A Related Work

A.1 FL-Based Hierarchical Architecture

With the advancement of research, hierarchical FL architectures have emerged as popular solutions for addressing large-scale distributed optimization and communication cost issues. Hierarchical architectures are mostly supported by collaborative training algorithms [Liu *et al.*, 2020a]. At the edge, performance improvement and cost minimization can be achieved by optimizing resource allocation for individual edge servers or by task association across multiple edge servers [Luo *et al.*, 2020]. Additionally, the edge and cloud can employ different strategies to handle uncertain states, thereby enhancing interaction performance [Wu *et al.*, 2020]. Based on the characteristics of FL and edge computing, the game-based method [Lu *et al.*, 2024] dynamically adjusts the relationship between the client and the edge server to achieve better resource allocation and communication optimization. Also, the architecture has great advantages in the intelligent control of the system [Wang *et al.*, 2019]. Current research on hierarchical architectures primarily focuses on inter-layer collaboration and task allocation mechanisms. Unlike these works, we emphasize adopting differentiated aggregation and update strategies based on the different requirements of models stored in the cloud and at the edge. This processing and coordination mechanism effectively reduces the costs associated with redundant training and model adaptation processes, achieving global and local relative improvements while improving data utilization for the system and users.

A.2 Personalized FL in Heterogeneous Env.

Given that the goal of software and systems is to provide better services to users, it is crucial to adjust models for personalization and localization to improve system adaptability and model performance. This is called Personalized Federated Learning (PFL). Personalized Federated Learning (PFL) is crucial to provide better customized models for users. From the perspective of model relations, `Per-FedAvg` [Fallah *et al.*, 2020] based on Model Agnostic Meta-Learning (MAML) achieves personalized improvement through local adaptation. Advanced personalized convergence is also achieved by using the `pFedMe` [T Dinh *et al.*, 2020] with regularization in the loss, by parameterized knowledge transfer [Zhang *et al.*, 2021], and by sharing representations across clients [Collins *et al.*, 2021]. The personalization process can focus not only on relationships between models but also on specific parts or components of models. Model partitioning enables partial personalization [Pillutla *et al.*, 2022], and the introduction of hypernetworks allows for the identification of contributing factors at the layer granularity [Ma *et al.*, 2022]. `FedPFT` [Peng *et al.*, 2024] adapts to downstream tasks by allocating foundation models and fine-tuning them through layer-by-layer compression and precise alignment. Additionally, introducing non-federated Batch-Normalization (BN) layers improves individual user personalization [Mills *et al.*, 2021]. These approaches achieve personalized adjustments to local models by processing different components at varying levels of granularity. Moreover, in heterogeneous environments, knowledge transfer based on semantic similarity [Yi *et al.*, 2024] is also an efficient method to enhance adaptability.

A.3 Balance of Generalization and Personalization

A high-performance global model provides greater adaptability and broader applicability for the system, while high-performance local personalized models benefit users in their specific contexts. Therefore, balancing the global and local is crucial in federated systems. Initially, model balancing was guided by fairness, ensuring that the global model performs well across diverse client data distributions without favoring any single client [Mohri *et al.*, 2019]. Gradually, the degree of optimization between local and global models can be controlled by parameter mixing [Hanzely and Richtárik, 2020], and optimal model mixing techniques that rely on mixed parameters have been shown to influence the generalization bounds of local and global models [Deng *et al.*, 2020]. Recent studies have found that this balancing process can be modeled using game theory as a competitive and cooperative interaction between clients and the server [Chen *et al.*, 2024]. Our objective is to leverage the advantages of hierarchical architectures to achieve relatively optimal performance for both the global model stored in the cloud and the local personalized models on edge clients.

