Client-Driven Federated Learning for Dynamic Mixtures of Distributions

Chenqing Zhu[1][0009−0008−5364−4447] and Songze Li[1,2][0000−0003−4282−3307]⋆

[1] Southeast University, Nanjing, China
{chenqingzhu, songzeli}@seu.edu.cn
[2] Engineering Research Center of Blockchain Application, Supervision and Management (Southeast University), Ministry of Education, Nanjing, China

Abstract. Conventional federated learning (FL) is largely server-driven, with the server controlling model evolution and client participation. This design is often mismatched with practical edge environments, where clients may require timely personalized model updates for their local tasks. We present Client-Driven Federated Learning (`CDFL`), an asynchronous personalized FL framework centered on client-side adaptation demands. In `CDFL`, a client initiates interaction when it needs a model refresh: it uploads its locally trained model to the server and receives a customized model in return. To support heterogeneous and time-varying data distributions, the server maintains a repository of cluster models and iteratively refines them using uploaded client models. Unlike conventional clustered FL methods that send multiple cluster models to clients for local distribution estimation, `CDFL` shifts this estimation process to the server and returns only a *single* model to each client, thereby reducing client-side computation and communication overhead. We provide a convergence analysis of `CDFL`. Extensive experiments on diverse datasets and settings show that `CDFL` consistently improves model quality and computational efficiency over representative baselines.

Keywords: Federated Learning · Client-Driven Learning · Asynchronous Personalization · Cluster Modeling · Edge Intelligence

1 Introduction

Federated Learning (FL) (18) enables multiple clients to collaboratively train models without sharing their raw data. In practical edge environments, however, FL must handle not only privacy requirements but also statistical heterogeneity, intermittent client availability, and limited client-side resources. These challenges have motivated extensive studies on personalized FL and clustered FL, which aim to provide better adaptation to heterogeneous clients (5; 17; 21).

Despite these advances, the coordination logic of most existing FL frameworks remains largely server-driven: the server decides when training proceeds, which clients participate, and how model evolution is organized. Such a design

⋆ Corresponding author.

may be mismatched with practical edge scenarios in which clients experience changing local tasks or drifting data distributions and may therefore need timely model refresh on demand. Although asynchronous FL allows clients to upload updates at different times, its primary focus is still on robust server-side optimization under stale or delayed updates, typically toward a shared model or a server-centric training objective (15; 13). This leaves a gap between asynchronous aggregation and asynchronous *personalized* model delivery.

In this paper, we address this gap through **C**lient-**D**riven Federated Learning (`CDFL`), a framework centered on client-side adaptation demands. We consider the setting where each client draws data from a time-varying mixture of latent cluster distributions and may request a model refresh when its mixture changes. In `CDFL`, the server acts as a lightweight service provider: upon receiving a locally trained model from a client, it returns a customized model adapted to the client's current distribution. Figure 1 illustrates the workflow.

To support this process, `CDFL` adopts a clustered FL design where the server maintains cluster models and performs cluster-related distribution estimation. Many clustered FL methods require clients to evaluate multiple cluster models locally, increasing client-side communication and computation costs (5; 17; 21). Other methods infer grouping from uploaded models, but often assume stronger synchronization (4; 15). In contrast, `CDFL` uses the uploaded local model to infer the client's current distributional preference at the server, updates the cluster repository accordingly, and returns only a *single* personalized model to the client. Our main contributions are summarized as follows:

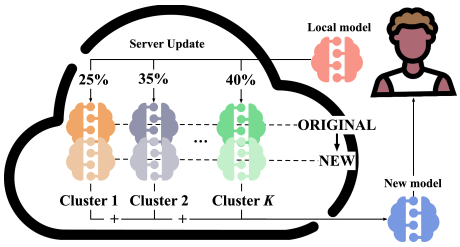


Fig. 1: High-level overview of `CDFL`. A client initiates interaction when it needs a model refresh. The server performs cluster-related inference using the uploaded model, updates own cluster models, and returns a refreshed customized model.

- We propose `CDFL`, a client-driven federated learning framework for asynchronous personalized adaptation, where clients request model refresh on demand instead of waiting for server-organized rounds.
- We design a server-side cluster inference mechanism with a maintained repository of cluster models, which avoids sending multiple candidate models to clients and enables single-model personalized delivery.
- We provide theoretical convergence analysis and extensive empirical evaluation of `CDFL` under diverse datasets and system settings, showing improved performance and reduced client-side overhead.

2 Related Work

Clustered and Personalized FL. To address statistical heterogeneity, prior work studies clustered and personalized FL, including hard and soft clustering methods (5; 17; 21). Some approaches estimate client distributions through

expectation-maximization or maintain multiple cluster-aware models, while others incorporate personalized regularization (16; 6; 22). However, existing clustered FL methods often require client-side estimation over multiple candidate models or infer grouping from uploaded models under stronger coordination assumptions (5; 17; 21; 4). In contrast, our framework shifts cluster-related estimation to the server and returns only a single personalized model to each client.

