BI-GRANULARITY WEIGHTED AGGREGATION FOR ASYNCHRONOUS FEDERATED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Asynchronous federated learning (AFL) enables collaborative model training under heterogeneous computation and communication latency, where participating clients may update and transmit their local models at substantially different speeds. Existing AFL methods typically discard updates from stragglers or reduce their aggregation weights according to update staleness. However, under severe system heterogeneity, such strategies may underutilise valuable client information, slow convergence, and increase discrepancies between the aggregated global model and the models that would have been obtained with more complete client participation. To address these challenges, we propose a novel bi-granularity weighted aggregation framework for AFL. The key idea is to reconstruct a pseudo local model for each straggling client, allowing federated aggregation to proceed by jointly incorporating received local models and reconstructed models for unavailable stragglers. To balance reconstruction quality and computational efficiency, rather than generating an entire model from scratch, our method reconstructs each pseudo model by reassembling neural blocks selected from available client models and leveraging the straggler's statistical fingerprints. During global aggregation, client contributions are jointly weighted at both the model level and neural-block level, enabling the server to account for heterogeneity at different granularities. Extensive experiments across diverse large-scale asynchronous settings demonstrate that the proposed method consistently improves performance over representative AFL baselines. Further analysis shows that the reconstructed pseudo models closely approximate the actual local models subsequently obtained from straggling clients, providing empirical evidence for the effectiveness of the proposed reconstruction mechanism.

1 INTRODUCTION

Federated learning (FL) enables multiple clients to collaboratively train a model while keeping their raw data local Kairouz et al. (2021); McMahan et al. (2017); Yin et al. (2021); Nguyen et al. (2021). In practical deployments, clients differ in computing power, network conditions, and device availability, so some take longer than others to train local models and upload their updates. Under synchronous aggregation, the server waits for the required client updates in each round, which can delay global model updates when some clients are slow. Asynchronous federated learning (AFL) allows global model updates to proceed alongside concurrent client training, without waiting for all selected clients to return their updates Huba et al. (2022); Wang et al. (2023a); Shenaj et al. (2023).

As a representative AFL method, FedBuff Nguyen et al. (2022) collects asynchronously arriving client updates and aggregates them once a predefined buffer threshold is reached. However, the global model may update through several versions before the stragglers arrive. The buffer can therefore contain stale updates computed from different earlier versions of the global model. These stale updates differ from those the same clients would compute from the current global model, potentially hindering convergence and degrading model performance.

Recent AFL methods seek to accelerate convergence by downweighting or discarding stale updates, or by correcting their update directions Sun et al. (2023); Hsu et al. (2019); Yu et al. (2024). These methods base their weighted aggregation solely on received model updates. Unfortunately, this reliance directly leads to the underutilization of valuable local information from stragglers. The buffer

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

may repeatedly fill with updates from faster clients, shifting the global update direction away from that obtained with more complete client participation. As system heterogeneity increases, this directional discrepancy can become more pronounced. *This raises a key question: how can information from stragglers be incorporated into the current global update in a timely manner without waiting for their actual updates?*

To address this challenge, we propose a bi-granularity weighted aggregation framework for AFL that incorporates both received client contributions and estimated straggler contributions. Our framework extends the scope of aggregation by retaining received local models as model-level contributions and reconstructing pseudo local models for stragglers at the neural-block level. By jointly weighting and aggregating these two types of contributions, we aim to approximate aggregation with information from all participating clients.

Specifically, our reconstruction method reuses trained neural blocks from available models rather than generating entire models from scratch. We group these blocks by representation similarity and use ordered block borrowing to construct diverse, structurally valid candidate models under a shared architecture. For each straggler, we then select a pseudo local model by matching candidate activation representations against its cached fingerprint, extracted on shared proxy inputs after a one-time local warm-up. These cached fingerprints enable client-specific reconstruction without access to the target models. Finally, we combine neural-block-level reconstructions for stragglers with model-level contributions from received local models to realize the proposed bi-granularity weighted aggregation.

We highlight the key contributions of our framework as follows:

- We propose a bi-granularity weighted aggregation framework that enables AFL to incorporate estimated contributions from clients whose updates have not yet arrived. It combines model-level contributions from received models with pseudo local models reconstructed at the neural-block level for stragglers.
- We develop a pseudo local model reconstruction method using limited available client models. Our ordered block-borrowing search expands candidate choice through diverse neural-block combinations under a shared architecture. Cached fingerprints enable specific model reconstruction by guiding pseudo-model selection without access to the stragglers' models.
- Experiments show accuracy gains over representative AFL baselines across diverse asynchronous settings, supporting the proposed framework. Large-scale evaluations examine sensitivity to client population and concurrency, while ablations support the benefits of the search design, source diversity, and client-specific matching.

2 Related Work

Asynchronous Federated Learning. AFL reduces synchronization waiting by allowing global updates to proceed before all participating clients finish their local tasks. FedBuff Nguyen et al. (2022) supports buffered asynchronous aggregation by collecting a predefined number of client updates before updating the global model. The collected updates may originate from earlier global model versions, motivating methods that account for update staleness. Momentum-based approaches use accumulated global update information to guide optimization Zang et al. (2024); Yan et al. (2025). MA further decomposes momentum-based stale updates into constituent momentum terms and assigns combinatorial weights to reduce momentum bias Yu et al. (2024). Adaptive optimization offers another approach by adjusting effective step sizes using accumulated optimization statistics Wang et al. (2024c; 2022); Reddi et al. (2020). These approaches primarily modify how received updates influence global optimization.

Our bi-granularity weighted aggregation framework extends AFL aggregation beyond clients whose updates have arrived by combining received model-level contributions with neural-block-level reconstructions for stragglers, without waiting for their actual updates.

Model Reassembly. Model reassembly (MR) transfers knowledge by constructing new models from trained neural blocks. Early work explored MR in centralized settings and formulated block selection and assembly as optimization problems Yang et al. (2022).

