy-Preserving Asynchronous Federated Learning for Heterogeneous IoT Devices

Hui Cheng, Qiao Xue, Youwen Zhu
College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China

Federated learning (FL), as a distributed machine learning paradigm, is particularly suitable for Internet of Things (IoT) scenarios where numerous edge devices collaboratively train a global model without sharing raw data, thereby reducing the risk of privacy leakage. However, IoT environments are typically characterized by device heterogeneity, unstable network connections, and limited computational and communication resources. These factors pose significant challenges to both training efficiency and privacy protection. For instance, resource-constrained IoT devices may slow down overall training progress, leading to inefficient resource utilization. Moreover, the gradient information uploaded by IoT devices can still be exploited by malicious attackers, resulting in potential privacy breaches. To address these issues, this thesis proposes a secure asynchronous federated learning algorithm tailored for IoT device-heterogeneous environments. The algorithm leverages model pruning to allocate sub-models with varying complexities according to the computational and communication capabilities of IoT clients, thereby improving resource utilization and accelerating training on low-end devices. Furthermore, it incorporates a staleness-aware asynchronous aggregation mechanism, dynamically adjusting aggregation weights to mitigate the negative effects of stale updates from delayed devices on the global model. To further enhance privacy protection, a differential privacy mechanism is integrated into the local training process by injecting carefully calibrated noise into gradient information, effectively preventing sensitive data leakage. Experimental results demonstrate that the proposed algorithm achieves superior comprehensive performance in model accuracy, convergence speed, and privacy protection strength, outperforming baseline asynchronous federated learning algorithms in device-heterogeneous environments.

Index Terms—**Asynchronous federated learning, Internet of Things, Differential privacy, Device heterogeneity, Privacy security.**

I. Introduction

In recent years, machine learning has become a key enabling technology across various domains due to its strong capabilities in data analysis and prediction. It plays a critical role in practical applications such as intelligent transportation, medical diagnostics, and the industrial Internet of Things (IIoT), and is evolving into a comprehensive discipline capable of addressing a wide range of complex problems. In IoT ecosystems, vast numbers of heterogeneous edge devices continuously generate massive volumes of data, creating opportunities for intelligent data-driven services such as smart manufacturing, autonomous driving, and predictive maintenance. This development provides efficient solutions to real-world demands [1]. As foundational infrastructures such as big data, cloud computing, and edge computing continue to mature, the barriers to model training and deployment have been significantly reduced, enhancing both the feasibility and efficiency of machine learning applications. However, despite this promising outlook, several challenges hinder the broader adoption of machine learning in IoT environments. In particular, data sharing remains a sensitive issue across many industries, and disparities in computational and communication resources across IoT devices further complicate large-scale collaborative deployment [2]. Two prominent challenges in this context are ensuring data privacy and accommodating the heterogeneity of IoT device capabilities.

Federated Learning (FL), introduced by Google in 2016 [3], is a distributed learning paradigm that allows IoT clients to collaboratively train a global model by uploading only local model updates, without exchanging raw data. This approach is particularly attractive in IoT and edge computing scenarios as it enhances communication efficiency and mitigates privacy risks. Nevertheless, conventional synchronous aggregation strategies face notable limitations in heterogeneous IoT settings, such as training delays caused by straggler devices, inefficient model updates, and unnecessary resource consumption [4]. To overcome these limitations, Asynchronous Federated Learning (AFL) has emerged as a promising alternative. In AFL, the server aggregates local updates immediately upon receipt, without waiting for all IoT devices to finish training. This greatly improves training efficiency and flexibility, especially in environments characterized by device heterogeneity and unstable communication links.

In the domain of data privacy, the growing value of data has heightened public concern over the risks of privacy leakage during data sharing processes. Newly enacted regulations, such as the General Data Protection Regulation (GDPR) in the European Union, have established stricter standards for data collection and usage, thereby increasing the difficulty of data acquisition [5]. Similarly, China officially implemented the Personal Information Protection Law on November 1, 2021, which further restricts the arbitrary collection and usage of personal data in commercial and academic contexts. Moreover, due to competitive concerns and privacy risks, organizations across industries are generally reluctant to share local IoT data. This cautious attitude is driven by multiple factors. In sensitive fields such as healthcare, finance, and IIoT, strict compliance requirements—exemplified by privacy regulations like GDPR—impose high barriers for data utilization. Enterprises are also concerned that disclosing core IoT data may undermine their competitive edge, which has led to the widespread

Manuscript received May 26, 2025; revised August 10, 2025. Corresponding author: Qiao Xue (email: qiaoxue@nuaa.edu.cn).

emergence of so-called "data silos." This phenomenon significantly impedes cross-institutional collaboration, as integrating data from multiple sources is not only costly but also operationally complex. Traditional machine learning, which relies on centralized data storage and processing, becomes increasingly incompatible with current demands for stringent privacy protection. In contrast, federated learning, which stores data locally on IoT devices and only transmits model updates to a central server, offers significant advantages in privacy-sensitive IoT scenarios. This decentralized nature enables FL to meet privacy requirements at a relatively low cost.

Device heterogeneity poses another fundamental challenge. With the rapid deployment of 5G networks and continuous advancements in IoT hardware, the number and diversity of edge devices have grown substantially. This has resulted in significant disparities in computational and communication capacities among IoT devices. For instance, there is often a large performance gap between high-end edge servers and resource-constrained sensors or embedded devices. Even among similar types of devices, communication delays and bandwidth availability can vary considerably due to differences in network conditions or geographic locations. In conventional FL frameworks, the server typically waits for all clients to complete local training before aggregating their updates. Slower IoT devices thus become bottlenecks, severely reducing training efficiency and overall resource utilization. Furthermore, some devices may fail to complete training or uploading due to intermittent connectivity or power limitations. The non-IID nature of local IoT data distributions across clients also adds to the training complexity, potentially introducing model bias and degrading both accuracy and generalization performance.

While asynchronous federated learning (AFL) shows significant promise in improving training efficiency and adapting to heterogeneous IoT environments, it introduces new security vulnerabilities. Specifically, gradient updates uploaded by clients may inadvertently leak information about their local IoT data. Malicious participants can potentially infer sensitive information about other clients by analyzing these updates. Recent advances in attack techniques—including membership inference, attribute inference, model inversion, and gradient leakage attacks—have posed serious threats to the privacy guarantees of FL systems [6]. Additionally, poisoning and backdoor attacks present further risks to the integrity of the global model. Current defense mechanisms against these privacy and security threats are often computationally intensive and thus difficult to implement effectively on resource-constrained and latency-sensitive IoT devices [7].

This work focuses on addressing the dual challenges of device heterogeneity and data privacy in IoT-based federated learning environments. We propose a novel heterogeneous federated learning algorithm that integrates dynamic model pruning, a time-aware asynchronous aggregation mechanism, and localized differential privacy protection, specifically designed to meet the constraints and requirements of IoT devices. The main contributions of this work are as follows:

1) **Resource-Aware Model Pruning for Heterogeneous Devices:** To address device heterogeneity in federated learning, we propose a dynamic model pruning method that adapts the global model to the computation and communication capabilities of each client. Specifically, sub-models are generated by adjusting the depth (i.e., number of hidden layers) according to individual resource constraints, enabling efficient and fair local training across diverse devices.
2) **Time-Aware Asynchronous Aggregation with Gradient Alignment:** To mitigate the negative impact of stale updates in asynchronous federated learning, we design a two-fold aggregation strategy. First, we introduce a gradient-alignment mechanism to ensure consistency among updates from heterogeneous sub-models. Second, we apply a time-decay weighting scheme to dynamically reduce the influence of delayed gradients, thereby improving the robustness and convergence rate of the global model.
3) **Localized Differential Privacy via Gaussian Perturbation:** We enhance privacy protection during model update transmission by injecting calibrated Gaussian noise into local gradients. This localized privacy-preserving mechanism ensures differential privacy without significantly compromising model utility. The privacy module is seamlessly integrated into our federated learning framework.
4) **Unified Framework and Comprehensive Evaluation:** We integrate the above modules into a unified asynchronous federated learning framework tailored to heterogeneous environments. Extensive experiments on benchmark datasets demonstrate the effectiveness of our approach in terms of model accuracy, convergence speed, and privacy protection under various levels of heterogeneity and noise.