Asynchronous FL. Asynchronous FL allows the server to process client updates without waiting for synchronized rounds, and mainly focuses on handling delayed or stale updates in server-side optimization (27; 1; 20). Some methods further use tiering, scheduling, or staleness-aware grouping mechanisms (24; 30). More recent studies begin to combine asynchrony with richer personalization structures: CASA considers asynchronous clustered FL (15), while EchoPFL studies asynchronous personalized FL with on-demand broadcast on mobile devices (13). Different from these methods, our goal is client-driven asynchronous personalized model refresh with server-side cluster inference.

Online and Client-Centric Personalization. Another line of work studies online adaptation, communication-efficient personalization, or user-centric objectives in FL (2; 8). Related personalized methods also include combining local and global models (1; 3) and application-specific systems such as HAR-oriented personalization (29). In comparison, ours focuses on time-varying cluster mixtures, client-initiated interaction, and server-side personalized delivery.

3 Problem Definition

Consider an FL system with one central server and many distributed clients. The server maintains K cluster models, corresponding to K distributions $P_1, \ldots, P_K$, and has a proxy dataset D_k for each P_k. Note that the existence of small-size proxy datasets is rather a mild and common assumption in FL literature (11; 14). The value of K is determined a priori according to the type of training objective or is deducted from a small amount of data collected in advance. Given a loss function $l(w; x, y)$, each cluster $k \in [K] \triangleq \{1, \ldots, K\}$ aims to find an optimal model w_k that minimizes the objective

$$F_k(w_k) = \mathbb{E}_{(x,y)\sim P_k}[l(w_k; x, y)]. \tag{1}$$

The training takes T global epochs. For each epoch $t \in [T]$, some client m collects local data following a mixture of distribution $P_m^t = \sum_{k=1}^K \alpha_{mk}^t P_k$, with $\alpha_{mk}^t \in [0, 1]$ and $\sum_{k=1}^K \alpha_{mk}^t = 1$. Here α_{mk}^t is the importance weight of cluster k to client m at epoch t. The importance weights may vary over time, i.e., $\alpha_{mk}^t \neq \alpha_{mk}^{t'}$ for $t \neq t'$, and are unknown to the client. Each time when client m's data distribution shifts, it may choose to fit the local model v_m^t to the new distribution, and uploads it to the server. The server enhances v_m^t to construct a personalized model u_m^t for client m:

$$u_m^t = g(v_m^t, \{w_k^{t-1}\}_{k=1}^K, \{D_k\}_{k=1}^K), \tag{2}$$

following some rule g, and returns u_m^t to client m. In the meantime, server also updates the cluster models using some function $\mathbf{h}$, such that

$$(w_1^t, \ldots, w_K^t) = \mathbf{h}(v_m^t, \{w_k^{t-1}\}_{k=1}^K, \{D_k\}_{k=1}^K). \tag{3}$$

Specifically, the local model v_m^t is obtained by optimizing the local objective

$$J_m^t(v_m^t; u_m^\tau) = \frac{1}{n_m^t} \mathbb{E}_{(x^i,y^i)\sim P_m^t} \Big[\sum_{i=1}^{n_m^t} l(v_m^t; x^i, y^i) \Big] + \frac{\rho}{2} \left\| v_m^t - u_m^\tau \right\|^2 . \tag{4}$$

Here n_m^t is the number of data samples at client m in epoch t; ρ is some scaling parameter; $\tau < t$ is the last epoch when client m uploads its model v_m^τ to server.

4 Client-driven Federated Learning

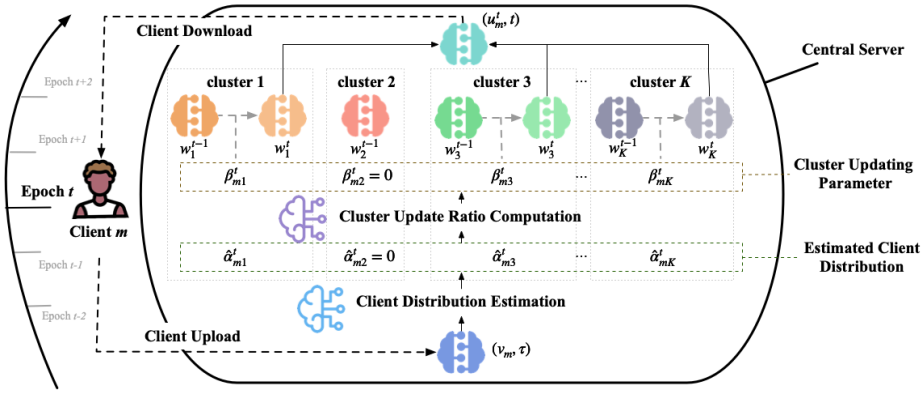


Fig. 2: `CDFL` workflow. Client m uploads model and epoch index of last model update (v_m, τ) to the server. Server performs distribution estimation, derives cluster updating parameters to update the cluster models, and finally constructs an aggregated model u_m^t to send back to the client. Note here as the client's distribution is estimated not to contain P_2, cluster 2's model is not updated and not used in computing u_m^t.