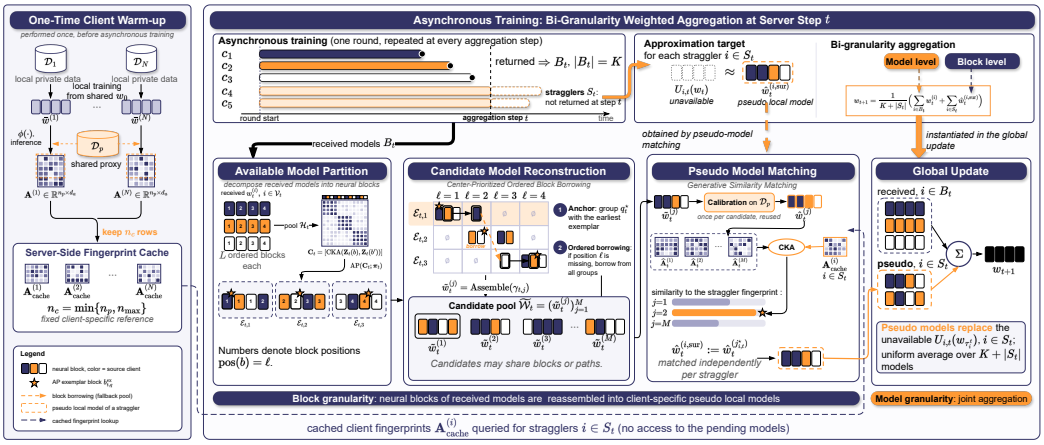


Figure 1: Overview of the proposed bi-granularity aggregation framework. Cached client fingerprints guide the selection of straggler-specific pseudo local models reconstructed from available neural blocks. The selected pseudo models are jointly aggregated with received local models.

In FL, pFedHR Wang et al. (2023b) adapted MR to construct client specific models. Subsequent studies refined different aspects of this process. pFedClub Wang et al. (2024a) introduced ordering constraints to organize block search and control candidate model size. LegoFL Leng et al. (2026) combined consensus and personalized learning with grafting-based assembly to improve knowledge integration and reduce reassembly costs. FedGraft Liu et al. (2025) addressed memory constraints by selectively reassembling neural blocks while largely retaining the local model architecture.

We use MR to reconstruct pseudo local models for stragglers from limited local information by combining neural blocks from different clients in diverse ways under a shared architecture. Cached client fingerprints guide straggler-specific selection without access to the pending local models, enabling the reconstructed contributions to complement received models in global aggregation.

3 Proposed Method: Bi-Granularity Weighted Aggregation

Our framework combines received model-level contributions with neural-block-level reconstructions for stragglers. Figure 1 provides an overview, with details presented below.

3.1 Approximation Target and Aggregation Rule

At aggregation step t, the server has received local models from clients in B_t. Let S_t denote the clients with pending tasks selected for model completion, where $B_t \cap S_t = \varnothing$. For each $i \in S_t$, the unavailable target is $U_{i,t}(w_{\tau_i^t})$: the local model produced by its pending task initialized from the global version τ_i^t.

If the true local models of all clients in $B_t \cup S_t$ were available, their average would give the reference aggregate:

$$w_{t+1}^{\text{ref}} := \frac{1}{K + |S_t|} \sum_{i \in B_t \cup S_t} U_{i,t}(w_{\tau_i^t}). \tag{1}$$

This reference retains each client's actual task initialization; it does not assume that all clients train from the current global model w_t.

Rather than waiting for the missing local models to arrive, we reconstruct a substitute $\hat{w}_t^{(i,\text{sur})}$ for each $i \in S_t$ to approximate $U_{i,t}(w_{\tau_i^t})$. Replacing the unavailable models in Eq. equation 1 with these substitutes gives the actual global update:

$$w_{t+1} = \frac{\sum_{i \in B_t} U_{i,t}(w_{\tau_i^t}) + \sum_{i \in S_t} \hat{w}_t^{(i,\text{sur})}}{K + |S_t|}. \tag{2}$$

Each represented client contributes one model, either received or reconstructed. The distinction from received-model averaging is the inclusion of estimated contributions from S_t, rather than a change in the averaging weights.

The central design problem is therefore to reconstruct client specific pseudo local models without access to the corresponding target models. To this end, the following subsections explain how client warm-up, available model partition, candidate model reconstruction, and pseudo model matching work together to obtain these approximations.

3.2 Client Warm-up

Approximating a pending local model requires a reference that distinguishes the target client from other clients. Since the target model itself is unavailable, we obtain this reference before asynchronous training through a one-time local warm-up. The resulting activation fingerprint is cached for subsequent pseudo model matching.

Each client first trains locally from a shared initialization using its private dataset. Let $\bar{w}^{(i)}$ denote the resulting model, and let $\phi_{\bar{w}^{(i)}}(x) \in \mathbb{R}^{d_a}$ denote the activation of a designated layer. Let $(x_r)_{r=1}^{n_p}$ be a fixed ordering of the shared proxy inputs in $\mathcal{D}_p$, where $n_p = |\mathcal{D}_p|$. Evaluating the warm-up model in inference mode yields:

$$\mathbf{A}^{(i)} := \begin{bmatrix} \phi_{\bar{w}^{(i)}}(x_1)^\top \\ \vdots \\ \phi_{\bar{w}^{(i)}}(x_{n_p})^\top \end{bmatrix} \in \mathbb{R}^{n_p \times d_a}. \tag{3}$$

For subsequent matching, the server stores a row-limited fingerprint:

$$\mathbf{A}^{(i)}_{\text{cache}} := \mathbf{A}^{(i)}_{1:n_c,:}, \qquad n_c = \min\{n_p, n_{\max}\}. \tag{4}$$

Here, $n_{\max}$ is the maximum number of proxy samples retained for fingerprint matching. A valid comparison requires client and candidate fingerprints to use the same proxy inputs in the same order, with a consistent activation extraction protocol.