Based on these challenges, this work proposes an asynchronous and privacy-preserving federated learning framework, PrivaAsyncFed, which will be detailed in the following sections. The rest of this paper is organized as follows. Section II introduces the preliminaries and problem definition. Section III explicates the SwapFed framework. Section IV presents experiment results. Section V reviews the related work. Section VI concludes the paper and outlines potential directions for future research.

II. Preliminaries and Problem Definition

In this section, we present the foundational challenges and corresponding techniques in asynchronous federated learning under heterogeneous conditions, focusing on device heterogeneity, model pruning, and privacy preservation.

A. Federated Learning

Federated Learning (FL) is a decentralized machine learning paradigm where training is performed locally on distributed devices, and only model updates are shared with a central server. Unlike conventional distributed learning, FL emphasizes data privacy, as raw data never leaves the local devices. FL is particularly suitable for privacy-sensitive scenarios and heterogeneous edge environments.

1) Synchronous Federated Learning

Synchronous Federated Learning (SyncFL) follows a round-based protocol where the central server coordinates global model updates after receiving training results from all participating clients. The process consists of the following steps:

- **Initialization:** The server initializes the global model W_G^0 and distributes it to K selected clients, along with training configurations such as learning rate, batch size, and local epochs.
- **Local Training:** Each client $k \in \{1, \ldots, K\}$ trains the received model W_G^t on its local dataset and obtains an updated model W_k^t.
- **Model Aggregation:** The server waits for all clients to upload their local models, and performs weighted aggregation based on the number of local samples n_k:

$$W_G^{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} W_k^t \tag{1}$$

where $n = \sum_{k=1}^{K} n_k$.

SyncFL provides stable convergence, especially under IID data distributions. However, it suffers from the "straggler problem": slow clients delay the entire training process, leading to idle resource consumption and low efficiency. This limitation becomes critical in heterogeneous environments where client capabilities vary significantly.

2) Asynchronous Federated Learning

Asynchronous Federated Learning (AsyncFL) eliminates the synchronization barrier by allowing clients to train and upload models independently. The server immediately updates the global model upon receiving any client update, without waiting for others. The training process is as follows:

- **Initialization:** Similar to SyncFL, the server initializes and broadcasts the global model W_G^0 to all clients.
- **Local Training and Update:** Clients perform local training independently and asynchronously upload their updated models. The uploaded model from client k at time step $t+k-1$ is denoted as W_k^{t+k-1}.
- **Model Aggregation:** The server performs pairwise aggregation using the latest global model and the received local update:

$$W_G^{t+k} = \frac{1}{2} \left(W_G^{t+k-1} + W_k^{t+k-1} \right) \tag{2}$$

AsyncFL enables more frequent global model updates, making the training process significantly faster and more adaptive. Since fast clients are no longer blocked by slow ones, system efficiency is improved and computing resources are better utilized. This property is particularly beneficial in heterogeneous systems.

Compared to SyncFL, AsyncFL exhibits faster convergence due to its high update frequency. After each local training round, a client can immediately contribute to the global model, allowing the system to quickly adapt to new data. This immediacy also enhances performance in non-IID and resource-diverse environments.

B. Differential Privacy

Differential Privacy (DP) is a rigorous mathematical framework for data privacy that ensures the inclusion or exclusion of any single individual's data has a minimal impact on the output of a computation. Originally introduced by Dwork et al. [8], DP provides formal guarantees against inference attacks from adversaries observing the output.

1) Formal Definitions

The following definitions provide the theoretical foundation for designing our asynchronous and privacy-preserving algorithm.

Definition 2.1 (ϵ-Differential Privacy [9]): A randomized algorithm $\mathcal{A} : \mathcal{D} \to \mathcal{R}$ satisfies ϵ-differential privacy if for any two adjacent datasets D and D', and for all subsets $S \subseteq \mathcal{R}$, the following holds:

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} \Pr[\mathcal{A}(D') \in S] \tag{3}$$

While ϵ-DP offers strong privacy guarantees, it can be overly strict in practical applications involving high-dimensional data or complex models. To address this, a relaxed variant is used:

Definition 2.2 ((ϵ, δ)-Differential Privacy [10]): A randomized algorithm $\mathcal{A}$ satisfies (ϵ, δ)-differential privacy if for any adjacent datasets D and D' and any subset $S \subseteq \mathcal{R}$,

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} \Pr[\mathcal{A}(D') \in S] + \delta \tag{4}$$

Here, ϵ controls the privacy budget: lower ϵ implies stronger privacy but reduced utility. δ represents the probability of privacy failure. When $\delta = 0$, (ϵ, δ)-DP reduces to pure ϵ-DP.

2) Global Sensitivity and Noise Mechanisms

Definition 2.3 (Global Sensitivity): For a function $f : \mathcal{D} \to \mathbb{R}^d$, its L_2-sensitivity is defined as:

$$\Delta f = \max_{D, D'} \|f(D) - f(D')\|_2 \tag{5}$$

where D and D' differ in at most one record. Sensitivity quantifies the maximum change in function output due to a single data point and directly determines the magnitude of noise required.

Two common mechanisms to achieve differential privacy are the Laplace and Gaussian mechanisms:

a) Laplace Mechanism.: Given function f, the Laplace mechanism adds noise drawn from a Laplace distribution:

$$\mathcal{A}(D) = f(D) + \mathrm{Lap}\left(\frac{\Delta f}{\epsilon}\right) \tag{6}$$

The Laplace distribution is defined by:

$$\mathrm{Lap}(x \mid \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|x|}{\lambda}\right) \tag{7}$$

It is commonly used to satisfy pure ϵ-DP and is simple to implement.

b) Gaussian Mechanism.: To achieve (ϵ, δ)-DP, the Gaussian mechanism adds noise drawn from a Gaussian distribution:

$$\mathcal{A}(D) = f(D) + \mathcal{N}(0, \sigma^2) \tag{8}$$

The standard deviation σ must satisfy:

$$\sigma \geq \frac{\Delta f \cdot \sqrt{2 \ln(1.25/\delta)}}{\epsilon} \tag{9}$$

Compared to Laplace, Gaussian noise enables a better trade-off between privacy and utility, especially in high-dimensional or iterative learning scenarios.

3) Deployment Architectures

DP can be implemented via two architectural paradigms: centralized and local differential privacy.

a) Centralized Differential Privacy (CDP).: In CDP, users transmit raw data or model parameters to a trusted central server, which performs DP noise addition. This centralized mechanism simplifies noise calibration and can provide stronger utility guarantees. However, it assumes full trust in the server—if compromised, user data privacy is at risk. Additionally, CDP can impose significant computational and communication overhead on the server.

b) Local Differential Privacy (LDP).: In contrast, LDP does not rely on a trusted aggregator. Each user perturbs their own data or model updates locally before transmission:

$$\text{Local Update} = f(D_k) + \text{Noise}$$

LDP offers stronger protection against server-side adversaries and is well-suited for edge devices with privacy concerns. However, as each client independently adds noise, the aggregated model may suffer from reduced accuracy.