B Sequence Diagram of Architecture Workflow

The process begins with the edge devices initiating communication with the fog layer by sending identification information [*register()*]. In response, the fog layer provides the edge devices with initial cluster configurations [*sendEdgeClusterConfig()*]. This initial clustering is done randomly, and the *edge* devices are assigned to the cluster controlled by the fog nodes. Concurrently, the cloud initializes the global model [*initGlobalModel()*] and distributes the initial model to the fog layer, laying the foundation for subsequent local training of client models on the edge devices and aggregation within the fog and cloud layers. The processes of initial clustering for edge devices and the model initialization in the cloud server are performed in parallel. Upon receiving the initial global model from the cloud server [*sendinitGlobalModel()*], the fog layers distribute the global model to all of the edge devices for which they are responsible [*sendGlobalModel()*]. The objective here is to train the global model on local data available at the edge device. This training process occurs in a loop, iteratively refining the global model and the local models until both reach a certain level of convergence and performance.

Once the distribution from the fog layer to the edge devices is complete, the edge devices begin local training [*beginLocalTraining()*], updating the model based on local data. After a round of training is completed, the private information is separated from the trained model [*separateBNStatistics()*] to create a personalized and non-personalized model. The private information is represented by separable parameters stored in BN Statistics. Both models are then concurrently sent to the fog layer [*sendBNStatistics()* and *sendNPModel()*] for further aggregation. The aggregation at the fog layers [*ag-*

gregateLocalModels()] enhances the models' generalization across the data from various edge devices while maintaining privacy, and then updates the BN statistics [*updateBNStatistics()*]. Then, according to Definition 1 & Definition 2, based on the designed mechanism and the feedback from the cloud and the edge in the previous round, the joint optimization strategy of this round is adjusted [*updateJStrategy (via R Condition)*].

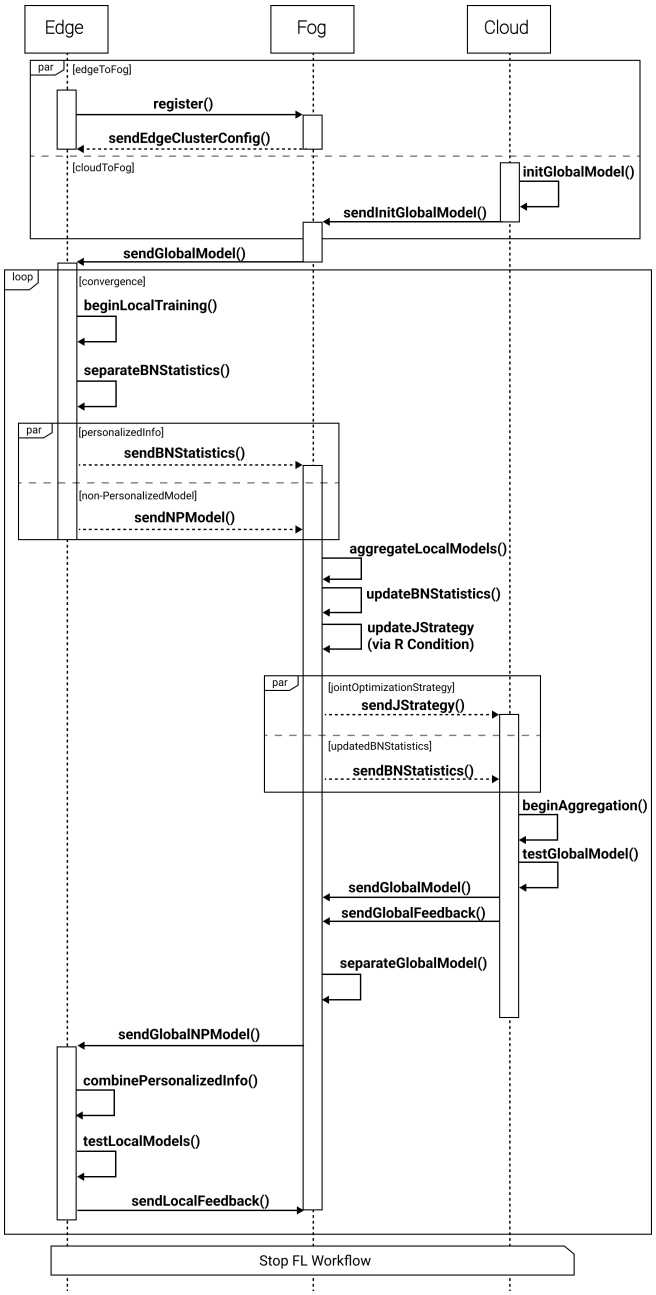


Figure 5: Sequence diagram of the workflow across edge, fog, and cloud of `JobFed`