Local warm-up makes the representation depend on the client's private training data, while shared proxy inputs provide a common basis for comparing models. The cached fingerprint serves as a fixed client specific reference, rather than an observation of the pending local model. Its use in selecting reconstructed models is described in the pseudo model matching subsection.

3.3 Available Model Partition

At each aggregation step, we decompose the available source models into neural blocks and organize them by representation similarity. Let $\varnothing \neq \mathcal{V}_t \subseteq B_t$ index the source clients, and write $w_t^{(i)} := U_{i,t}(w_{\tau_i^t})$ for their received local models. The source collection is $\mathcal{W}_t := \{w_t^{(i)}\}_{i \in \mathcal{V}_t}$. Since these models share an architecture, each is decomposed into an ordered sequence of L blocks, $(b_{t,1}^{(i)}, \ldots, b_{t,L}^{(i)})$. The resulting block pool is:

$$\mathcal{H}_t := \left\{ b_{t,\ell}^{(i)} \mid i \in \mathcal{V}_t,\ 1 \le \ell \le L \right\}. \tag{5}$$

Each block retains its source client and position identifiers, which are used to constrain assembly.

Grouping blocks only by position does not account for differences in their learned representations. Following recent MR methods Leng et al. (2026); Wang et al. (2024a), we organize the pool using activation similarity. For each $b \in \mathcal{H}_t$, let $\mathbf{Z}_t(b)$ denote its output activations evaluated within its source model on shared proxy inputs. Their pairwise similarity matrix is:

$$\mathbf{C}_t := \left[\mathrm{CKA}\big(\mathbf{Z}_t(b), \mathbf{Z}_t(b')\big) \right]_{b,b' \in \mathcal{H}_t}, \tag{6}$$

where centered kernel alignment (CKA) measures similarity between the sampled representations rather than exact functional equivalence. We apply Affinity Propagation (AP) Frey & Dueck (2007) to obtain similarity groups and their representative blocks:

$$\big(\mathcal{E}_{t,q}, b_{t,q}^{\text{ex}}\big)_{q=1}^{Q_t} := \mathrm{AP}\big(\mathbf{C}_t; \boldsymbol{\pi}_t\big). \tag{7}$$

Here, $\boldsymbol{\pi}_t$ specifies the self-preference values. The groups $\{\mathcal{E}_{t,q}\}_{q=1}^{Q_t}$ partition $\mathcal{H}_t$, and each exemplar $b_{t,q}^{\mathrm{ex}} \in \mathcal{E}_{t,q}$ is an actual source block.

AP provides representative blocks without requiring a prescribed number of groups; its preferences influence the resulting grouping. These exemplars serve as anchors for candidate construction in the next subsection. The groups organize blocks by representation similarity, while candidate reconstruction must also respect block order and input–output compatibility.

3.4 Candidate Model Reconstruction

Using the similarity groups in Eq. equation 7, we construct candidate models by selecting and assembling source blocks. Under the shared architecture, the search varies block sources rather than network topology: each candidate contains one block at each of the L prescribed positions. We propose *Center-Prioritized Ordered Block Borrowing*, which combines exemplar-guided anchor selection with position-wise block borrowing to construct complete assembly paths.

Center-Prioritized Anchor Selection. Fixing a single first-position block restricts the candidate combinations at the start of each path. We instead select an anchor group using its exemplar and sample first-position blocks from that group.

Let $\mathrm{pos}(b)$ denote the position identifier retained for block b, so that $\mathrm{pos}(b_{t,\ell}^{(i)}) = \ell$. For each group, define its position specific subset as:

$$\mathcal{E}_{t,q}^{(\ell)} := \{b \in \mathcal{E}_{t,q} \mid \mathrm{pos}(b) = \ell\}. \tag{8}$$

Among groups containing first-position blocks, we prioritize the group whose exemplar occurs earliest in the source architecture:

$$q_t^\star \in \operatorname*{arg\,min}_{q:\, \mathcal{E}_{t,q}^{(1)} \neq \varnothing} \mathrm{pos}\big(b_{t,q}^{\mathrm{ex}}\big). \tag{9}$$

Ties are resolved by a fixed rule. For each candidate $j \in \{1, \ldots, M\}$, we sample $\tilde{b}_{t,1}^{(j)} \sim \mathrm{Uniform}(\mathcal{E}_{t,q_t^\star}^{(1)})$ and initialize its group index as $q_{t,1}^{(j)} = q_t^\star$. Sampling with replacement permits anchor reuse while allowing different starting blocks when the pool contains multiple choices.

Ordered Block Borrowing. An anchor group need not contain blocks at every remaining position. Restricting assembly to that group could therefore leave the path incomplete. We address this limitation during construction by borrowing a block whenever the current group lacks the next required position.

For $\ell = 2, \ldots, L$, let $q_{t,\ell-1}^{(j)}$ be the group containing the previously selected block. The sampling pool for position ℓ is:

$$\mathcal{P}_{t,\ell}^{(j)} := \begin{cases} \mathcal{E}_{t,q_{t,\ell-1}^{(j)}}^{(\ell)}, & \text{if } \mathcal{E}_{t,q_{t,\ell-1}^{(j)}}^{(\ell)} \neq \varnothing, \\ \displaystyle\bigcup_{q=1}^{Q_t} \mathcal{E}_{t,q}^{(\ell)}, & \text{otherwise.} \end{cases} \tag{10}$$

We sample $\tilde{b}_{t,\ell}^{(j)} \sim \mathrm{Uniform}(\mathcal{P}_{t,\ell}^{(j)})$ and set $q_{t,\ell}^{(j)}$ to the index of its containing group. Because $\mathcal{V}_t \neq \varnothing$ and every source model contains all L positions, the fallback pool is nonempty. The procedure therefore selects one block at every position without a separate step for filling missing positions.