C. Resource-Aware Model Pruning for Device Heterogeneity

In traditional federated learning, all participating clients typically use a uniform global model. However, this assumption overlooks the substantial variations in computational and communication capacities across devices in real-world settings. Enforcing a shared full-size model can lead to several challenges: faster clients may dominate the aggregation process in asynchronous federated learning, resulting in underutilization of slower clients' data; meanwhile, resource-constrained devices may fail to complete training tasks, potentially forcing the server to lower the global model complexity, thereby limiting performance.

To address these issues, we adopt a resource-aware model pruning strategy inspired by HeteroFL [11] and structured pruning techniques [12]. The server maintains a complete global model and dynamically generates sub-models of different sizes by reducing the number of channels in hidden layers based on each client's capability. For example, consider a convolutional layer with weight tensor $W_G \in \mathbb{R}^{d_g \times k_g}$. Given a scaling factor $r \in (0, 1]$, a client with pruning level p uses a sub-model parameter set $W_l^p \subseteq W_G$, where $d_g^p = r^{p-1} d_g$ and $k_g^p = r^{p-1} k_g$. The parameter count becomes $|W_l^p| = r^{2(p-1)} |W_G|$, representing a pruning ratio $R = r^{2(p-1)}$.

During aggregation, a layer-wise fusion scheme is applied. Parameters shared across all models are aggregated over all clients, while parameters specific to higher-complexity models are averaged only among clients capable of supporting them. Let the number of clients at each pruning level be $\{m_1, m_2, \ldots, m_p\}$. The aggregated global model W_G is constructed as follows:

$$W_l^p = \frac{1}{m} \sum_{i=1}^{m} W_i^p, \qquad \text{(shared)}$$

$$\Delta W_l^k = \frac{1}{n_k} \sum_{i=1}^{n_k} \left(W_i^k \setminus W_i^{k+1} \right), \quad \text{for } k = p-1, \ldots, 1$$

$$W_G = W_l^1 \cup \bigcup_{k=1}^{p-1} \Delta W_l^k,$$

This hierarchical aggregation strategy enhances the model's adaptability to heterogeneous environments while maintaining global convergence. By tailoring model complexity to local device constraints, it significantly improves resource efficiency and enables broader participation across devices.

D. Problem Definition

In asynchronous federated learning (AFL), clients perform local training independently and send their updates to the server without waiting for others. This decoupling between clients' training and server aggregation introduces a critical challenge: *model staleness*.

Let W_G^t denote the global model at training round t. A client k receives the global model at iteration τ and trains locally to produce an updated model W_k, which is then sent back to the server at round t for aggregation. The staleness of the local model is defined as the difference $s = t - \tau$, which measures how outdated the local model is compared to the current global state.

A larger staleness s implies that the local model was trained on an older version of the global model, potentially resulting in a conflicting update direction. This outdated update may degrade the global model's convergence or even cause divergence in training. Therefore, a key problem in AFL is how to adjust the aggregation of client models based on their staleness.

Problem Statement: Given a set of local models $\{W_k\}$ submitted at time t, each associated with a staleness value $s_k = t - \tau_k$, the goal is to design a staleness-aware aggregation mechanism that dynamically adjusts the aggregation weight of each model to minimize the adverse effects of stale updates on global model convergence and performance.

This paper addresses this challenge by introducing a staleness-controlled aggregation function, in which the aggregation weight is inversely related to the degree of staleness, and is adaptively scaled to balance stability and learning efficiency.

III. PrivaAsyncFed: A Secure Asynchronous Federated Learning Algorithm

In this section, we explicate the proposed solution PrivaAsyncFed designed for device-heterogeneous environments.

It addresses the issue of stale model updates caused by slow clients through staleness-aware weighted aggregation. Additionally, it enhances model efficiency and privacy via structured model pruning and local differential privacy.

A. *Model Pruning for Device Heterogeneity*

In asynchronous federated learning under device-heterogeneous settings, the server maintains a full global model W_G, while the size of the local model trained on each client is constrained by its computational capabilities. To adapt the model complexity to diverse client resources, we adopt a structured model pruning strategy based on a predefined channel scaling table, allowing hidden layers to be pruned accordingly.

The hidden layers of the global model W_G consist of several convolutional layers. We perform structured pruning by reducing the number of input and output channels in each layer. Formally, the weight tensor of a convolutional layer is denoted as $W_G \in \mathbb{R}^{d_g \times k_g}$, where d_g and k_g represent the number of input and output channels, respectively. Given a channel scaling table $S = \{1, 0.75, 0.5, 0.25\}$ and a pruning level $p \in \{1, 2, 3, 4\}$, we define the pruned number of channels for each hidden layer as:

$$d_g^p = \lfloor S[p] \cdot d_g \rfloor, \quad k_g^p = \lfloor S[p] \cdot k_g \rfloor \tag{10}$$

where $\lfloor \cdot \rfloor$ denotes the floor operation to ensure integer channel counts. The total number of parameters in the pruned model is reduced to:

$$|W_l^p| = S[p]^2 \cdot |W_G| \tag{11}$$

Clients either report their hardware capabilities during initialization or the server estimates their training speed from initial tasks. The pruning level p is determined based on clients' measured computation time or declared hardware specifications. Based on this information, the server assigns an appropriate pruning level p to each client. When dispatching training tasks, the server generates a customized sub-model by applying the pruning ratio corresponding to level p. We define the pruning operation as:

$$W_k = \texttt{pruning}(W_G, S[p])$$

This mechanism ensures that each client trains a sub-model that matches its available resources, improving the overall efficiency and scalability of the system.

This figure 1 illustrates the core framework of PrivaAsyncFed. At the top, the global model generates locally adapted models of varying sizes (Client 1-3) through structured pruning. Clients train their personalized models on local datasets and return updates to the server, which aggregates them asynchronously based on timestamps (Clock 1-3). Arrows depict the bidirectional flow of model distribution and aggregation.

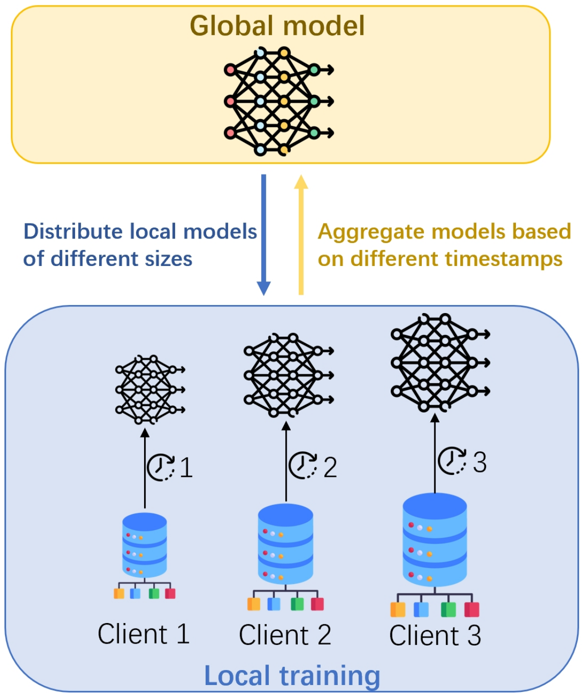

Global model
Distribute local models of different sizes
Aggregate models based on different timestamps
1
2
3
Client 1
Client 2
Client 3
Local training

Fig. 1: Overview of the PrivaAsyncFed framework showing the relationship between the global model, pruned sub-models, and heterogeneous clients.

B. *Asynchronous Aggregation for Heterogeneous Models*

In heterogeneous model settings, clients may train sub-models whose dimensions do not match the global model due to their limited computational capabilities. Specifically, slower devices are often assigned sub-models with reduced hidden layers, resulting in fewer parameters. Consequently, some parameters in the global model are missing in the client model, leading to dimensional inconsistency during aggregation.