We describe the operations of the client and the server respectively in the proposed `CDFL` framework. The entire workflow of `CDFL` is depicted in Fig. 2 and detailed in Algorithm 1.

4.1 Client Update

Initialization. `CDFL` is designed to be fully open and dynamic, which does not presuppose a fixed total number of clients. A client can seamlessly joint the system, simply via retrieving an initialization tuple from the server, comprising an initial model and the index of current epoch, denoted as (u_m^t, t).

Training and Uploading. Whenever a client m feels necessary to perform a model update, perhaps due to the occurrence of distribution shift and the resulting performance degradation, it performs local training on current dataset to obtain a local model v_m, and uploads (v_m, τ) to the server, where τ is the index of the latest epoch in which client m updates its model with the server. Having received v_m, the server generates an enhanced model u_m^t for current epoch t, and returns (u_m^t, t) to client m. Finally, client m updates $\tau = t$, and starts to use u_m^t for local tasks.

Algorithm 1 `CDFL`

Input: Server pre-trained model w_k^0, server proxy dataset $D_k \sim P_k$ ($k \in [K]$), staleness threshold $\tau_0 < T$, cluster update threshold $\beta_0 \in (0,1)$

Output: Local model parameter v_m, cluster model parameters $w_1, \ldots, w_K$

Initialization: Server sends $\left(u^0, 0\right)$ to each client, $u^0 = \frac{1}{K}\sum_{k=1}^K w_k^0$. Global epoch $t \leftarrow 0$. Run `Client()` thread and `Server()` thread asynchronously in parallel.

Thread `Server()`**:**

while *no client uploads* **do**
  Wait for client update. **if** *client* m *uploads* (v_m, τ) **then**
    $t \leftarrow t+1$; $u_m^t \leftarrow$ `ServerUpdate` (v_m, τ, t); send $\left(u_m^t, t\right)$ to client m.

Thread `Client()`**:**

foreach *client* m *in parallel* **do**
  Receive pair (u_m, t) from server. Set local model $v_m \leftarrow u_m$, local timestamp $t_m \leftarrow t$. **while** *active* **do**
    if *choose to upload* **then**
      Define $J_m(v_m; u_m)$ as in (4). **foreach** *local iteration* h **do**
        $v_{m,h} \leftarrow v_{m,h-1} - \gamma \nabla J_m(v_{m,h-1}; u_m)$ `/* learning rate` γ `*/`
      Upload (v_m, t_m) and wait for server response.

Function `ServerUpdate`(v_m, τ, t)**:**

if $t - \tau > \tau_0$ **then**
  `/* Client deprecated. Cluster models stay, no updates needed. */`
  foreach $k \in [K]$ **do** $w_k^t \leftarrow w_k^{t-1}$ **return** $u_m^t = \sum_{k=1}^K \hat{\alpha}_{mk}^\tau w_k^t$.

$\hat{\alpha}_{m1}^t, ..., \hat{\alpha}_{mK}^t \leftarrow$ `DistributionEstimate` $(v_m, w_1^{t-1}, ..., w_K^{t-1}, D_1, ..., D_K, t)$
$\beta_{m1}^t, ..., \beta_{mK}^t \leftarrow$ `UpdateRatioCompute` $(\hat{\alpha}_{m1}^t, ..., \hat{\alpha}_{mK}^t, \beta_0, \tau, t)$ **foreach** $k \in [K]$ **do** $w_k^t \leftarrow \left(1 - \beta_{mk}^t\right) w_k^{t-1} + \beta_{mk}^t v_m$ **return** $u_m^t = \sum_{k=1}^K \hat{\alpha}_{mk}^t w_k^t$.

4.2 Server Update

Throughout the entire process of `CDFL`, the server passively waits for clients' update requests. Upon receipt of an uploaded model, the server first increments the epoch counter to obtain the current epoch t, then the server initiates a two-step evaluation process. Firstly, it checks if the client's model is too stale. Specifically, as client m uploads (v_m, τ), if $t - \tau > \tau_0$ for some preset staleness threshold τ_0, the server refrains from updating the cluster models, i.e., $w_k^t = w_k^{t-1}$ for all k, and returns the client a personalized model as an aggregation of current cluster models. Otherwise, the server proceeds to estimate client m's data distribution. Subsequently, it updates each cluster model, and constructs a new personalized model as an aggregation of the updated models for the client.

Distribution Estimation. Upon client m uploading v_m at epoch t (referred to as v_m^t for clarity), the estimation of its distribution hinges on several components, including v_m^t, the latest cluster models $w_1^{t-1}, \ldots, w_K^{t-1}$, and their proxy datasets $D_1, \ldots, D_K$. Two key metrics are evaluated to estimate the importance weights in client m's distribution. The first is the empirical loss of v_m^t on D_k, denoted by $F(v_m^t; D_k)$, for all $k \in [K]$. Specifically, when $F(v_m^t; D_k) < F(v_m^t; D_{k'})$,

v_m^t fits P_k better than $P_{k'}$, which indicates that cluster k should have a higher importance weight than cluster k' in client m's distribution P_m^t. The second metric is a measure on the similarity between v_m^t and the cluster models. This similarity can be materialized either through the loss difference between the cluster model and the client's model on proxy data, i.e., $|F(w_k^{t-1}; D_k) - F(v_m^t; D_k)|$, or the l_2 distance between the models, i.e., $\left\|v_m^t - w_k^{t-1}\right\|$. Leveraging these metrics, we propose `DistributionEstimation` in Algorithm 2 to estimate the importance weights of P_m^t as $\hat{\alpha}_{m1}^t, \ldots, \hat{\alpha}_{mK}^t$.