The completed path and its reconstructed model are:

$$\begin{aligned} \gamma_{t,j} &:= \big(\tilde{b}_{t,1}^{(j)}, \ldots, \tilde{b}_{t,L}^{(j)}\big), \\ \tilde{w}_t^{(j)} &:= \mathrm{Assemble}(\gamma_{t,j}). \end{aligned} \tag{11}$$

All source models use common block boundaries, and assembly preserves the prescribed connections and position-compatible interfaces. Repeating the procedure yields the indexed candidate pool $\widetilde{\mathcal{W}}_t := (\tilde{w}_t^{(j)})_{j=1}^M$. Candidates may share blocks or even complete paths; the procedure permits different source combinations but does not require all M models to be distinct. The next subsection selects a pseudo local model for each straggler from this pool.

3.5 PSEUDO-MODEL MATCHING

For each straggler, we select a reconstructed candidate as its pseudo local model by comparing candidate activation representations with the client's cached fingerprint. Since the target local model is unavailable, this comparison provides a client specific selection criterion rather than a direct measurement of reconstruction error. We refer to this procedure as *Generative Similarity Matching*.

Candidate Calibration. Before matching, each candidate $\tilde{w}_t^{(j)}$ is fine-tuned on proxy data to obtain a calibrated model $\hat{w}_t^{(j)}$. This step is intended to reduce representation mismatch between blocks after reassembly, while preserving the assembled architecture. Calibration is performed once per candidate at aggregation step t, and the resulting model is reused across client matching queries.

Fingerprint-Based Matching. We evaluate each calibrated model on the same n_c proxy inputs retained in the cached client fingerprints, using the same input order and activation extraction convention:

$$\widehat{\mathbf{A}}_t^{(j)} := \begin{bmatrix} \phi_{\hat{w}_t^{(j)}}(x_1)^\top \\ \vdots \\ \phi_{\hat{w}_t^{(j)}}(x_{n_c})^\top \end{bmatrix} \in \mathbb{R}^{n_c \times d_a}. \tag{12}$$

Activations are extracted in inference mode. For each $i \in S_t$, we select:

$$j_{i,t}^\star \in \arg\max_{1 \le j \le M} \mathrm{CKA}\left(\mathbf{A}_{\mathrm{cache}}^{(i)}, \widehat{\mathbf{A}}_t^{(j)}\right). \tag{13}$$

Ties are broken by candidate order. Matching is performed independently for each straggler, so the same candidate may be selected for multiple clients. The pseudo local model introduced in Section 3.1 is instantiated as:

$$\hat{w}_t^{(i,\mathrm{sur})} := \hat{w}_t^{(j_{i,t}^\star)}, \qquad i \in S_t. \tag{14}$$

It approximates the pending local model $U_{i,t}(w_{\tau_i^t})$. The server inserts these pseudo local models into Eq. equation 2 alongside the received models, without waiting for the corresponding updates.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Datasets and Networks. We evaluate on SVHN Netzer et al. (2011), CIFAR-10, and CIFAR-100 Krizhevsky et al. (2009), using ResNet-18 He et al. (2016) as the default backbone. More details are in Appendix A.

Baselines. We compare against representative AFL methods, including FedBuff Nguyen et al. (2022), CA^{2}FL Wang et al. (2024b), FADAS Wang et al. (2024c), and AdaMasFL Yan et al. (2025).

Asynchronous Simulation. We consider three challenging asynchronous settings. The first uses a physics-based asynchronous simulator Yang & Liu (2025). The other two sample client delays from predefined intervals Wang et al. (2024c). Both use the intervals $[1, 2]$, $[3, 5]$, and $[50, 80]$, with respective client proportions of $(0.5, 0.3, 0.2)$ and $(0.4, 0.2, 0.4)$. The latter increases the fraction of clients in the highest-delay interval from 20% to 40%.

Implementation and Hyperparameters. Implementation details and hyperparameter settings are provided in Appendix B.

4.2 PERFORMANCE COMPARISON

Performance under Non-IID Settings (Main task). Table 1 reports test accuracy across three datasets, three Dirichlet concentration parameters, and three asynchronous settings. GUA achieves

Table 1: Test accuracy (%) under different statistical heterogeneity levels (α) and straggler settings. The best and second-best results are highlighted in red and blue, respectively.

Setting	Method	SVHN			CIFAR-10			CIFAR-100		
		0.6	0.3	0.1	0.6	0.3	0.1	0.6	0.3	0.1
Real-world (Stragglers 10-30%)	FedBuff	94.53±0.15	93.00±0.78	90.36±0.34	84.57±0.30	75.55±0.36	56.23±1.46	63.47±0.17	61.08±0.35	53.64±0.48
	CA^{2}FL	94.13±0.06	93.19±0.45	91.09±0.29	87.60±1.16	85.08±0.28	67.88±0.72	66.97±0.27	64.66±0.60	62.24±0.68
	FADAS	92.82±0.63	90.12±1.67	84.75±1.52	83.48±0.19	80.70±0.33	77.24±0.83	58.36±0.40	55.47±0.67	49.24±1.49
	AdaMasFL	90.42±0.25	86.71±5.31	89.17±0.26	59.89±0.16	57.80±0.22	42.44±1.60	22.69±0.11	21.72±0.28	21.20±0.32
	Ours	95.24±0.10	94.83±0.67	92.30±0.23	87.74±0.24	85.99±0.42	73.88±0.60	67.53±1.24	65.84±0.16	62.44±1.07
Uniform w/ Delay (Stragglers 30-40%)	FedBuff	94.64±0.17	93.97±0.67	91.14±0.64	84.27±1.11	74.58±2.47	61.22±4.11	63.81±0.41	61.60±0.21	54.54±1.09
	CA^{2}FL	91.84±1.22	84.12±2.69	36.26±3.19	55.72±3.12	49.36±2.86	23.33±1.88	36.96±1.18	30.54±3.69	12.69±1.92
	FADAS	94.58±0.64	93.28±0.48	91.59±0.29	84.05±0.90	82.29±0.43	70.92±0.43	59.49±0.60	57.79±1.06	52.60±1.02
	AdaMasFL	89.23±0.20	89.42±0.48	87.35±0.44	60.50±0.25	58.17±0.12	43.94±2.27	22.78±0.16	21.81±0.14	21.65±0.15
	Ours	94.93±0.36	94.42±0.02	93.35±0.62	85.15±0.63	82.33±1.08	72.12±0.76	64.42±0.75	62.09±0.93	57.37±1.81
Uniform w/ Delay (Stragglers 40-60%)	FedBuff	94.86±0.05	93.47±0.32	92.77±0.50	76.52±0.97	63.49±0.57	62.15±1.87	63.95±0.65	61.54±1.04	54.22±0.88
	CA^{2}FL	90.25±1.63	87.00±4.19	64.42±4.94	53.76±5.86	46.10±2.11	23.68±2.52	36.66±0.74	31.88±1.76	13.30±2.47
	FADAS	94.21±0.06	92.91±0.65	90.23±0.71	81.76±0.61	81.37±1.93	71.71±1.30	57.91±0.65	56.58±0.35	51.69±0.60
	AdaMasFL	89.36±0.24	89.16±0.30	87.23±0.47	60.71±0.13	58.56±0.13	45.62±0.22	21.89±0.23	21.83±0.33	21.71±0.27
	Ours	95.02±0.68	94.17±0.20	93.46±0.68	84.33±0.58	82.33±1.08	72.64±1.23	62.74±0.67	60.71±0.50	54.50±0.24