To address this issue, we propose a selective aggregation strategy that aggregates only the shared parameters between the global and client models. Let W_G^t denote the global model at round t, and W_k^t denote the client k's local model. Let P be the set of parameters shared between W_G^t and W_k^t. For each parameter $p \in P$, its values in the global and local models are denoted as $W_{G,p}^t$ and $W_{k,p}^t$, respectively. To balance the contribution of both models, we introduce a local aggregation weight α, and update the global parameter as:

$$W_{G,p}^{t+1} = (1-\alpha) W_{G,p}^t + \alpha W_{k,p}^t \tag{12}$$

This formulation reflects the relative importance of the global and local models in the update process. A larger α increases the influence of the local model in the aggregation.

For parameters $q \notin P$—i.e., those absent in the client model—the global model retains their current values without modification:

$$W_{G,q}^{t+1} = W_{G,q}^t \tag{13}$$

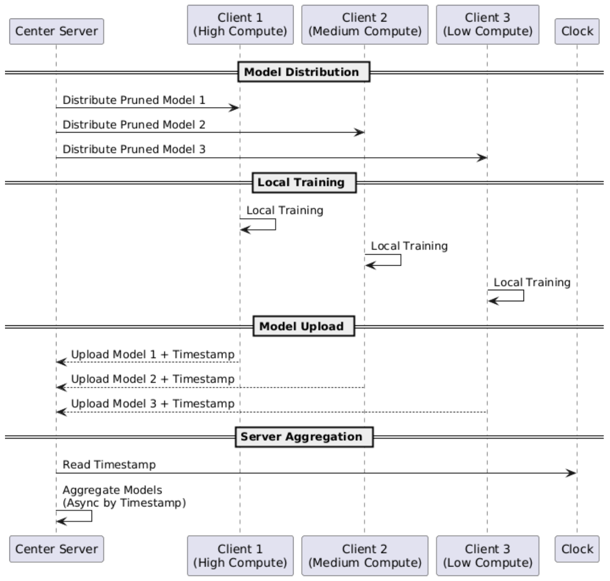

Center Server
Client 1 (High Compute)
Client 2 (Medium Compute)
Client 3 (Low Compute)
Clock
Model Distribution
Distribute Pruned Model 1
Distribute Pruned Model 2
Distribute Pruned Model 3
Local Training
Local Training
Local Training
Local Training
Model Upload
Upload Model 1 + Timestamp
Upload Model 2 + Timestamp
Upload Model 3 + Timestamp
Server Aggregation
Read Timestamp
Aggregate Models (Async by Timestamp)

Fig. 2: Temporal diagram illustrating the asynchronous update process with staleness-aware weighting in PrivaAsyncFed.

Combining both cases, the update rule for the global model becomes:

$$W_G^{t+1} = \begin{cases} (1-\alpha)W_{G,p}^t + \alpha W_{k,p}^t, & \text{if } p \in P \\ W_{G,q}^t, & \text{if } q \notin P \end{cases} \tag{14}$$

This strategy enables effective integration of heterogeneous models while preserving the unique parameters of the global model.

Furthermore, in our asynchronous setting, the server updates the global model immediately upon receiving an update from any client, without waiting for all clients. After aggregation, a new training task is dispatched to the responding client.

To mitigate the negative impact of stale updates, we introduce a staleness-aware weighting mechanism. When a client receives the global model, it also receives a timestamp τ indicating the current global round. Upon returning the local update, the client also sends back τ. The staleness is computed as $t - \tau$, where t is the current server round. The local aggregation weight α is then computed as:

$$\alpha = \beta \cdot S(t - \tau) \tag{15}$$

where $S(t-\tau)$ is a staleness decay function satisfying $S(0) = 1$ and decreasing with increasing staleness.

We define $S(t - \tau)$ as:

$$S_{a,b}(t-\tau) = \begin{cases} 1, & t - \tau \le b \\ \frac{1}{a(t-\tau-b)}, & \text{otherwise} \end{cases} \tag{16}$$

Here, b is a threshold beyond which the staleness starts penalizing the aggregation weight, and a controls the decay rate. When staleness is small ($t - \tau \le b$), local updates are deemed fresh and merged with constant importance $\beta \in (0, 1)$. As staleness grows beyond b, the contribution of outdated models decays progressively, thereby reducing their adverse impact on the global model.

Figure 2 depicts the temporal workflow of PrivaAsyncFed's asynchronous aggregation process. The central server distributes pruned local models to three clients with varying computational capabilities. After local training, clients upload their updated models along with timestamps τ_k at different speeds. Upon receiving an update from client k, the server first computes the staleness as $t-\tau_k$, where t represents the current global round. The aggregation weight $\alpha_k = \beta \cdot S_{a,b}(t - \tau_k)$ is then calculated according to the staleness decay function defined in Equation (16). As demonstrated in the figure, high-compute clients (e.g., Client 1) complete training faster, resulting in minimal staleness ($t-\tau \approx 0$) and maximal contribution to the global update ($\alpha_k = \beta$). In contrast, low-compute clients (e.g., Client 3) produce updates with significant staleness, whose influence is progressively reduced through the decay function $S_{a,b}(\cdot)$. This staleness-aware mechanism effectively balances model freshness and training efficiency in heterogeneous environments.

C. Client-Side Differential Privacy Protection

To address privacy concerns in federated learning, especially under the local differential privacy assumption where the server is considered untrusted, we adopt a local implementation of the differential privacy (DP) mechanism. The following outlines the DP procedure executed by each client.

Assume the global model received from the server is W_G, the client's local dataset is D, the learning rate is μ, and the gradient clipping threshold is C. The loss function is denoted as $\mathcal{L}(D, W_G)$. The differentially private training process on the client includes the following steps:

a) Step 1: Compute and Clip Per-Example Gradients.: The client computes the per-sample gradient g_i for each $D_i \in D$, and applies L2 clipping as:

$$g_i = \frac{g_i}{\max(1, \frac{\|g_i\|_2}{C})} \tag{17}$$

b) Step 2: Average the Gradients.: The average gradient over the dataset is calculated as:

$$g = \frac{1}{|D|}\sum_{i=1}^{|D|} g_i = \frac{1}{|D|}\sum_{i=1}^{|D|} \nabla\mathcal{L}(D_i, W_G) \tag{18}$$

c) Step 3: Apply Gradient Descent.: The client performs local model update:

$$W_k = W_G - \mu g \tag{19}$$

d) Step 4: Add Noise for Privacy.: To ensure differential privacy, noise is added to the model parameters:

$$W_k' = W_k + \text{noise}(\Delta f/\epsilon) \tag{20}$$

Here, noise$(\Delta f/\epsilon)$ denotes a noise function dependent on the sensitivity Δf and the privacy budget ϵ.

e) Gradient Clipping Justification.: Gradient clipping ensures the L2 norm of each g_i does not exceed C, which is critical for bounding the noise scale. Without clipping, gradients may be unbounded, resulting in excessive noise that harms model convergence. Clipping not only helps meet DP guarantees but also improves training stability.