Cluster Updating. The server updates the model of each cluster k as,

$$w_k^t = (1 - \beta_{mk}^t) w_k^{t-1} + \beta_{mk}^t v_m^t, \tag{5}$$

where β_{mk}^t is the updating ratio contributed by client m to cluster k at epoch t. The value of β_{mk}^t depends on 1) the correlation between v_m^t and w_k^{t-1} as measured by the weight $\hat{\alpha}_{mk}^t$ evaluated for distribution estimation; and 2) the staleness of v_m^t indicated by the epoch index τ in which client m's model is lastly updated. Algorithm 2 shows procedures to compute the updating ratio.

4.3 Convergence Analysis

To assist the convergence analysis, we make the following assumptions that are standard in analyses of FL algorithms (see, e.g., (27; 5; 21)).

Assumption 1 *F_k is L_k-smooth and μ_k-convex and for $L_k, \mu_k > 0, k \in [K]$.*

Assumption 2 *Each client executes $H_{min} \leq$ local updates $\leq H_{max}$ before uploading.*

Assumption 3 *In (4), we simplify v_m^t as v denoting local model of client m at current epoch t, and u_m^τ as u denoting the latest model received from server. Let $f_m^t(v) \triangleq \frac{1}{n_m^t} \mathbb{E}_{(x^i, y^i) \sim P_m^t} [\sum_{i=1}^{n_m^t} l(v; x^i, y^i)]$, then $J_m^t(v; u) = f_m^t(v) + \frac{\rho}{2} \|v - u\|^2$. We assume $\forall m, \forall t \in T$, we have $\|\nabla f_m^t(v)\|^2 \leq V_1$ and $\|\nabla J_m^t(v; u)\|^2 \leq V_2$, for constants V_1 and V_2.*

Assumption 4 *The distance between optimal models of different clusters is bounded as $a_0 \Delta \leq \|w_k^* - w_{k'}^*\| \leq \Delta$, for some constant Δ, and $0 \leq a_0 \leq 1$, for all $k \neq k'$.*

Assumption 5 *The l_2-norm of cluster k's model is bounded as $\|w_k\|_2 \leq a_k \Delta$ with $a_k > 0$.*

Theorem 1. *For a client with model v and a cluster k, we consider consecutive S_k epochs, such that in each epoch the data of the client contains an non-zero component of P_k, and the client and cluster k updates v and w_k respectively as in Algorithm 1, then with the above assumptions, choosing $\rho \geq \frac{2V_1 + \frac{1}{2}\|v-u\|^2 + \sqrt{4\|v-u\|^2(1+V_1)\epsilon}}{2\|v-u\|^2}$ for all possible v and u, and a small constant $\epsilon > 0$, we have*

Algorithm 2 DistributionEstimation & UpdateRatioCompute

Function `DistributionEstimation`$(v_m, w_1, \ldots, w_K, D_1, \ldots, D_K, t)$**:**

  foreach $k \in [K]$ **do**

    $l_k \leftarrow F(v_m; D_k)$; $d_{1k} \leftarrow |F(w_k; D_k) - l_k|$; $d_{2k} \leftarrow \|v_m - w_k\|$

    $l_k \leftarrow l_k - l_{bar}$; $d_{1k} \leftarrow d_{1k} - d_{1bar}$; $d_{2k} \leftarrow d_{2k} - d_{2bar}$

    /* l_{bar}, d_{1bar}, d_{2bar} are hyper-parameters to control the scale. */

    $\hat{\alpha}^t_{mk} \leftarrow \frac{1}{K-1} \cdot \left(c_1 \cdot \frac{\sum_{i \neq k} l_i}{\sum_i l_i} + c_2 \cdot \frac{\sum_{i \neq k} d_{1i}}{\sum_i d_{1i}} + (1 - c_1 - c_2) \cdot \frac{\sum_{i \neq k} d_{2i}}{\sum_i d_{2i}} \right)$

    /* $c_1, c_2 \in [0, 1]$ are hyper-parameters. $c_1 + c_2 \in [0, 1]$. */

  $\hat{\alpha}^t_{m1}, \ldots, \hat{\alpha}^t_{mK} \leftarrow \mathrm{softmax}(\hat{\alpha}^t_{m1} \cdot A, ..., \hat{\alpha}^t_{mK} \cdot A)$

  /* $A > 0$ is the hyper-parameter to magnify the difference of estimation results of clusters. Estimated weights $\hat{\alpha}^t_{mk} \in [0, 1]$ for $k \in [K]$, and $\sum_{k=1}^K \hat{\alpha}^t_{mk} = 1$. */

