FEDJIGSAW: MULTI-AGENT COLLABORATIVE MODEL REASSEMBLY FOR DECENTRALIZED HETEROGENEOUS FEDERATED LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Model-Heterogeneous Federated Learning (MHFL) enables clients with diverse resource constraints to collaboratively train models with heterogeneous architectures. Existing approaches typically construct client models by extracting sub-models from a predefined global backbone. However, this top-down paradigm confines personalization to a prescribed architectural space. Meanwhile, other architecture-search-based methods impose significant computational and memory overhead to the server. We present `FedJigsaw`, a novel framework that reformulates model personalization as bottom-up model reassembly. Our key insight is that cohesive functional modules can serve as reusable units of both model structure and learned knowledge. Analogous to assembling a Jigsaw puzzle, each client dynamically constructs its personalized architecture by selecting compatible modules learned locally or obtained from its neighbors. We formulate the assembly process as a sequential decision problem, enabling each client to adapt its architecture to its resource constraints and learning context. To keep this process lightweight, `FedJigsaw` employs an attention-based pointer network to directly score a variable set of feasible modules, rather than enumerating the complete candidate architectures. Our experimental results demonstrate that, compared with the state-of-the-art MHFL schemes, `FedJigsaw` improves local accuracy by up to 8.56%, reduces per-client communication by 63.8%, and shortens round time by 17.3%, demonstrating the effectiveness and scalability of module-level reassembly.

1 INTRODUCTION

Model-Heterogeneous Federated Learning (MHFL) enables clients with different computational and communication capabilities to collaboratively learn using models of various sizes. This capability is essential in edge environments, where a single global backbone model may exceed the memory, computation, or latency budget of resource-constrained devices (Lim et al., 2020). Although conventional Personalized FL approaches can adapt model parameters, representation or aggregation methods to heterogeneous local data distribution (Xu et al., 2023; Tastan et al., 2024; Li et al., 2021), such weight-space personalization does not directly address architecture-level resource constraints and heterogeneity issue. MHFL therefore is required to solve a coupled objective: each client should obtain a model suited to its local data distribution and environmental context information, while still benefiting from knowledge shared among resource-heterogeneous clients.

Existing MHFL approaches mainly follow three paradigms. **Partial-Training** methods extract client-specific submodels from a shared backbone, supporting different computational budgets while retaining a common parameter structure (Wen et al., 2022; Diao et al., 2021; Horvath et al., 2021; Alam et al., 2022; Chen et al., 2024; Wu et al., 2024). However, their architecture choices remain coupled with the topology of the predefined global model. **Knowledge-Distillation and mutual-learning** methods allow heterogeneous architectures to exchange predictions or representations, although knowledge is communicated indirectly and additional models or reference data may be required (Lin et al., 2020; Li & Wang, 2019; Yi et al., 2024; Xie et al., 2024; Huang et al., 2024). **Federated architecture search** methods provide better structural flexibility, but generally search within centrally specified spaces or repeatedly construct and evaluate candidate models (He et al., 2020; Wang et al., 2023a; Yao & Li, 2024; Liao et al., 2025). These studies motivate an alternative

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.

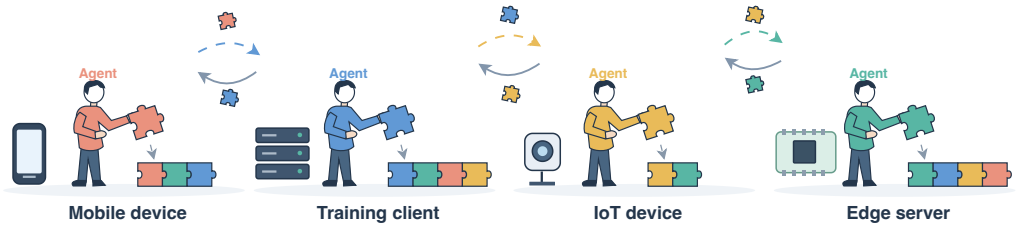


Figure 1: Agent-driven collaborative model reassembly across heterogeneous clients.

design in which clients sequentially construct models from modules distributed across the federation, without requiring an unified global supernet on a centralized server.

Motivations Our central research questions is: *Can a client construct a context-specific model by reusing functional modules (e.g., convolution block, encoder/decoder block) learned by itself and its communication neighbours?* Here, the client context includes its local data distribution, resource constraint, partially assembled model, and the functional modules available in its neighbourhood. As illustrated in Fig. 1, each heterogeneous client is associated with an assembly agent that constructs a personalized model by selecting modules from its local and neighbors. Context-specific assembly therefore does not imply unrestricted model generation, but identifies a suitable composition from compatible modules available within the client's communication neighborhood.