The updates to statistics and strategy are transmitted to the cloud server [*sendJStrategy()* and *sendBNStatistics()*], where the cloud server performs global aggregation of the updated information according to the strategy [*beginAggregation()*]. This step integrates the insights of all fog layers to form a refined global generalization model that captures the collective knowledge of the entire system. After aggregation is completed, the cloud server will store the global model and update and overwrite it after each training. After the global update is completed, the global model is tested using the global data distribution [*testGlobalModel()*], and the test feedback is sent back for the next round of strategy updates [*sendGlobalFeedback()*].

The cloud then sends the global aggregated model back to the fog layer [*sendGlobalModel()*], which, after separation [*separateGlobalModel()*], distributes the global model without personalized information to the edge devices represented by all clients [*sendGlobalNPModel()*]. This model contains the public information of all clients in the system on their respective devices and has been globally adapted to make it more suitable for subsequent local personalized adjustments. After receiving this model, the client combines it with local private information and performs the personalization process [*combinePersonalizedInfo()*]. During the personalization process, the BN layer adjusts the local model of each client to adapt the global model to local data distributions. After local adaptation is completed, they are tested using a local distribution dataset [*testLocalModels()*], and the test feedback is sent back to the fog for the next round of strategy updates [*sendLocalFeedback()*]. At the same time, the local personalized models will participate in subsequent training as updated local models.

C `JobFed`'s Technical Supplements

C.1 Dynamic Adjustment Rule for the Mixing Parameter

Remark 2. *We consider the method of dynamic mixing parameters [Deng* et al.*, 2020]. However, the difference is that the original work focused on adjusting the mixing parameters, minimizing the generalization bound, and then finding the best balance between the global model and the local model. Instead of adjusting model dependencies, we focus on the progress of the personalization and then use a Relative Optimal Control method to transform the process into a joint optimization problem.*

Remark 3 (Relationship Between R and $J_{\text{joint}}(\alpha_i)$)**.** *According to the definitions, α controls the mixing ratio. When α_i is large, the system relies more on the global generalization model; when α_i is small, the system relies more on the local personalized model. β regulates the weight in the relative optimization function. When β is large, the loss ratio plays a major role in the optimization; when β is small, the loss change ratio dominates the optimization direction.*

R and $J_{\text{joint}}(\alpha_i)$ are two independent optimization objectives, but their optimization processes are complementary. Based on this balance, we optimize $J_{\text{joint}}(\alpha_i)$ by adjusting α_i, further refining the contribution ratio between the global and personalized models in the system. Therefore, the coordination of α and β ensures that the system neither over-relies on

the global model nor ignores the personalized model during the optimization process.

C.2 Stopping Criterion

Definition 3 (Stopping Condition under Normal Convergence)**.** *To ensure that the training process stops at the appropriate time, we introduce a stopping criterion based on the stability of the joint optimization function $J_{joint}(\alpha_i)$ and the rate of change in the losses of the global and personalized models. The stopping criterion is designed to ensure that the training process halts when the system has likely reached an optimal balance between the global generalization and local personalized models. The criterion involves three conditions: If*

$$\left|\frac{dJ_{joint}(\alpha_i)}{d\alpha_i}\right| \leq \zeta \text{ and } |\Delta\mathcal{L}_{global}| \leq \epsilon_1 \text{ and } |\Delta\mathcal{L}_{local}| \leq \epsilon_2 \tag{10}$$

then stop training.

$\frac{dJ_{\text{joint}}(\alpha_i)}{d\alpha_i}$ represents the derivative of the joint optimization function with respect to the mixing parameter α_i. It indicates how sensitive the joint optimization function is to changes in α_i. When this derivative is small, it suggests that further adjustments to α_i will have little effect on improving the balance between the global generalization and local personalized models. That is, the balancing condition of both models is reached. ζ is a threshold parameter that determines when the derivative $\frac{dJ_{\text{joint}}(\alpha_i)}{d\alpha_i}$ is small enough to consider stopping the training. A smaller value of ζ implies that we require a very stable joint optimization function before stopping, while a larger ζ allows for stopping the training even if there is still some potential for improvement.