Function `UpdateRaTioCompute`$(\hat{\alpha}^t_{m1}, \ldots, \hat{\alpha}^t_{mK}, \beta_0, \tau, t)$**:**

  foreach $k \in [K]$ **do**

    $\beta_{11}, \ldots, \beta_{1K} \leftarrow \hat{\alpha}^t_{m1}, \ldots, \hat{\alpha}^t_{mK}$; $\beta_{1max} \leftarrow \max(\beta_{1k})$.

    if $\beta_{1k} < \beta_{1bar}$ **then** $\beta_{1k} \leftarrow 0$; **else then** $\beta_{1k} \leftarrow \beta_{1k} / \beta_{1max}$; $e_k \leftarrow t$

    /* β_{1bar} is a preset threshold. Do not update cluster k when $\beta_{1k} < \beta_{1bar}$. e_k is the last epoch when cluster k is updated. */

    if $e_k - \tau < b$ **then** $\beta_{2k} \leftarrow 1$; **else then** $\beta_{2k} \leftarrow 1/\left(a(e_k - \tau) + 1\right)$

    /* a, b are hyper-parameters to control staleness. */

    $\beta^t_{mk} \leftarrow \beta_0 \cdot \beta_{1k} \beta_{2k}$ /* $\beta^t_{mk} \in [0, \beta_0]$; β_0 governs the maximal local model modification to the cluster model. */

  return $\beta^t_{m1}, ..., \beta^t_{mK}$

$$\mathbb{E}[\|\nabla F_k(v)\|^2] \leq \frac{\mathbb{E}[F_k(w_k^0) - F_k(w_k^{S_k})]}{\beta_0 \gamma \epsilon S_k H_{min}} + \frac{\left(\frac{L_k}{2} + \rho H_{max} + \frac{\rho H_{max}^2}{2}\right) \gamma H_{max} V_2}{\epsilon H_{min}} +$$
$$\frac{\sqrt{V_1}\left(2\sum_{i=1}^K a_i + (2K+1)a_k + K\right)\Delta}{\gamma \epsilon H_{min}} + \frac{\left(\frac{L_k}{2} + \rho\right)\left(2\sum_{i=1}^K a_i + (2K+1)a_k + K\right)^2 \Delta^2}{\gamma \epsilon H_{min}},$$

where w_k^0 and $w_k^{S_k}$ denotes cluster k's initial model and the model after S_k updates respectively.

Proof. Due to page limits, we only sketch the main steps here; the complete proofs are provided in Appendix B of the arXiv version (12).

Let $\mathbb{J}_k(v; u) = F_k(v) + \frac{\rho}{2}\|v - u\|^2$ and consider the local update $v_h = v_{h-1} - \gamma \nabla J_m(v_{h-1}; u_\tau)$. By Taylor expansion and L_k-smoothness, one-step descent yields

$$\mathbb{E}[F_k(v_h) - F_k(w^*)] \leq F_k(v_{h-1}) - F_k(w^*) - \gamma\epsilon\|\nabla F_k(v_{h-1})\|^2 + O(\gamma^2).$$

Summing over $h = 0, \ldots, H-1$ gives

$$\mathbb{E}[F_k(v_H) - F_k(u_\tau)] \leq -\gamma\epsilon \sum_{h=0}^{H-1} \|\nabla F_k(v_h)\|^2 + O(H\gamma^2). \tag{6}$$

For cluster update $w_s = (1 - \beta_s) w_{s-1} + \beta_s v_H$, convexity and smoothness imply

$$\mathbb{E}[F_k(w_s) - F_k(w_{s-1})] \leq -\beta_s \gamma \epsilon \sum_{h=0}^{H-1} \|\nabla F_k(v_h)\|^2 + O(\beta_s(\gamma^2 + \Delta^2)). \tag{7}$$

Finally, averaging across S_k updates and using $\sum_{s=1}^{S_k} H_s \geq S_k H_{\min}$, we obtain

$$\mathbb{E}\|\nabla F_k(v)\|^2 \;\leq\; O\Big(\tfrac{1}{\gamma S_k H_{\min}}\Big) + O(\gamma) + O(\frac{1}{\gamma}\Delta) + O\Big(\tfrac{1}{\gamma}\Delta^2\Big). \tag{8}$$

This establishes the stated convergence bound.

5 Experiments

5.1 Setup

We evaluate `CDFL` on synthetic clustered benchmarks and a digit recognition task. For synthetic data, we construct **Rotated FashionMNIST** (26), **Rotated CIFAR-100** (9), and **Rotated MiniImagenet-100** (23) by rotating each dataset by $i \cdot \frac{360}{K}$ degrees ($i = 0, \ldots, K-1$) to form K latent clusters, where $K = \{2, 3, 4, 6\}$. Each cluster contains 60k/10k, 50k/10k, and 48k/12k train/test samples, respectively. For **Digit Recognition**, we use MNIST+USPS (10; 7), SVHN (19), and CCPD (28) as three clusters.