the highest mean accuracy in 24 of the 27 configurations and ranks second in the remaining three. The gains are measured against different strongest baselines across settings. For example, under the second delay-based setting, GUA improves CIFAR-100 accuracy at $\alpha = 0.1$ from 54.54% to 57.37%, a gain of 2.83 percentage points over FedBuff. Under the third setting, it improves CIFAR-10 accuracy at $\alpha = 0.6$ from 81.76% to 84.33%, outperforming FADAS by 2.57 percentage points.

Comparing two delay-based settings reveals different sensitivities to a larger high-delay client fraction. The absolute accuracy change of GUA is at most 0.25 percentage points on SVHN and 0.82 on CIFAR-10. CIFAR-100 is more sensitive, with decreases of 1.68, 1.38, and 2.87 percentage points at $\alpha = 0.6$, 0.3, and 0.1, respectively. Thus, the method maintains similar accuracy across the two settings on SVHN and CIFAR-10, but remains affected by increased client delays on CIFAR-100.

The advantage is not uniform across all conditions. In the simulator-based setting, FADAS exceeds GUA by 3.36 percentage points on CIFAR-10 at $\alpha = 0.1$. In the third setting, FedBuff leads by 1.21 and 0.83 percentage points on CIFAR-100 at $\alpha = 0.6$ and 0.3, respectively. These exceptions show that the benefit of model reconstruction depends on the combination of task, data heterogeneity, and asynchronous conditions, rather than increasing uniformly with their severity.

Performance with Larger Client Populations. Table 2 examines GUA with 500 clients and concurrency ratios of 0.1 and 0.2, using the default 100-client configuration with a concurrency ratio of 0.1 as a reference. At the same concurrency ratio of 0.1, increasing the population to 500 reduces accuracy by 1.20–1.89 percentage points on SVHN, 2.61–3.88 on CIFAR-10, and 5.02–7.86 on CIFAR-100. Under a fixed total training-data budget, this comparison also involves smaller local datasets, rather than a change in client count alone.

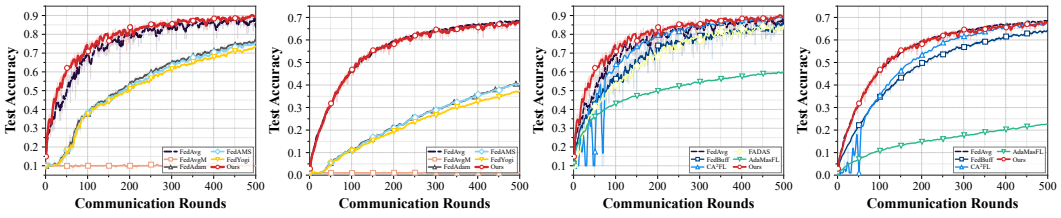
Increasing the concurrency ratio from 0.1 to 0.2 improves accuracy in all nine configurations, with gains ranging from 0.28 to 3.88 percentage points. At $\alpha = 0.1$, the improvements are 1.98, 3.88, and 2.36 percentage points on SVHN, CIFAR-10, and CIFAR-100, respectively. The resulting SVHN and CIFAR-10 accuracies exceed the default configuration by 0.40 and 1.27 percentage points. However, CIFAR-100 remains 3.05–5.50 percentage points below the default configuration. Higher concurrency therefore offsets part of the accuracy loss in the larger population, but does not remove the sensitivity to client scale across all datasets.

4.3 Convergence Behavior

Fig. 2(c)–(d) compares training trajectories under the simulator-based asynchronous setting with $\alpha = 0.6$. At approximately 200 server updates, GUA reaches about 83% accuracy on CIFAR-10

Table 2: Test accuracy (%) of our framework across client populations and concurrency ratios under the simulator-based asynchronous setting. The default configuration uses 100 clients and a concurrency ratio of 0.1.

Multiplier	SVHN			CIFAR-10			CIFAR-100		
	0.6	0.3	0.1	0.6	0.3	0.1	0.6	0.3	0.1
500/0.1	93.35±0.35	93.63±0.07	90.72±0.65	83.86±0.47	82.49±1.07	71.27±0.88	62.51±0.05	60.33±0.53	54.58±0.50
500/0.2	93.76±0.27	93.91±0.30	92.70±0.07	85.90±0.76	84.86±0.59	75.15±0.53	63.45±0.58	62.79±0.81	56.94±0.72
Ours	95.24±0.10	94.83±0.67	92.30±0.23	87.74±0.24	85.99±0.42	73.88±0.60	67.53±1.24	65.84±0.16	62.44±1.07



(a) CIFAR-10: conventional. (b) CIFAR-100: conventional. (c) CIFAR-10: AFL. (d) CIFAR-100: AFL.