Functional modules are useful assembly units because they jointly carry structural and parametric information. Their *structural information*, including its type, input/output interface, and relative position in its source model, supports compatibility checking and describes its functional role. This information helps determine where the module can be placed, while its learned weights carry *parametric information* preserving transformation acquired from the source client's data. Reusing complete modules can therefore transfer both functional structure and learned knowledge.

Motivated by this observation, we propose `FedJigsaw`, a novel framework that formulates personalized model construction as sequential, resource-constrained module reassembly. Clients share information about their available modules with their neighbours. Each client then uses an attention-based mechanism to select compatible modules according to the local assembly context. Only the parameters of the selected module are retrieved, and each client further returns the trained versions to the neighbors from which the modules were retrieved. This design supports personalized architectures and cross-client knowledge sharing without requiring a shared global supernet.

`FedJigsaw` requires neither centralized aggregation nor centralized training. Each assembly policy can be trained locally with local information. However, concurrent client policy updates may introduce non-stationary. We therefore provide an *optional* training coordination enhancement that uses compact cross-client summaries, rather than raw data, to provide a more informative global signal. Clients may omit this stage when even such summaries are considered privacy-sensitive, while model assembly and execution remain fully decentralized.

Contributions Our main contributions are summarized as follows:

- **Distributed model-reassembly formulation** We formulate MHFL personalization as a bottom-up construction procedure from compatible functional modules distributed across clients. This formulation avoids requiring every client architecture to be extracted from a single centralized global model, while remaining explicit about the repository based assembly space.
- **Collaborative Architecture for Personalized Model Reassembly** We develop `AttenAssemble` to sequentially select modules according to local context and capacity constraints, and `SymbioArchitect` to support metadata discovery, module sharing and feedback. An optional `CoRe-Tune` mechanism further coordinates policy training using cross-client information while additional information sharing is acceptable. Together, they allow structurally different client models to collaborate through reusable modules.
- **Empirical Validation** Experiments evaluate `FedJigsaw` under heterogeneous data and different environmental settings. Among the evaluated decentralized baselines, `FedJigsaw` achieves

the highest local accuracy in six of the eight settings, while additional analyses characterize its generalization, module assembly behavior, and system costs. Meanwhile, on text classification, it further reduces communication up to 63.8% and round time up to 17.3%.

2 LITERATURE REVIEW

Knowledge Distillation and Mutual Learning in MHFL Knowledge Distillation and Mutual Learning transfer predictions or representations between heterogeneous models. FedDF (Lin et al., 2020) and FedMD (Li & Wang, 2019) use reference data to align client predictions, while pFedKnow (Wang et al., 2023b) and FedMRL (Yi et al., 2024) employ shared model for representation transfer. These methods avoid direct parameter compatibility but share knowledge indirectly and may require auxiliary data or models. In contrast, `FedJigsaw` directly reuses learned modules.

Partial Training in MHFL Partial-training methods extracts resource-aware subnetworks from a shared global model. HeteroFL (Diao et al., 2021), FjORD (Horvath et al., 2021), FlexFL (Chen et al., 2024) use predefined capability levels, while FedRolex (Alam et al., 2022) rotates subnetworks to improve parameter coverage. FIARSE (Wu et al., 2024) further adapts submodel extraction to client importance. Although these approaches are efficient, these methods remain constrained by a common backbone, and extracting submodels may not preserve complete functional structures.

Federated Neural Architecture Search Federated NAS introduces greater architectural flexibility by jointly optimizing model structures and parameters. FedNAS (He et al., 2020) searches within a shared supernet, pFedHR (Wang et al., 2023a) constructs heterogeneous model through layer matching, and PerFedRLNAS (Yao & Li, 2024) and Peaches (Yan et al., 2024) employ policy-based architecture decisions. However, maintaing supernets or repeatedly constructing and evaluating model candidates can introduce high computational overhead. `FedJigsaw` instead performs lightweight selection over reusable modules discovered from neighboring clients.

3 METHODOLOGY

3.1 PROBLEM FORMULATION: COLLABORATIVE MODEL REASSEMBLY

We consider a federation of heterogeneous clients, each with a local dataset and a device-specific resource budget. Our objective is to enable each client to construct a personalized model by assembling reusable `structural modules`, while adapting to its local data and resource constraints.