Baselines. We compare `CDFL` with three types of baselines. 1) **Async but non-clustered:** `FedBuff` (20) and `CA2FL` (25), which maintain a single global model and test whether asynchronous aggregation alone is sufficient under dynamic mixture distributions. 2) **Cluster-aware:** `FedSoft-Async`, our asynchronous adaptation of `FedSoft` (21), where clients receive all cluster models and estimate mixture weights locally. 3) **No collaboration:** `Local`, where clients train independently. All baselines are trained on each client's current local data and evaluated on its current test distribution. Recent related methods such as CASA (15) and EchoPFL (13) are not included because no publicly available implementation could be fairly adapted to our dynamic mixture setting.

During initialization, clients use the averaged cluster model. Each client holds 500–2000 samples, with 40%–90% drawn from a dominant distribution and the rest from other clusters. After initialization, clients decide asynchronously when to update, and each upload-download cycle brings new local data. The number of clients is set to $20K$, and the staleness threshold is set to $\tau_0 = 20K$. On average, each client performs 25 updates on FashionMNIST, 20 on CIFAR-100 and MiniImagenet, and 40 on Digit Recognition with 100 clients. Unless otherwise specified, we set $\beta_0 = 0.025$, $a = 10$, $b = 5$, and $\rho = 0.1$. Detailed dataset-specific distribution-estimation parameters are deferred to the extended arXiv version (12). All experiments are conducted on a single NVIDIA RTX 4090 GPU. Each experiment is repeated 5 times, and we report the mean and standard deviation. Our code is publicly available at `https://github.com/chenqing-zhu/CCFL`.

5.2 Results

Accuracy. Final client and cluster accuracy are reported in Table 1. Overall, all collaborative methods outperform `Local`, confirming the benefit of server-assisted updating under dynamic distribution shifts. Among them, `CDFL` and

`FedSoft-Async` achieve the strongest overall results, while `CDFL` is generally better on cluster accuracy and attains the best or comparable client accuracy in most settings. Its advantage becomes more evident as the number of clusters increases, suggesting that server-side cluster inference is particularly helpful when client distributions are more heterogeneous. `FedBuff` remains competitive among the non-clustered asynchronous baselines, whereas `CA2FL` is less effective in our setting, likely because its calibration and scheduling mechanisms are less suited to clients with time-varying mixture distributions.

Table 1: Final post-refresh client accuracy and cluster accuracy on different datasets. "Client" denotes the final client accuracy after model refresh. Cluster accuracy is reported only for cluster-aware methods.

Dataset (Clusters)	`CDFL`		`FedSoft-Async`		`FedBuff`	`CA2FL`	`Local`
	Client	Cluster	Client	Cluster	Client	Client	Client
FashionMNIST (2)	.834±.003	.838±.007	.836±.003	.833±.008	.784±.004	.590±.001	.781±.019
FashionMNIST (3)	.823±.003	.835±.003	.818±.003	.823±.006	.771±.002	.592±.007	.746±.053
FashionMNIST (4)	.800±.006	.829±.005	.786±.007	.800±.017	.758±.007	.514±.005	.694±.072
FashionMNIST (6)	.788±.018	.818±.010	.761±.023	.769±.051	.750±.018	.507±.003	.697±.074
CIFAR-100 (2)	.392±.007	.417±.015	.397±.004	.416±.011	.338±.008	.347±.001	.280±.029
CIFAR-100 (3)	.303±.029	.362±.032	.295±.008	.348±.024	.296±.005	.248±.001	.208±.033
CIFAR-100 (4)	.376±.012	.432±.019	.361±.015	.430±.022	.334±.001	.336 ±.001	.261±.034
CIFAR-100 (6)	.304±.008	.376±.019	.292±.017	.371±.035	.299±.008	.383±.001	.214±.038
MiniImagenet (2)	.369±.003	.385±.006	.374±.003	.388±.002	.349±.002	.328±.004	.225±.030
MiniImagenet (3)	.314±.018	.359±.008	.308±.011	.353±.004	.307±.005	.274±.010	.180±.028
MiniImagenet (4)	.376±.014	.411±.007	.371±.009	.407±.008	.354±.004	.344±.009	.219±.029
MiniImagenet (6)	.334±.007	.386±.011	.323±.011	.380±.013	.300±.005	.295±.011	.192±.029
DigitRecognition	.883±.056	.890±.130	.890±.052	.870±.102	.855±.008	.582±.012	.820±.083