Then, a smaller value of $\Delta\mathcal{L}_{\text{global}}$ indicates that the global model's loss has stabilized, suggesting that further training may not yield significant improvements, so as $\Delta\mathcal{L}_{\text{local}}$. ϵ_1 and ϵ_2 are threshold parameters that determine when the change rates of the losses, global and local, are small enough to consider stopping training. A smaller value of ϵ implies that we require the losses to be very stable before stopping. This is expected to be a human-controlled parameter.

D Experimental Preparation

D.1 Model Setting and Data Processing

We use 3 datasets and 3 structures for this experiment, they are:

MNIST [LeCun *et al.*, 1998], 60K training samples, 10K test samples, 28×28 grayscale. Classification Structure: 2×Fully-Connected (FC) layers of 200 neurons with 1×BN layer in the middle, softmax output.

CIFAR-10 [Krizhevsky *et al.*, 2009], 50K training samples, 10K test samples, 32×32 RGB. DNN Model: 2×Convolutional (Conv) layers with 32 and 64 filters, respectively, each followed by 1×BN layer, ReLU, and 1×MaxPooling. The flattened output is passed through 1×FC layer with 512 neurons, followed by ReLU and 1×FC layer for softmax output.

CIFAR-100 [Krizhevsky *et al.*, 2009], 50K training samples, 10K test samples, 32×32 RGB. We use fine_labels for fine-grained testing. Resnet-18 Structure: 1×Conv layer with 7×7 filters, 2×Pooling layers, 8×Residual units, and 1×FC layer. Specifically, BN layers are used after Conv operations, without replacement by Group Normalization. Finally softmax output.

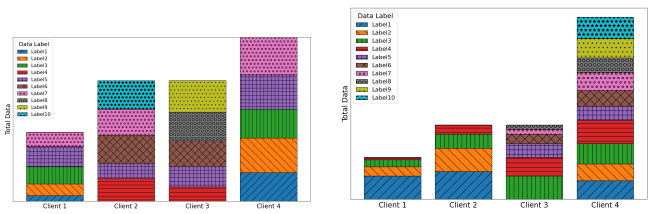
In large-scale environments with a high number of participants, the sample size allocated to each client tends to be relatively small. From the perspective of data heterogeneity, this increases the extremity of the environment, i.e. the difficulty of the task. For the fine-grained CIFAR-100 dataset, we perform multiple data allocations to ensure that each client has the basic amount of data required to start training.

D.2 Experimental Parameter Settings

In addition to the parameters mentioned in the main text, the client batch size (B) is set to 20. The total rounds (T) are set to 300 (standard) or a threshold value (variable) depending on different experiments. The client epoch (E) is set to 1. The training device ($device$) is defaulted to GPU. The random seed is set to 0 by default. The client learning rate ($client\_lr$) and server learning rate ($server\_lr$) vary depending on the baselines and experimental environments. In `pFedMe`, the server updates the global model with the designed parameter (β) set to 1.0, and in `Per-FedAvg`, β varies between 0.1/0.2 depending on the experiments. For `SCAFFOLD` and `FedProx`, the weight decay ($weight\_decay$) is set to 1e-4. The proximal regularization parameter ($FedProx\_mu$) in `FedProx` is set to 1e-4. The α update learning rate (η) used in our approach is set to 0.1 by default, with the predefined balancing ratio (γ) defaulting to 0.5, and the balancing ratio in the condition ($optBeta$) set to 0.5 by default. For experiments where the results do not show significant differences as in the ablation study on $optBeta$, two decimal places are retained; for all other experiments, one decimal place is retained.