Structural Modules Each client i is initialized with a base model, which is manually partitioned into M_i structural modules based on their functional design. For example, for CNN and VGG models, each convolution block can be grouped with its normalization, activation and pooling operation when applicable, while fully connected layers remain separate. For transformer based models, each complete encoder/decoder block forms a module. These modules preserve their internal structure and serve as reusable jigsaw pieces carrying structural and learned knowledge.

Together, these modules form client i's local repository $\mathcal{R}_i^k = \{(\theta_{i,j}^k, m_{i,j})\}_{j=1}^{M_i}$ at communication round k, where $\theta_{i,j}^k$ denotes the module parameters, and $m_{i,j} = \langle i, j, type_{i,j}, d_{i,j}^{\text{in}}, d_{i,j}^{\text{out}} \rangle$ is the module metadata recording the source client, relative position, module type, and input/output shapes. This repository provides components for model assembly and identities for subsequent parameter updates.

Model Reassembly The local repositories collectively define the module space $\mathcal{M}^k = \bigcup_{i \in \mathcal{N}} \mathcal{R}_i^k$, where $\mathcal{N}$ is the set of clients. This space is distributed across the federation rather than maintained as a global repository in a specific server. Given its communication neighbors $\mathcal{N}_i$, client i can access the candidate pool $\mathcal{C}_i^k = \bigcup_{u \in \{i\} \cup \mathcal{N}_i} \mathcal{R}_u^k \subseteq \mathcal{M}^k$. At this stage, access refers to observing the module metadata, while parameters are retrieved only after the selection.

Starting with the context, client i sequentially selects modules from $\mathcal{C}_i^k$ to construct its model f_i^k:

$$f_i^k = g_{i,T_i}^k \circ \cdots \circ g_{i,1}^k, \tag{1}$$

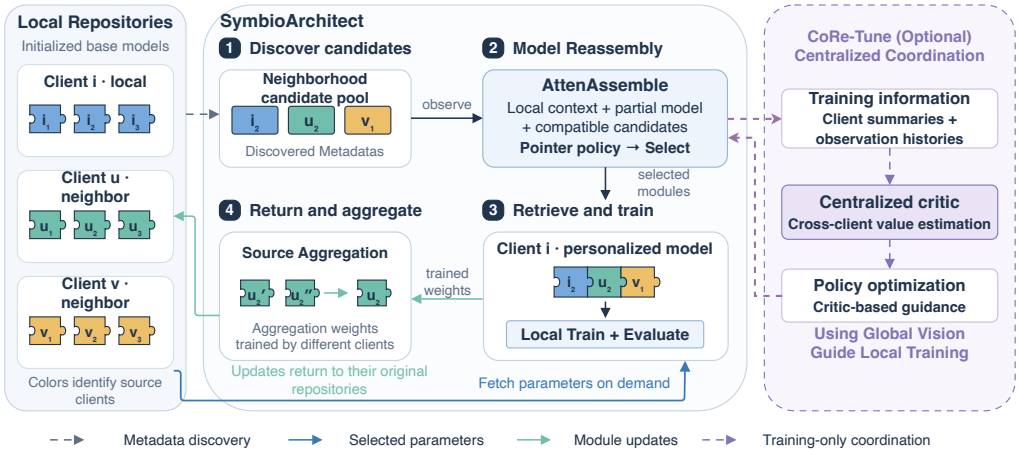


Figure 2: Overview of the `FedJigsaw` framework.

where $g_{i,t}^k$ denotes the function implemented by the module selected at step t, and T_i is the number of assembled modules. A valid assembly requires compatible shapes between consecutive modules and a final output matching the local task. The optimization objective is to find a feasible composition that achieves high predictive performance after local training considering resource constraints.

Sequential Decision Making Model reassembly is a sequential decision-making process in which a client interacts with an evolving assembly environment. At each step, clients select a module based on its local context and the partially assembled model. Appending the selected model changes the assembly model and the feasible choices for the next step. Once assembly is complete, local training and evaluation provides feedback on the resulting composition. By this mean, this interaction makes **Reinforcement Learning** a natural fit for learning assembly policies. However, clients face different and changing candidate pools, while parameter updates to shared modules can affect the outcomes of other's assembly decisions. With only local and neighborhood information, each client learns in a **Partially Observable, Non-stationary Environment** shaped by other clients' evolving policies and module updates. These interdependencies motivate `FedJigsaw`, our federated learning framework for collaborative model reassembly.