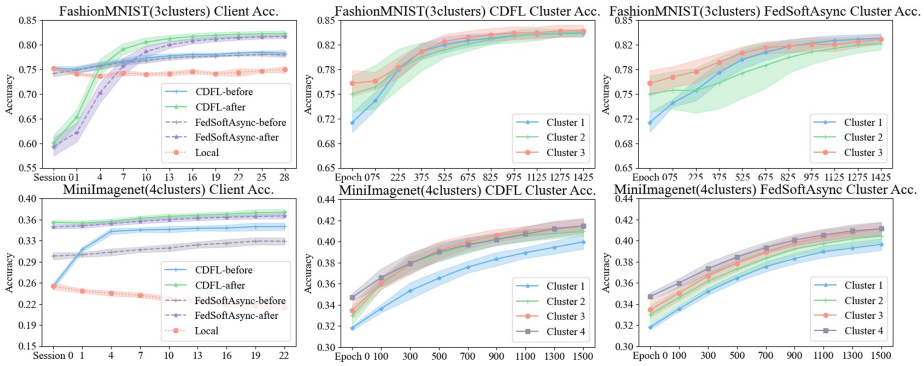
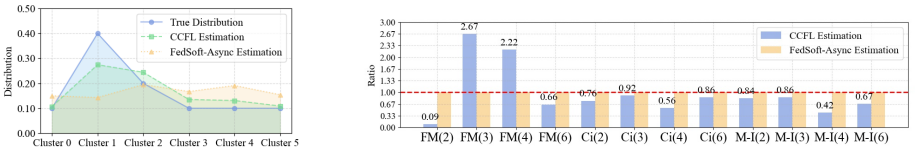


Fig. 3: Average accuracy of clients and clusters over time on FashionMNIST (3 clusters) and MiniImagenet (4 clusters). For client accuracy, in each session all the clients have updated their models for at least once; for cluster accuracy, exact one client updates its model with the sever in each epoch.

Fig. 3 further shows client and cluster accuracy over time. For client accuracy, both `CDFL` and `FedSoft-Async` improve clearly after model refresh compared with the corresponding pre-refresh performance, confirming the benefit of server interaction. On FashionMNIST, a short warm-up stage is observed before refreshed models consistently surpass locally trained ones, likely due to weak cluster models in early epochs. This phenomenon is less visible on MiniImagenet, where server-assisted updating helps from early stages.

Distribution Estimation. To assess the proposed distribution estimation method in Algorithm 2, we empirically compare the estimation outcomes of `CDFL` and those of `FedSoft-Async`. To quantify this assessment, we employ the KL-divergence metric $KL(P||Q)$, which measures the information loss when approximating the true distribution P with the estimated distribution Q. Lower KL divergence value implies more accurate estimation. As shown in Fig. 4(b), over all datasets and cluster numbers, `CDFL` constantly outperforms `FedSoft-Async`.



(a) Distribution estimation. (b) KL-Divergence of Distribution.

Fig. 4: (a) Estimated distributions in the MiniImagenet (6 clusters) experiment; (b) KL-divergence between the true distribution and the distribution estimated by CDFL and FedSoft-Async. FM(k): FashionMNIST (k clusters); Ci: CIFAR-100; M-I: MiniImagenet-100.

Communication and Computation Overhead. We compare the communication and computation overheads between `FedSoft-Async` and `CDFL` in Fig. 5. Specifically, we focus on download communication overhead, as both methods upload one local model to the server. We normalize both the communication and computation overhead of `CDFL` to 1, and measure the relative values of `FedSoft-Async`. Since clients in `CDFL` solely download a single aggregated model, and no additional computation beyond local model training is needed, the communication and computation overhead is significantly reduced.

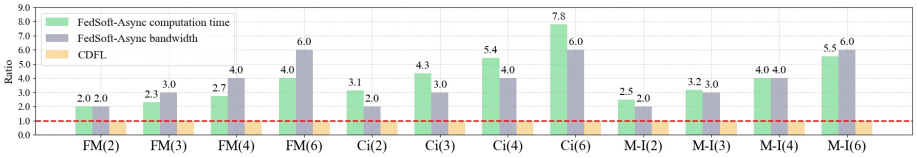


Fig. 5: Communication and computation overheads of `FedSoft-Async` and `CDFL`.

5.3 Ablation Study

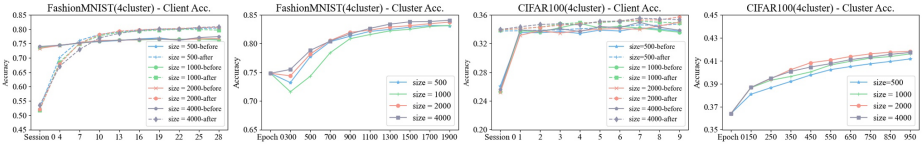


Fig. 6: Average accuracy for different sizes of proxy datasets at the server.

Size of Proxy Dataset. We evaluate the performance of `CDFL` for various sizes of public datasets on FashionMNIST(4 clusters) and CIFAR100(4clusters) in Fig. 6, ranging from 500 to 4000 samples. In those experiments, although the performance difference for using smaller sizes of proxy datasets has relatively

lower performances, but the difference is negligible. We demonstrate that even a proxy dataset size of 500 samples (**approximately 0.8% of the training data**) can yield relatively favorable performance outcomes. Therefore, we consider the inclusion of a small but available proxy dataset to be a reasonable and beneficial practice. Other ablation studies on values of ρ, a, b, and number of clients can be found in Appendix C of the arXiv version (12).

6 Conclusion

We presented `CDFL`, a client-driven federated learning framework for asynchronous personalization under dynamic mixture distributions. With server-side cluster inference, `CDFL` reduces client-side burden and adapts efficiently to distribution shifts. Theory and experiments validate its effectiveness.

Acknowledgements This work is supported in part by New Generation Artificial Intelligence-National Science and Technology Major Project (2025ZD0123504).