Algorithm 1 Secure Asynchronous Federated Learning under Device Heterogeneity (PrivaAsyncFed)

```
Require: Total global rounds $T$, number of clients $K$, local epochs $H$, local dataset $D_k$, learning rate $\mu$, clipping bound $C$, privacy budget $\epsilon$, decay coefficient $\beta$, pruning ratio schedule $S[\cdot]$
Ensure: Final global model $\omega_G^T$
     Server Initialization:
 1:  Initialize global model $\omega_G^0$
 2:  Broadcast initial model and assign ID to each client $k \in \{1, \ldots, K\}$
 3:  Assign client-specific pruning ratio $S[k]$ to client $k$
     Server Execution (for each round $t = 1$ to $T$):
 4:  Wait to receive update $(\Delta_k, \tau_k)$ from any client $k$
 5:  Compute aggregation weight: $a_k = \beta \cdot S[t - \tau_k]$
 6:  Model Aggregation:
 7:  for each parameter group $p$ in $\omega_G$ do
 8:      $\omega_{G,p}^t \leftarrow (1 - a_k) \cdot \omega_{G,p}^{t-1} + a_k \cdot (\omega_{k,p}^{t-1} + \Delta_k)$
 9:  end for
10:  Model Pruning:
11:  $\omega_G^p \leftarrow \mathtt{prune}(\omega_G^t, S[k])$ {Prune based on client $k$'s ratio}
12:  Send $(\omega_G^p, t)$ to client $k$
     Client $k$ Execution (upon receiving $(\omega_G^p, t)$):
13:  Set local model: $\omega_k^0 \leftarrow \omega_G^p$
14:  Set local time stamp: $\tau \leftarrow t$
15:  Compute sensitivity: $\Delta f \leftarrow \mu \cdot \frac{2C}{|D_k|}$
16:  for $h = 0$ to $H - 1$ do
17:      Sample mini-batch $\mathcal{B}_k^h \sim D_k$
18:      Compute gradient: $g_k^h \leftarrow \nabla f(\omega_k^h; \mathcal{B}_k^h)$
19:      Gradient Clipping:
20:      $\tilde{g}_k^h \leftarrow g_k^h / \max\left(1, \frac{\|g_k^h\|_2}{C}\right)$
21:      Model Update:
22:      $\omega_k^{h+1} \leftarrow \omega_k^h - \mu \cdot \tilde{g}_k^h$
23:      Add Gaussian Noise:
24:      $\omega_k^{h+1} \leftarrow \omega_k^{h+1} + \gamma_k^h$, where $\gamma_k^h \sim \mathcal{N}(0, (\Delta f/\epsilon)^2)$
25:  end for
26:  Compute model update: $\Delta_k \leftarrow \omega_k^H - \omega_G^p$
27:  Upload $(\Delta_k, \tau)$ to server
```

TABLE I: Experimental Environment Configuration

Environment	Configuration
Operating System	Windows 10 22H2 64-bit
Processor	AMD Ryzen 7 5800H @ 3.20GHz
Memory	16 GB
GPU Memory	6 GB
Storage	512 GB
Programming Language	Python 3.12.3

f) Sensitivity Analysis.: The sensitivity Δf is defined as the maximum L2 distance between outputs on two adjacent datasets D and D', differing in at most one data point D_k. Let W_k and W_k' be the updated parameters based on D and D', respectively. Then:

$$\Delta f = \max_{D,D'} \|W_k - W_k'\|_2 = \max_{D,D'} \|\mu(g - g')\|_2 = \mu \cdot \max_{D,D'} \|g - g'\|_2 \tag{21}$$

Given that only one sample differs between D and D', the difference in gradients simplifies to:

$$\|g - g'\|_2 = \frac{1}{|D|} \|g_k - g_k'\|_2 \tag{22}$$

Using the L2 norm bound from clipping:

$$\|g - g'\|_2 \le \frac{\|g_k\|_2 + \|g_k'\|_2}{|D|} \le \frac{2C}{|D|} \tag{23}$$

Thus, the sensitivity is:

$$\Delta f = \mu \cdot \frac{2C}{|D|} \tag{24}$$

g) Noise Calibration via Gaussian Mechanism.: To achieve (ϵ, δ)-differential privacy, the noise is sampled from a Gaussian distribution:

$$\text{noise}(\Delta f/\epsilon) \sim \mathcal{N}\left(0, \sigma^2\right), \quad \text{where } \sigma \ge \frac{\Delta f \cdot \sqrt{2 \ln(1.25/\delta)}}{\epsilon} \tag{25}$$

The privacy budget ϵ controls the noise level: a smaller ϵ implies stronger privacy but higher noise, potentially degrading model utility. In our experiments, we analyze the trade-off between privacy and performance by varying ϵ.

h) Algorithm Overview.: The detailed pseudocode of the proposed algorithm is presented in Algorithm 1, which consists of two main components: the client-side procedure and the server-side procedure.

In this algorithm, $w_{k,p}$ denotes the parameters shared between the global model and the client's personalized model. Communication between the clients and the server is asynchronous.

During initialization, each client is assigned the same model state (w_0, t_0), even before any updates are received. After initialization, the server proceeds asynchronously and accepts T updates from clients in the order they arrive.

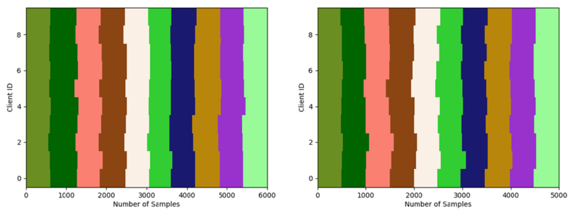


Fig. 3: IID Data Partitioning for MNIST (left) and CIFAR-10 (right)

IV. Experiments

In this section, we conduct extensive experiments to evaluate the proposed method. Firstly, we describe the experimental settings, including the employed datasets, competing methods, hardware environments, and experimental parameters. Subsequently, we present detailed experimental results that confirm the advantages of our method compared to the baseline approaches.

A. Experimental Setup

To validate the effectiveness of the proposed PrivaAsyncFed algorithm in device-heterogeneous federated learning scenarios, we conduct a comprehensive set of experiments. This section details the datasets, experimental environment, simulation settings, evaluation metrics, and baseline configurations.

1) Datasets

We evaluate our method on two widely used classification datasets with different data characteristics and complexity:

- **MNIST** [13]: A benchmark dataset for handwritten digit recognition, consisting of 60,000 training and 10,000 testing grayscale images of size 28×28. Each image represents a digit from 0 to 9, totaling 10 classes.
- **CIFAR-10** [14]: A standard dataset for natural image classification, comprising 50,000 training and 10,000 testing RGB images of size 32×32. The dataset contains 10 categories such as airplane, automobile, cat, and dog.

2) Experimental Environment

All experiments are conducted on a single physical machine simulating a federated learning environment. Table I presents the system specifications.

3) Simulation of Device Heterogeneity

To simulate device heterogeneity, we emulate a network of 10 heterogeneous clients. Each client is assigned a different response latency to reflect diverse computational capabilities. Specifically, we define four device types with varying training speeds: *fast*, *medium-fast*, *medium-slow*, and *slow*. The client distribution is as follows: three *fast*, three *medium-fast*, two *medium-slow*, and two *slow* clients.

Correspondingly, four model variants with reduced hidden layer sizes are used to match device capabilities. The reduction ratios are $\{1.0, 0.75, 0.5, 0.25\}$ for *fast*, *medium-fast*, *medium-slow*, and *slow* clients, respectively.

4) IID Data Partitioning

To isolate the influence of device heterogeneity, data heterogeneity is not considered in this study. Each dataset is partitioned in an IID manner by evenly distributing class labels among all clients. This setup ensures comparable experimental conditions and helps focus on evaluating the impact of asynchronous training and privacy noise. Figure 3 shows the IID distribution of data across 10 clients for MNIST (left) and CIFAR-10 (right), with different colors representing distinct class labels. We adopt an IID partition to isolate the effects of device heterogeneity; evaluation under non-IID data will be explored in future work.

5) Evaluation Metric

Model performance is measured using classification accuracy, defined as the proportion of correctly predicted samples out of the total predictions.

6) Model Architecture and Training Settings

We employ a modified version of LeNet [13] adapted for three-channel RGB input. Additionally, the architecture supports pruning of hidden layers to accommodate device heterogeneity.

The learning rate for both datasets is set to 0.01. For MNIST, the local training step is 1, while for CIFAR-10, it is set to 5 due to convergence issues with smaller steps. The server performs 400 asynchronous aggregations. The gradient clipping threshold is fixed at 2. Table II summarizes the hyperparameters used.