Figure 2: Convergence comparisons under the simulator-based asynchronous setting with $\alpha = 0.6$. Panels (a)–(b) compare conventional federated optimizers, while panels (c)–(d) compare AFL baselines with FedAvg included as a reference.

and 59% on CIFAR-100, compared with 79% and 54% for CA^{2}FL, and 74% and 50% for FedBuff. The advantage is most visible during early and intermediate training, while CA^{2}FL narrows the gap later. On CIFAR-100, however, FedAvg follows a nearly identical trajectory to us, so the advantage is not universal across all plotted methods. Additional comparisons with conventional federated optimizers are shown in Fig. 2(a)–(b).

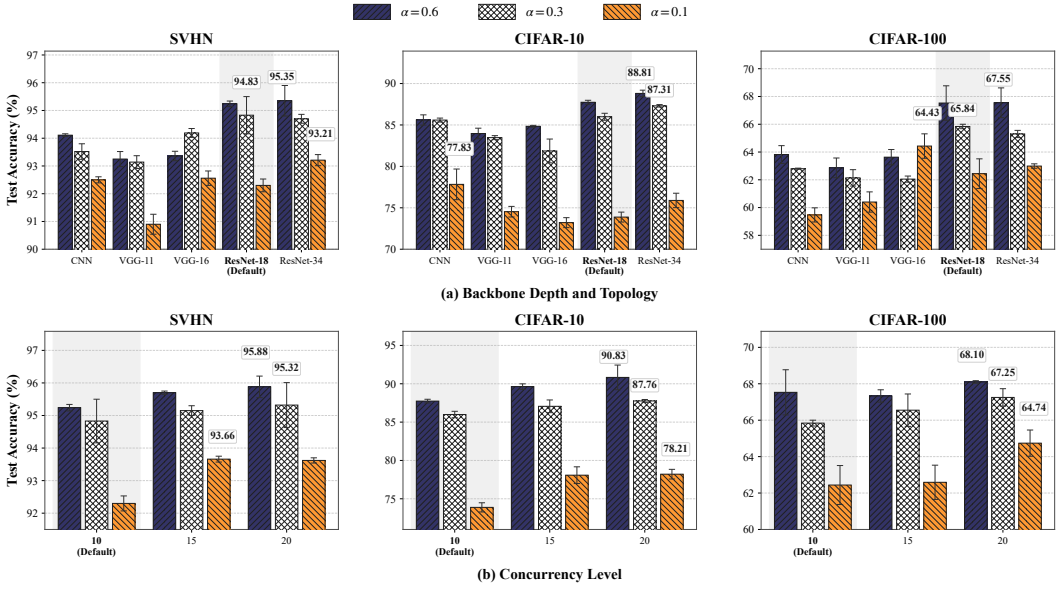


Figure 3: Sensitivity to backbone architecture and concurrency level under the real-world straggler setting. Panels (a) and (b) evaluate backbone depth and topology and the concurrency level, respectively. Gray regions highlight the default settings, and the annotated values indicate the highest mean accuracy for each Non-IID level.

4.4 Sensitivity Analysis

4.4.1 Architecture and Concurrency

(1) **Backbone Architecture.** Figure 3(a) compares the backbones. Accuracy does not increase monotonically with network depth. On CIFAR-10, ResNet-34 performs best at $\alpha = 0.6$ and 0.3, whereas the shallower CNN achieves the highest accuracy of 77.83% at $\alpha = 0.1$. On CIFAR-100 at $\alpha = 0.1$, VGG-16 performs best at 64.43%. These results indicate that backbone selection depends on both the dataset and statistical heterogeneity, rather than depth alone. (2) **Concurrency Level.** Figure 3(b) shows that increasing concurrency from 10 to 20 improves mean accuracy in all nine configurations. At $\alpha = 0.1$, CIFAR-10 improves from 73.88% to 78.21%, and CIFAR-100 from 62.44% to 64.74%. The improvement is not monotonic in every case: SVHN at $\alpha = 0.1$ peaks at concurrency 15 and decreases slightly at 20. Higher concurrency is therefore beneficial overall, but its marginal benefit depends on the task and data distribution.

4.4.2 Reconstruction Hyperparameters

(1) **Client Warm-up Epochs.** Figure 4(a) in Appendix. C shows that five warm-up epochs yield lower mean accuracy than the default 15 epochs in all nine configurations. Further increases do not produce monotonic gains. For example, extending warm-up from 15 to 100 epochs improves CIFAR-10 accuracy by only 0.30, 0.55, and 0.95 percentage points at $\alpha = 0.6$, 0.3, and 0.1, respectively. Longer warm-up can improve accuracy, but the additional benefit varies across configurations. (2) **Candidate Pool Size.** Figure 4(b) varies the candidate multiplier $M_t/|S_t|$. Increasing the multiplier from $1\times$ to $15\times$ improves mean accuracy in all nine configurations, but increasing it beyond the default $10\times$ is not uniformly beneficial. On CIFAR-100, $15\times$ improves accuracy at $\alpha = 0.1$ from 62.44% to 63.96%, whereas $10\times$ remains best at $\alpha = 0.3$. A larger search budget therefore offers potential gains without guaranteeing better selected pseudo models. (3) **Candidate Calibration Epochs.** Figure 4(c) shows that the default calibration budget of 10 epochs achieves the highest mean accuracy in all nine configurations. Increasing the budget to 20 epochs lowers accuracy in every configuration relative to 10 epochs. These results support the chosen calibration budget within the tested range and show that additional candidate fine-tuning does not necessarily improve the final global model.