D.3 Data Partitioning and Visualization

For the dataset partitioning, consistent with previous work, we employ a Dirichlet distribution to adjust the proportions of samples allocated to different clients, thereby creating a non-IID data distribution. To better align with real-world data distributions, we introduce a data quantity imbalance process, deliberately ensuring that different clients vary in both their class distributions and sample quantities. We denote these as Data Scenario 1 and Data Scenario 2, respectively. The Figure 6 presented are visual representations and do not reflect the actual data distribution.



(a) Data Scenario 1 (b) Data Scenario 2

Figure 6: Data Distribution Visualization Examples

Specifically, Data Scenario 1 is based on the classical non-IID distribution, characterized by class difference. Subsequently, we utilize sharding to assign varying data quantities to each client, which more closely resembles real-world conditions compared to previous approaches. In our setup, $min\_shards\_per\_client = 1$ and $max\_shards\_per\_client = 2$, meaning each client can receive one or two data shards. Assuming the system comprises 4 clients and 10 data classes, as illustrated in Figure 6a, each client is allocated data from 5 classes, encompassing all 10 categories within the task. Clients 2 and 3 possess the same total quantities of data but differ in their class distributions, here for example, sharing 3 common classes. Clients 1 and 4 share the same class distributions but differ in data quantities. Additionally, Clients 1 and 2 have differing class distributions and data quantities.

Data Scenario 2 builds upon Data Scenario 1 but is designed to be more extreme, presenting a more challenging data distribution scenario. Specifically, we manually assign the same class distributions to 50% of the clients. Furthermore, through random sampling, certain clients are assigned a greater variety of classes, resulting in mixed class distributions across clients. Based on this foundation and following the same sharding method, we provide clients with data shards of varying sizes, thereby achieving an extreme data distribution scenario. As depicted in Figure 6b, Clients 1 and 2 share the same class distributions but differ in quantities. Clients 2 and 3 have different class distributions but the same total data quantities, with Client 3 possessing a greater variety of classes. Client 4 includes all class categories and has the largest data quantity. However, the random allocation process may sometimes result in Client 4 including all class categories but having the smallest quantity.

E Experimental Results and Analysis

E.1 Technical Supplement to Stability of Global Generalization Model

We process the data and figures here to make them more clearly show the results we wish to illustrate. By using colored bands to cover the amplitude of the convergence process, we visually demonstrate the impact of the parameter-mixing-based `JobFed` method on the global generalization model. Narrower band regions indicate higher convergence stability, while wider regions reflect greater fluctuations in the global convergence process. For ease of comparison, we darken the shadow effect of the `JobFed` by 30%.

E.2 Analysis Supplement to Stability of Global Generalization Model

In this way, it is easier to understand the consideration of stability of **Definition 2** in Sec.4.3. When achieving a certain level of convergence performance, we aim for global convergence to be achieved steadily, ensuring that fluctuations from individual client models do not overly influence the global. Therefore, our approach focuses on loss fluctuations to prevent excessive updates from either the global or local. Whether it is the global generalization model or local personalized models, overly rapid adaptation to data distributions is risky to the other party, as it signifies that the model(s) are excessively adjusting to their respective data distributions. Sudden changes in the global generalization model, regardless of whether performance is enhanced or diminished, can impact the adaptation and balance of other local participants to their respective data scenarios, and vice versa.

E.3 Analysis Supplement to Comprehensive Comparison of Different Sampling Rates, Data Scenarios, and Noise Conditions

Among the 64 results, 61 are the best results, and the other 3 that do not achieve the best also obtain the second-best results. This shows that in a complex environment with more participants and noise, `JobFed`'s balance-based optimization method needs to take a certain amount of time (the preparation phase) to stabilize the strategy and coordinate the models in the hierarchical architecture when the models are in the early unstable balancing stage. This is reflected in the slower convergence speed of one of the global/local models compared to baselines with specific optimization objectives. However, the final result is promising, as the comprehensive results outperform single-objective (generalization or personalization) optimization methods.

Except for `JobFed`, in this large-scale complex environment with different sampling rates and noise conditions, the vanilla method `fedAvg` shows strong adaptability and accounts for most of the second-best results, which also illustrates the comprehensiveness and scalability of the initial FL method design.