7) Comparison and Privacy Budget Settings

We adopt a controlled variable approach for comparative analysis. The differential privacy budget ϵ is treated as the primary variable, ranging from 1 to 10. Additionally, a non-private baseline is included. All other experimental parameters remain fixed across runs, as detailed in Table II.

B. Experimental Results

1) Training under Different Privacy Budgets

Figures 4a and 4b illustrate the training accuracy of the PrivaAsyncFed algorithm on MNIST and CIFAR-10 datasets under different privacy budgets. The x-axis represents the number of global aggregation rounds, and the y-axis indicates the test accuracy. We consider privacy budgets $\epsilon = 1, 5, 10$ and a non-private case (denoted as $\epsilon = \infty$).

The results show that our algorithm achieves stable convergence under all privacy settings. However, a smaller ϵ (i.e., stronger privacy) introduces more noise, leading to reduced accuracy. For example, on MNIST, when $\epsilon = 1$, the model converges more slowly and plateaus at around 92.5%, whereas for $\epsilon = 5$, the final accuracy exceeds 95%. As ϵ increases, the performance gradually approaches the non-private case.

2) Average Accuracy Analysis

Figures 4c and 4d show the average accuracy of the final global model under different privacy budgets. The x-axis represents the privacy budget ϵ, and the y-axis denotes the average accuracy after convergence.

As ϵ increases from 1 to 10, the average model accuracy improves accordingly. This demonstrates that stronger privacy (smaller ϵ) leads to greater noise and thus lower accuracy. When $\epsilon \geq 7$, the accuracy gap narrows, suggesting that even small noise may still slightly affect the global model's aggregation process.

3) Comparison with Baseline Method

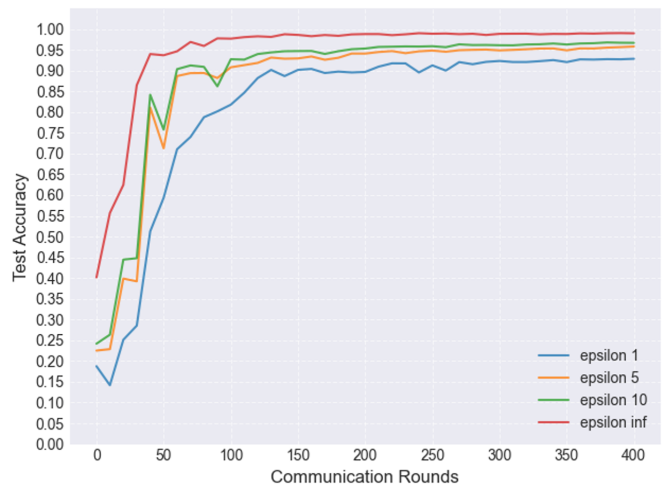
We select DP-FedAsync as the main baseline because it represents a classical asynchronous FL algorithm enhanced with differential privacy, which aligns closely with our proposed framework. We compare PrivaAsyncFed with a baseline method, **DP-FedAsync**, which is a differentially private variant of FedAsync [15]. In DP-FedAsync, Gaussian noise is injected during local training to protect privacy. For fair comparison, both algorithms use identical configurations and datasets.

Figures 5a–5d depict the test accuracy comparisons between PrivaAsyncFed and DP-FedAsync under $\epsilon = 1$ and $\epsilon = 5$ on both datasets.The performance gain of PrivaAsyncFed primarily stems from its staleness-aware aggregation and resource-adaptive pruning, which jointly improve convergence stability and model efficiency.

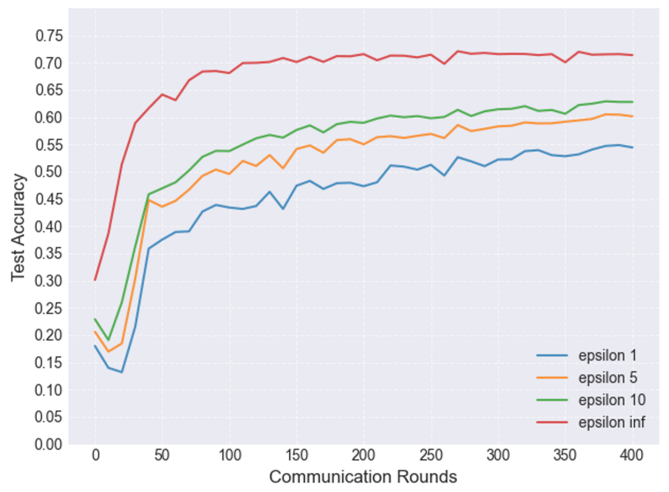
Across all settings, PrivaAsyncFed consistently outperforms DP-FedAsync in terms of model accuracy. The results validate the effectiveness of our algorithm in achieving a better balance between privacy protection and model utility. Compared to the baseline, our method demonstrates superior accuracy, faster

TABLE II: Hyperparameters and Model Configuration

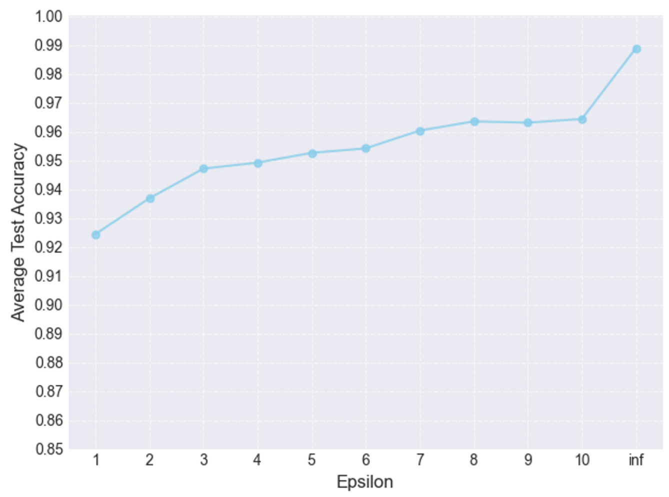
Dataset	Model	Hidden Sizes	Local Steps	Batch Size	Optimizer	LR	Agg. Rounds
MNIST	LeNet	[64,128,256,512]	1	64	SGD	0.01	400
CIFAR-10	LeNet	[64,128,256,512]	5	64	SGD	0.01	400



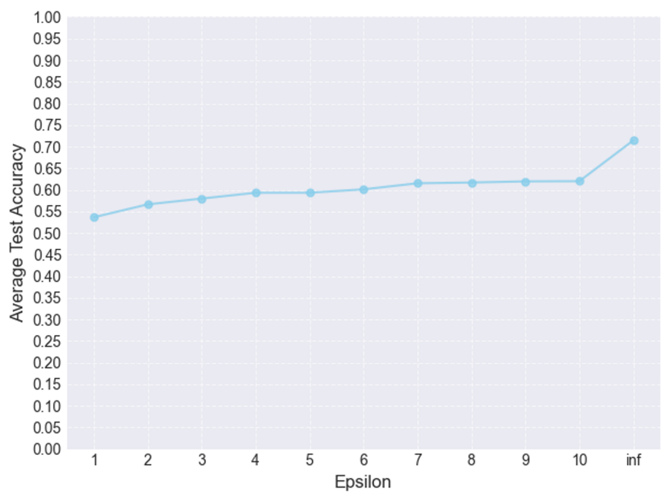
(a) Training performance on MNIST under different ϵ values



(b) Training performance on CIFAR-10 under different ϵ values



(c) Average accuracy on MNIST with different ϵ values



(d) Average accuracy on CIFAR-10 with different ϵ values

Fig. 4: Experimental setup and basic performance evaluation under varying privacy budgets (ϵ).

convergence, enhanced privacy protection, and higher training efficiency.