4.5 Ablation Study

Table 4 in Appendix D compares five ablation variants with the full framework. (1) **Search Rules.** Random anchor selection reduces accuracy by 1.41–5.39 percentage points across the evaluated settings. Replacing ordered search with random block search lowers CIFAR-10 accuracy by 3.63 and 7.15 percentage points at $\alpha = 0.6$ and 0.1, respectively. Removing block borrowing has its largest effect on CIFAR-100 at $\alpha = 0.1$, reducing accuracy from 62.44% to 54.78%. These results support the combined use of our design details. (2) **Source Model Diversity.** Restricting reconstruction to blocks from only 1–2 clients reduces accuracy by 2.36–4.46 percentage points. The consistent decrease supports using a broader source model pool rather than restricting reconstruction to a few clients. (3) **Client Specific Matching.** Replacing fingerprint-based matching with random assignment causes the largest average decrease among the tested variants: 6.14 percentage points across the four configurations. The loss reaches 9.89 percentage points on CIFAR-10 at $\alpha = 0.1$. Thus, constructing candidate models alone is insufficient to retain the full framework's performance; selecting them for individual stragglers provides an additional benefit.

5 Conclusion

We presented a bi-granularity AFL framework that combines neural-block-level reconstruction with model-level aggregation to incorporate estimated straggler contributions without waiting for their actual updates. By reusing trained blocks and selecting reconstructed candidates through cached client fingerprints, the framework applies MR to approximate unavailable local models. Experiments show improved accuracy over representative AFL baselines in most evaluated settings.

AI USE STATEMENT

In this work, We used generative AI tools to assist with translation, improve the readability of the manuscript, and refine the visual presentation of figures. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. Papaya: Practical, private, and scalable federated learning. *Proceedings of Machine Learning and Systems*, 4:814–832, 2022.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Zeqi Leng, Chunxu Zhang, Guodong Long, and Bo Yang. Lego-fl: Learning heterogeneous federated models as a lego assembly games. In *Forty-third International Conference on Machine Learning*. PMLR, 2026.

Ruixuan Liu, Ming Hu, Zeke Xia, Xiaofei Xie, Jun Xia, Pengyu Zhang, Yihao Huang, and Mingsong Chen. Fedgraft: Memory-aware heterogeneous federated learning via model grafting. *IEEE Transactions on Mobile Computing*, 24(12):13506–13519, 2025. doi: 10.1109/TMC.2025.3591537.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 4. Granada, 2011.

Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE communications surveys & tutorials*, 23(3):1622–1658, 2021.

John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *International conference on artificial intelligence and statistics*, pp. 3581–3607. PMLR, 2022.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

Donald Shenaj, Marco Toldo, Alberto Rigon, and Pietro Zanuttigh. Asynchronous federated continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5055–5063, 2023.

Hao Sun, Li Shen, Shixiang Chen, Jingwei Sun, Jing Li, Guangzhong Sun, and Dacheng Tao. Fedlalr: Client-specific adaptive learning rates achieve linear speedup for non-iid data. *arXiv preprint arXiv:2309.09719*, 2023.

Ewen Wang, Boyi Chen, Mosharaf Chowdhury, Ajay Kannan, and Franco Liang. Flint: A platform for federated learning integration. *Proceedings of Machine Learning and Systems*, 5:21–34, 2023a.

Jiaqi Wang, Xingyi Yang, Suhan Cui, Liwei Che, Lingjuan Lyu, Dongkuan Xu, and Fenglong Ma. Towards personalized federated learning via heterogeneous model reassembly. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023b. Curran Associates Inc.

Jiaqi Wang, Qi Li, Lingjuan Lyu, and Fenglong Ma. Controllable heterogeneous model aggregation for personalized federated learning. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024a. Curran Associates Inc. ISBN 9798331314385.

Yujia Wang, Lu Lin, and Jinghui Chen. Communication-efficient adaptive federated learning. In *International conference on machine learning*, pp. 22802–22838. PMLR, 2022.

Yujia Wang, Yuanpu Cao, Jingcheng Wu, Ruoyu Chen, and Jinghui Chen. Tackling the data heterogeneity in asynchronous federated learning with cached update calibration. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 16160–16192, 2024b.

Yujia Wang, Shiqiang Wang, Songtao Lu, and Jinghui Chen. Fadas: Towards federated adaptive asynchronous optimization. *arXiv preprint arXiv:2407.18365*, 2024c.

Wenjing Yan, Xiangyu Zhong, Xiaolu Wang, and Ying-Jun Angela Zhang. Momentum-driven adaptivity: Towards tuning-free asynchronous federated learning. In *Forty-second International Conference on Machine Learning*, 2025.

Jiarong Yang and Yuan Liu. Gas: Generative activation-aided asynchronous split federated learning. volume 39, pp. 21956 – 21964, Philadelphia, PA, United states, 2025.

Xingyi Yang, Daquan Zhou, Songhua Liu, Jingwen Ye, and Xinchao Wang. Deep model reassembly. *Advances in neural information processing systems*, 35:25739–25753, 2022.

Xuefei Yin, Yanming Zhu, and Jiankun Hu. A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)*, 54(6): 1–36, 2021.

Tao Yu, Congzheng Song, Jianyu Wang, and Mona Chitnis. Momentum approximation in asynchronous private federated learning. In *International Workshop on Federated Foundation Models in Conjunction with NeurIPS 2024*, 2024. URL `https://neurips.cc/virtual/2024/100387`.

Yu Zang, Zhe Xue, Shilong Ou, Lingyang Chu, Junping Du, and Yunfei Long. Efficient asynchronous federated learning with prospective momentum aggregation and fine-grained correction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 16642–16650, 2024.

A Appendix

A Datasets and Networks.

.

For each dataset, we randomly reserve 10% of the training data as shared proxy data and partition the remainder among clients using a Dirichlet distribution with concentration parameter $\alpha \in \{0.6, 0.3, 0.1\}$ Hsu et al. (2019). All methods use the same client data partitions. The reserved subset is also excluded from the client training sets of baselines that do not use proxy data.