3.2 The FedJigsaw Framework

In order to train and coordinate the model reassembly policy, `FedJigsaw` contains three key components, as illustrated in Fig. 2. `AttenAssemble` selects compatible modules through a context-aware policy, `SymbioArchitect` supports module sharing and parameter udpates, and `CoRe-Tune` optionally coordinates policy learning through centralized training with decentralized execution (CTDE). Together, they connect local model assembly with collaborative module reuse

3.2.1 AttenAssemble

`AttenAssemble` learns a context-aware policy for sequential module selection. At assembly step t of round k, client i observes its local data representation, the metadata of previously selected modules, and its neighborhood candidate pool as the context. We denote this observation as $o_{i,t}^k$. The valid action set $\mathcal{A}_{i,t}^k$ contains selections of modules whose input shapes match the current intermediate output, together with a `STOP` action when termination is admissible. Selecting a module appends it to the partial architecture, whereas `STOP` completes assembly.

Since the valid action set $\mathcal{A}_{i,t}^k$ varies across clients and assembly steps, the policy scores the current candidates rather than using a fixed-dimensional action head. To handle the varying amount of actions, inspired by pointer network (Vinyals et al., 2015), we jointly encode the local data representation, current assembled partial architecture, and candidate action metadata using a Transformer encoder. The contextual representation of the last selected module in the partial architecture serves as the query

$q_{i,t}^k$, with the data token used before any module is selected. Each valid action a is represented by a candidate embedding $z_{i,t,a}^k$, including a dedicated embedding for `STOP` token. With these definitions, the policy is represented as:

$$\pi_{\phi_i}(a \mid o_{i,t}^k) = \frac{\exp\big(s_{\phi_i}(q_{i,t}^k, z_{i,t,a}^k)/\tau\big)}{\sum_{b \in \mathcal{A}_{i,t}^k} \exp\Big(s_{\phi_i}(q_{i,t}^k, z_{i,t,b}^k)/\tau\Big)}, \quad a \in \mathcal{A}_{i,t}^k, \tag{2}$$

where s_{ϕ_i} represents the learned attention scoring function, ϕ_i denotes the policy parameters, and $\tau > 0$ controls the sampling temperature. The policy then produces a probability distribution over the remaining valid actions, accommodating a variable number of available actions.

After the model reassembly decision is made, the selected module parameters are then used to instantiate the model for local training and evaluation. The evaluation results contribute to a terminal reward that guides policy training across the sequence of assembly decisions. The Reinforcement Learning training algorithm and the detailed architectures are provided in the Appendix A.1.

3.2.2 SYMBIOARCHITECT

While `AttenAssemble` works as the policy to assemble a local personalized model, `SymbioArchitect` connects the local model assembly with the collaborative module sharing, including metadata discovery, parameter retrieval, and source-based parameter aggregation.

Module Discovery and Transfer Each client broadcasts the its local repository metadata to its communication neighbors, allowing neighbors to know the module information without detailed parameters. Then `AttenAssemble` selects compatible modules and records their source identities. After the model assemble decision is made, the client then retrieves the parameters of selected remote modules and trains the assembled model with its local data. The source repositories retain their module definitions across rounds, including the unselected modules for future usage.

Source-Based Aggregation After the local training, each client returns the updated module parameters to their respective sources. For module j retrieved from client u, let $\mathcal{B}_{u,j}^k$ denote the set of clients that trained a copy of this module in round k. The source updates its repository entry by averaging these trained copies, which is defined as:

$$\theta_{u,j}^{k+1} = \frac{1}{|\mathcal{B}_{u,j}^k|} \sum_{i \in \mathcal{B}_{u,j}^k} \widetilde{\theta}_{u,j}^{k,i}, \qquad |\mathcal{B}_{u,j}^k| > 0, \tag{3}$$

where $\widetilde{\theta}_{u,j}^{k,i}$ represents the parameters of module (u, j) after training within client i's assembled model in round k. If modules receiving no updates, then retain their previous parameters. Only copes of the same source modules are aggregated, allowing clients with different architectures to jointly update shared modules without sharing their entire models.

3.2.3 CORE-TUNE

As clients concurrently update their assembly policies, from each client's perspective, the environment becomes non-stationary since the feedback on its assembly decisions depends on other client's evolving policies. However, each policy can be trained with local information without sharing addtional privacy-sensitive representations. We therefore introduce `CoRe-Tune`as an *optional* CTDE-based training stage, using cross-client summaries derived from task data rather than rwae samples to guide policy training. This *optional* critic is used only during the training stage and is *not required* for decentralized model assembly.