References

[1] Chen, Y., Ning, Y., Slawski, M., Rangwala, H.: Asynchronous online federated learning for edge devices with non-iid data. In: 2020 IEEE International Conference on Big Data (Big Data). pp. 15–24. IEEE (2020)
[2] Damaskinos, G., Guerraoui, R., Kermarrec, A.M., Nitu, V., Patra, R., Taiani, F.: Fleet: Online federated learning via staleness awareness and performance prediction. ACM Transactions on Intelligent Systems and Technology (TIST) **13**(5), 1–30 (2022)
[3] Deng, Y., Kamani, M.M., Mahdavi, M.: Adaptive personalized federated learning. arXiv preprint arXiv:2003.13461 (2020)
[4] Duan, M., Liu, D., Ji, X., Liu, R., Liang, L., Chen, X., Tan, Y.: Fedgroup: Efficient federated learning via decomposed similarity-based clustering. In: 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). pp. 228–237. IEEE (2021)
[5] Ghosh, A., Chung, J., Yin, D., Ramchandran, K.: An efficient framework for clustered federated learning. Advances in Neural Information Processing Systems **33**, 19586–19597 (2020)
[6] Ghosh, A., Mazumdar, A., et al.: An improved algorithm for clustered federated learning. arXiv preprint arXiv:2210.11538 (2022)
[7] Hull, J.J.: A database for handwritten text recognition research. IEEE Transactions on Pattern Analysis and Machine Intelligence **16**(5), 550–554 (1994)
[8] Khan, A.F., Wang, X., Le, Q., et al.: Ip-fl: Incentivized and personalized federated learning. arXiv preprint arXiv:2304.07514 (2023)
[9] Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
[10] LeCun, Y., Cortes, C., Burges, C.J.: Mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist/` (2010)
[11] Li, D., Wang, J.: Fedmd: Heterogenous federated learning via model distillation. arXiv preprint arXiv:1910.03581 (2019)
[12] Li, S., Zhu, C.: Towards client driven federated learning. arXiv preprint arXiv:2405.15407 (2024)

[13] Li, X., Liu, S., Zhou, Z., Guo, B., Xu, Y., Yu, Z.: Echopfl: Asynchronous personalized federated learning on mobile devices with on-demand staleness control. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies **8**(1), 1–22 (2024)
[14] Lin, T., Kong, L., Stich, S.U., Jaggi, M.: Ensemble distillation for robust model fusion in federated learning. Advances in Neural Information Processing Systems **33**, 2351–2363 (2020)
[15] Liu, B., Ma, Y., Zhou, Z., Shi, Y., Li, S., Tong, Y.: Casa: Clustered federated learning with asynchronous clients. In: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 1851–1862 (2024)
[16] Ma, J., Long, G., Zhou, T., Jiang, J., Zhang, C.: On the convergence of clustered federated learning. arXiv preprint arXiv:2202.06187 (2022)
[17] Mansour, Y., Mohri, M., Ro, J., Suresh, A.T.: Three approaches for personalization with applications to federated learning. arXiv preprint arXiv:2002.10619 (2020)
[18] McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)
[19] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y., et al.: Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning. vol. 2011, p. 4. Granada (2011)
[20] Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., Huba, D.: Federated learning with buffered asynchronous aggregation. In: Proceedings of The 25th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 151, pp. 3581–3607 (2022)
[21] Ruan, Y., Joe-Wong, C.: Fedsoft: Soft clustered federated learning with proximal local updating. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 8124–8131 (2022)
[22] Tang, X., Guo, S., Guo, J.: Personalized federated learning with contextualized generalization. arXiv preprint arXiv:2106.13044 (2021)
[23] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. Advances in neural information processing systems **29** (2016)
[24] Wang, X., Wang, Y.: Asynchronous hierarchical federated learning. arXiv preprint arXiv:2206.00054 (2022)
[25] Wang, Y., Cao, Y., Wu, J., Chen, R., Chen, J.: Tackling the data heterogeneity in asynchronous federated learning with cached update calibration. In: International Conference on Learning Representations. vol. 2024, pp. 16160–16192 (2024)
[26] Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)
[27] Xie, C., Koyejo, S., Gupta, I.: Asynchronous federated optimization. arXiv preprint arXiv:1903.03934 (2019)
[28] Xu, Z., Yang, W., Meng, A., Lu, N., Huang, H., Ying, C., Huang, L.: Towards end-to-end license plate detection and recognition: A large dataset and baseline. In: Proceedings of the European conference on computer vision (ECCV). pp. 255–271 (2018)
[29] Yu, H., Chen, Z., Zhang, X., Chen, X., Zhuang, F., Xiong, H., Cheng, X.: Fedhar: Semi-supervised online learning for personalized federated human activity recognition. IEEE transactions on mobile computing **22**(6), 3318–3332 (2021)
[30] Zhang, Y., Duan, M., Liu, D., Li, L., Ren, A., Chen, X., Tan, Y., Wang, C.: Csafl: A clustered semi-asynchronous federated learning framework. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–10. IEEE (2021)