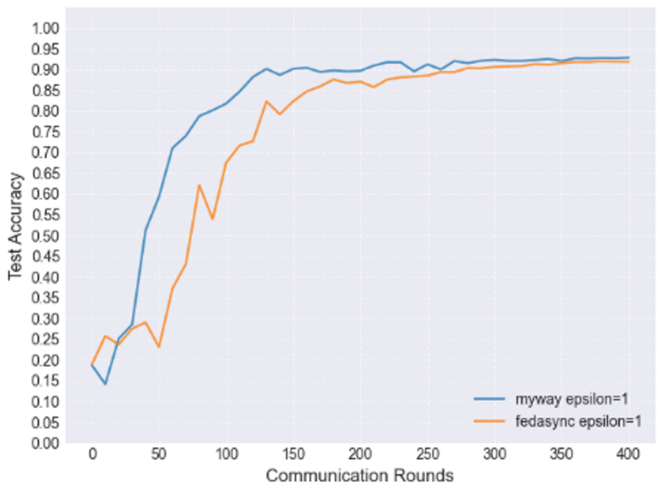
Moreover, due to model pruning and asynchronous communication, PrivaAsyncFed potentially reduces overall communication volume and energy cost, which will be quantitatively studied in future work.

V. Related work

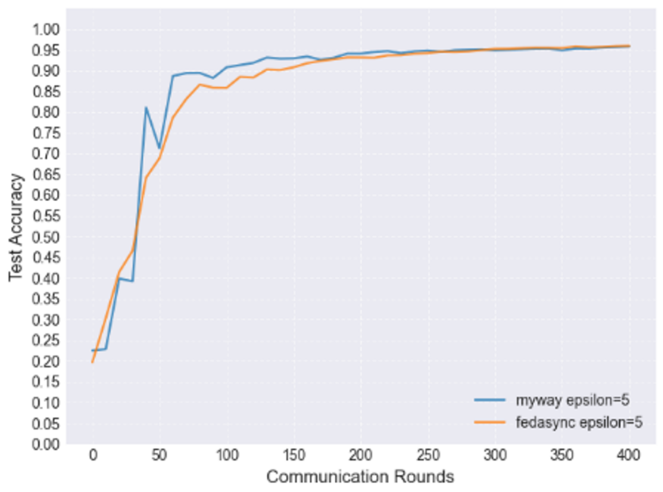
In this section, we systematically review the existing literature on

A. Federated Learning

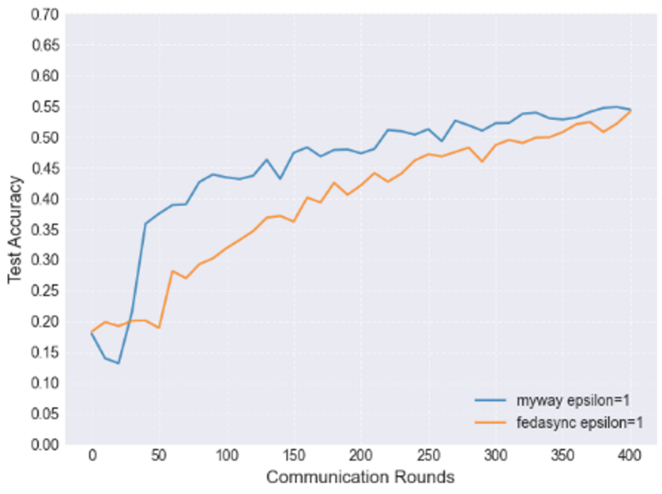
Within the federated learning (FL) framework, FedAvg [3] is a classical and widely adopted algorithm known for its effectiveness. It relaxes the constraint on the number of local training steps at each client by allowing multiple local updates before model synchronization. After performing several rounds of local training, clients upload their updated models to the central server, which aggregates them through a weighted averaging strategy to update the global model. By enabling multiple local iterations, FedAvg significantly reduces com-



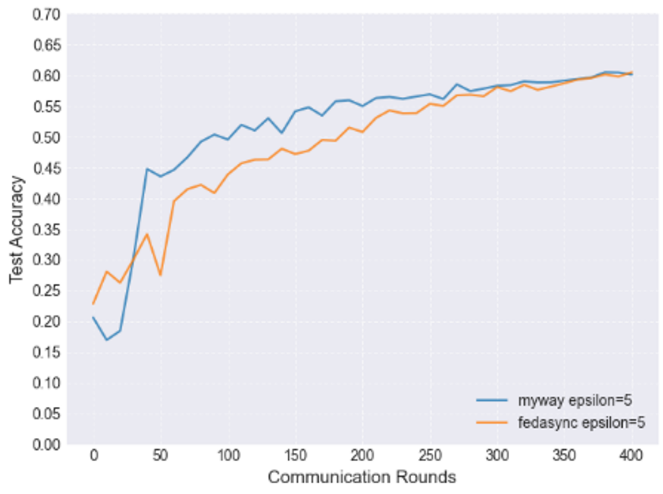
(a) Comparison on MNIST ($\epsilon = 1$)



(b) Comparison on MNIST ($\epsilon = 5$)



(c) Comparison on CIFAR-10 ($\epsilon = 1$)



(d) Comparison on CIFAR-10 ($\epsilon = 5$)

Fig. 5: Performance comparison under different datasets and privacy budgets.

munication overhead and improves overall training efficiency.

FedProx [12] extends FedAvg by addressing challenges arising from non-independent and identically distributed (non-IID) data and system heterogeneity. Its core idea is to introduce a proximal term into the local objective function, encouraging local updates to stay closer to the global model. This regularization enhances model consistency across clients, particularly beneficial in asynchronous federated learning settings. The proximal term in FedProx helps mitigate local model drift caused by asynchronous updates, thereby improving robustness and convergence speed in heterogeneous environments, offering more reliable support for real-world FL deployments.

While synchronous FL performs well under homogeneous settings with uniform computational and communication capabilities, it suffers from performance bottlenecks in heterogeneous environments, where slower clients can delay the overall training process. To address this, HeteroFL [11] introduces a novel approach that supports heterogeneous model architectures, breaking the conventional assumption that all clients must train models with the same structure. In HeteroFL, the server maintains a full global model, while each client selects a sub-model according to its resource constraints. These sub-models typically consist of the early layers of the global model or pruned versions with fewer channels. After local training, clients upload only the corresponding parts of their sub-models, and the server aggregates and updates the relevant substructures layer by layer. This design not only reduces the computational and communication burden on resource-constrained clients but also ensures effective utilization of their local data. Simultaneously, high-capacity clients can train more parameters, thus continuously improving the global model's performance. HeteroFL successfully balances efficiency and accuracy in heterogeneous environments.

In another example, Choi et al. [16] address the dual challenges of high communication costs and performance degradation caused by non-IID data in FL. They propose a communication-efficient FL framework based on knowledge distillation and data augmentation. Their method introduces a model-splitting mechanism, where deep neural networks are divided into a shared public component and a private local component. Specifically, all clients share a shallow public model with a unified structure, which extracts generalizable features and participates in federated aggregation via knowledge distillation. Meanwhile, each client retains a private model component that is trained and updated exclusively on its local data to capture personalized patterns.

Asynchronous federated learning (AFL) is also effective in addressing the issue of slow devices bottlenecking overall training progress. By eliminating the idle waiting time after fast clients complete their local updates, AFL improves both device utilization and training efficiency. In AFL, the server adopts an asynchronous update mechanism, allowing global model updates to proceed immediately upon receiving updates from a subset of clients, rather than waiting for all participants to finish. This mechanism significantly accelerates the training process. However, it also introduces the challenge of inconsistent model updates due to varying staleness of client contributions.