B Implementation Details and Hyperparameters

We conduct experiments using PyTorch 2.7.0 on NVIDIA RTX 5090 GPUs. Unless otherwise specified, we use $N = 100$ clients, with 10 clients training concurrently and a buffer threshold of $K = 10$. Training runs for 500 server updates with a default learning rate of 0.01. For our method, the one-time client warm-up lasts 15 epochs. At each aggregation step t with $S_t \neq \varnothing$, we construct $M_t = 10|S_t|$ candidates and calibrate each for 10 epochs. Method-specific baseline hyperparameters follow the configurations reported in the original papers. We report the mean and standard deviation over three independent runs.

C Sensitivity Analysis

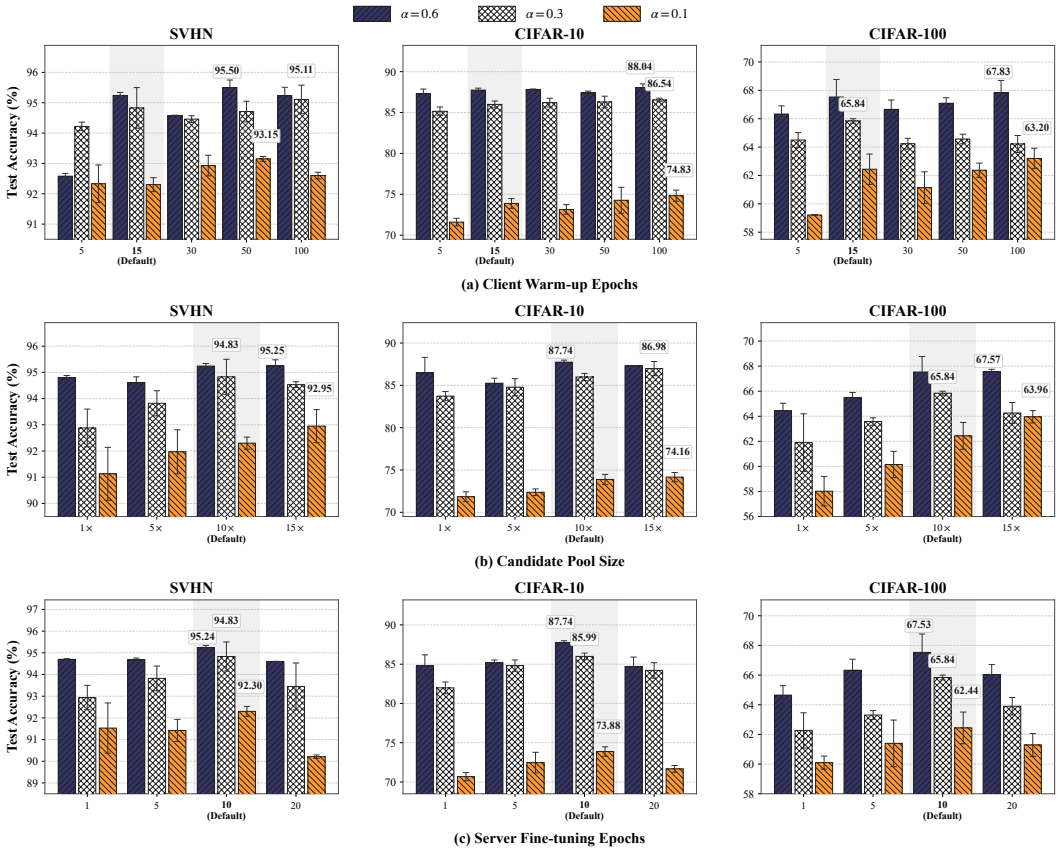


Figure 4: Sensitivity to the principal hyperparameters of GUA under the real-world straggler setting. Panels (a), (b), and (c) evaluate the client warm-up epochs, candidate pool size, and server-side fine-tuning epochs, respectively. Gray regions highlight the default settings, and the annotated values indicate the highest mean accuracy for each Non-IID level.

Table 4 provides the numerical results for the ablation analysis.

D Ablation Results

Table 4 in Appendix B compares five ablation variants with the full framework under the simulator-based asynchronous setting.

Table 3: Backbone configurations used in the architecture sensitivity analysis.

Backbone	**Topology (Primary Unit)**	**Nominal Depth**	**Reassembly Units**
CNN	Plain (Conv/FC)	9	9
VGG-11	Plain (Conv/FC)	11	11
VGG-16	Plain (Conv/FC)	16	16
ResNet-18	Residual (BasicBlock)	18	10
ResNet-34	Residual (BasicBlock)	34	18

Table 4: Ablation of the search, generation, and matching components under the simulator-based asynchronous setting. Results are test accuracy (%), reported as mean and standard deviation. Bold values indicate the highest mean in each column.

Stage	**Ablation Variant**	**CIFAR-10**		**CIFAR-100**	
		$\alpha = 0.6$	$\alpha = 0.1$	$\alpha = 0.6$	$\alpha = 0.1$
Search	Random anchor selection	86.33 ± 0.86	68.49 ± 0.42	64.37 ± 0.82	57.19 ± 0.65
	Without block borrowing	85.82 ± 1.23	68.44 ± 1.82	65.59 ± 1.06	54.78 ± 0.60
	Random block search	84.11 ± 0.35	66.73 ± 0.91	64.70 ± 0.36	57.59 ± 1.40
Generation	Few-source reconstruction (1–2 clients)	84.58 ± 1.09	71.06 ± 0.91	65.17 ± 0.61	57.98 ± 0.75
Matching	Random candidate assignment	82.83 ± 1.59	63.99 ± 2.48	64.58 ± 1.37	55.63 ± 0.28
Full framework		$\mathbf{87.74} \pm 0.24$	$\mathbf{73.88} \pm 0.60$	$\mathbf{67.53} \pm 1.24$	$\mathbf{62.44} \pm 1.07$