To address this issue, Xie et al. [15] proposed incorporating a staleness-aware weighting strategy during aggregation. By assigning time-decay factors to outdated gradients, their approach reduces the adverse impact of stale updates on the global model, effectively mitigating model drift and enhancing the stability and reliability of AFL in practice. Similarly, Wu et al. [17] proposed an AFL scheme that classifies client updates into three categories: fresh, tolerable, and stale. Fresh updates are aggregated immediately, tolerable ones are deferred to the next round, while stale updates are discarded entirely. Nguyen et al. [18] introduced FedBuff, an AFL algorithm that incorporates a buffering mechanism on the server side. The server accumulates model updates within a predefined time window and then performs a collective aggregation.

Current research on AFL primarily focuses on alleviating the negative effects of straggler clients caused by asynchronous operations. Key research directions include client selection strategies, weighted aggregation schemes, gradient compression techniques, semi-asynchronous learning paradigms, and model partitioning approaches.

From both domestic and international perspectives, federated learning has received widespread attention and has become a highly active research area. Many scholars are devoted to improving the performance of FL algorithms under various conditions, such as adapting to complex data distributions and heterogeneous device environments. In terms of privacy protection, researchers continue to explore more effective encryption and privacy-preserving techniques to ensure data security during model training. Meanwhile, the application of federated learning in practical domains such as healthcare, finance, and the Internet of Things (IoT) is also advancing steadily. These efforts aim to bridge the gap between theoretical advances and real-world deployment, enabling FL to effectively address data collaboration and privacy challenges across diverse industries.

1) Privacy Protection in Federated Learning

Although federated learning (FL) enhances privacy by keeping data local, evolving attack techniques have demonstrated that sharing only model parameters still poses risks of information leakage. For instance, Ziegler et al. [19] applied deep leakage from gradients (DLG) to attack a DenseNet121 model trained under an FL setting. Using only gradient updates from local clients—without any prior knowledge of the original data—they successfully reconstructed high-fidelity chest X-ray images closely resembling the original samples. In asynchronous FL, clients may receive different versions of the global model due to variations in update timing. When a client consecutively updates its local model based on sequential global models, an adversary may infer sensitive data by analyzing changes across neighboring global model versions. These cases illustrate that, even when training data is retained locally, the uploaded model updates can still reveal private information.

The FL framework typically involves two parties: a central server and multiple clients. Adversaries may compromise either the server or a subset of clients to launch privacy attacks. In the first scenario, the server is assumed to be untrusted and may exploit collected model updates to infer private client data. In the second scenario, client-to-client trust is absent—an adversary may control one or more malicious clients, accessing their data, local models, and global updates to infer or poison information. Four main categories of attacks have been identified in FL: model poisoning, data poisoning, membership inference, and reconstruction inference. In this work, we focus on preventing inference of raw data from client-uploaded model parameters.

Privacy-preserving mechanisms in FL are generally categorized into three types [20]: centralized, distributed, and local privacy protection. In centralized privacy protection, data providers trust the data collector and do not interact with each other. Original data is uploaded to a central entity, which then performs privacy-preserving operations before sharing the data or results. Distributed privacy protection assumes no trust between data providers or the collector. Protection is enforced by a trusted third party that manages interactions and secures private data. In local privacy protection, each data provider independently applies privacy-preserving techniques before sharing data. Yang et al. [21] proposed a differentially private FL algorithm that allows clients to choose personalized privacy levels and add noise to their gradients before uploading them. FL systems must be designed to protect against privacy leakage from local updates—even if parameters are intercepted during transmission or the server itself is compromised [22]. This paper focuses on enhancing privacy in FL by applying local differential privacy (LDP) techniques at the client side.

In federated learning, differential privacy is typically achieved by injecting noise into model updates before they are uploaded [23]. Rather than manipulating the raw data directly, clients first constrain the magnitude of their computed gradients, and then add calibrated random noise to the clipped gradients before transmission. For example, Wu et al. [17] proposed a differentially private FL algorithm that adopts a segmented clipping strategy based on individual privacy requirements. This mechanism helps obscure each client's specific contribution to the global model update, making it difficult to infer original data characteristics from the gradients. By doing so, FL systems can prevent raw data leakage and also defend against gradient-based inference attacks, thus significantly improving the robustness of privacy protection in federated learning.

While prior studies have addressed asynchronous updates or differential privacy individually, few works have combined both in the context of heterogeneous IoT environments. PrivaAsyncFed bridges this gap by integrating local differential privacy with staleness-aware asynchronous aggregation.

VI. Conclusion

In this work, we proposed PrivaAsyncFed, a secure and privacy-preserving asynchronous federated learning algorithm tailored for heterogeneous IoT environments. Our approach addresses two key challenges prevalent in IoT-based FL: (i) device heterogeneity, arising from the diverse computational power and communication capabilities of IoT devices; and (ii) privacy risks due to potential exploitation of gradient information.

To tackle these challenges, PrivaAsyncFed adopts a staleness-aware weighted aggregation strategy, which mitigates the negative impact of delayed or outdated updates from low-capacity devices. At the same time, differential privacy mechanisms are incorporated into the local training phase to protect sensitive IoT data without significantly sacrificing model performance. Furthermore, the algorithm employs model pruning to allocate sub-models dynamically, improving resource utilization and reducing straggler effects.

We conducted extensive experiments on benchmark datasets that emulate heterogeneous IoT environments under varying levels of system heterogeneity and privacy budgets. The results demonstrate that PrivaAsyncFed achieves faster convergence, higher accuracy, and stronger privacy guarantees compared to existing asynchronous FL baselines. These findings confirm the potential of PrivaAsyncFed as a practical and robust solution for federated learning in real-world IoT scenarios, effectively balancing efficiency, robustness, and privacy protection. In future work, we plan to extend PrivaAsyncFed to non-IID data distributions and evaluate its communication efficiency in real-world IoT deployments.

References

[1] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, 2015.

[2] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, "A survey on federated learning," *Knowledge-Based Systems*, vol. 216, p. 106775, 2021.

[3] 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.

[4] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *Computer Science Review*, vol. 50, p. 100595, 2023.

[5] F. D. Protection, "General data protection regulation (gdpr)," *Intersoft Consulting, Accessed in October*, vol. 24, no. 1, 2018.

[6] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 739–753.

[7] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[8] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[9] C. Dwork, "Differential privacy," in *International colloquium on automata, languages, and programming*. Springer, 2006, pp. 1–12.

[10] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2006, pp. 486–503.

[11] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," *arXiv preprint arXiv:2010.01264*, 2020.

[12] 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.

[13] 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, 2002.

[14] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images.(2009)," 2009.

[15] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[16] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data," *arXiv preprint arXiv:1811.11479*, 2018.

[17] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "Safa: A semi-asynchronous protocol for fast federated learning with low overhead," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 655–668, 2020.

[18] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International conference on artificial intelligence and statistics*. PMLR, 2022, pp. 3581–3607.

[19] J. Ziegler, B. Pfitzner, H. Schulz, A. Saalbach, and B. Arnrich, "Defending against reconstruction attacks through differentially private federated learning for classification of heterogeneous chest x-ray data," *Sensors*, vol. 22, no. 14, p. 5195, 2022.

[20] A. El Ouadrhiri and A. Abdelhadi, "Differential privacy for deep and federated learning: A survey," *IEEE access*, vol. 10, pp. 22 359–22 380, 2022.

[21] G. Yang, S. Wang, and H. Wang, "Federated learning with personalized local differential privacy," in *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*. IEEE, 2021, pp. 484–489.

[22] X. Lu, Y. Liao, P. Lio, and P. Hui, "Privacy-preserving asynchronous federated learning mechanism for edge network computing," *Ieee Access*, vol. 8, pp. 48 970–48 981, 2020.

[23] B. Li, H. Gao, and X. Shi, "Feddp: Secure federated learning for disease prediction with imbalanced genetic data," *bioRxiv*, pp. 2023–01, 2023.