Centralized Policy Training During policy training, a centralized critic receives client summaries and local observation histories. Let $h_{i,t}^k$ denote client i's observation history up to assembly step t, and let Z^k denote an attention-based representation of the collected cross-client information for round k. The critic $V_\psi(h_{i,t}^k, Z^k)$ estimates the expected return using both the client's local history

and the broader training context. In contrast, the actor $\pi_{\phi_i}(a \mid o_{i,t}^k)$ selects modules using only locally available observations. Intuitively, the centralized critic uses cross-client information to provide a more informed training signal, helping each actor consider for interactions with other clients while making decisions from the local observation alone.

Policy Optimization Let $H_i = T_i + 1$ denote the number of assembly decisions, including the final `STOP` action. After the assembled model is trained and evaluated, client i receives a terminal reward R_i^k. As the intermediate assembly steps receive zero reward, giving the discounted return $G_{i,t}^k = \gamma^{H_i - t} R_i^k$, where γ is the discount factor. The critic provides a baseline for the policy update, which is defined as:

$$\nabla_{\phi_i} J_i = \mathbb{E}\left[\sum_{t=1}^{H_i} \nabla_{\phi_i} \log \pi_{\phi_i}(a_{i,t}^k \mid o_{i,t}^k)\left(G_{i,t}^k - V_\psi(h_{i,t}^k, Z^k)\right)\right] \tag{4}$$

where J_i denotes client i's expected terminal reward under the current joint policy. The centralized critic learns to predict assembly returns and estimates advantage that guide clients' policy updates.

Decentralized Execution Once the policy training is completed, each clients executes its assembly policy using local data representations, the assembled model's partial architecture, and neighbourhood repository metadata. The centralized critic is no longer required for action selection, while neighbourhood module sharing continues through `SymbioArchitect`.

4 Experiments

4.1 Experiment Setup

We evaluate `FedJigsaw` on CIFAR-10, CIFAR-100, EMNIST, and SVHN for image classification, and AG News for text classification, using lightweight CNNs, VGG variants, and RoBERTa-base, respectively. We compare against the centralized MHFL baselines FedRolex (Alam et al., 2022), FIARSE (Wu et al., 2024), FedMRL (Yi et al., 2024), PerFedRLNAS (Yao & Li, 2024), pFedHR Wang et al. (2023a), as well as the decentralized baselines ProxyFL (Kalra et al., 2023) and DeSA (Huang et al., 2024). Data heterogeneity is generated using Dirichlet partitions with $\alpha = 0.3$ and $\alpha = 1.0$, while resource heterogeneity is represented by four client-capacity tiers. These tiers determine the maximum assembly depth in `FedJigsaw`, and baseline configurations are matched according to their profiled FLOPs. Detailed architectures, capacity mappings, training settings, and implementation environments are provided in Appendix A.4.1.

4.2 Overall Performance

We evaluate the mean local test accuracy across 20 clients, with five clients assigned to each capacity tier. `FedJigsaw`, ProxyFL, and DeSA using a 20-client topology with 5 neighbors each, whereas the centralized baselines communicate through a server. Additional results in Appendix A.4.4 show that `FedJigsaw` is robust to neighborhood density. All methods are trained up to 1,000 communication rounds, with early stopping after 200 rounds without improvement. The assembly policies of `FedJigsaw` are optimized for the first 500 rounds. Tab. 1 report the personalized model local accuracy, which reflects model personalization, our primary goal.

Comparison with decentralized methods Among the decentralized baseline, `FedJigsaw` achieves the best local accuracy in six of eight image classifications. Its advantage are particularly evident on CIFAR-100, EMNIST, and SVHN, where it improves the strongest decentralized baseline by 2.13% - 8.56%. On CIFAR-10, ProxyFL obtains marginally higher local accuracy, exceeding `FedJigsaw` by only 0.53% and 0.52% higher under both data distributions, whereas FedJigsaw performs better on six of the remaining eight image settings. These results are consistent with their knowledge-sharing mechanisms. ProxyFL compresses knowledge into a proxy model, while DeSA communicates through synthetic anchors, both transferring knowledge indirectly. In contrast, `FedJigsaw` directly reuses completed learned modules and returns the updated parameters, allowing both structural and parametric knowledge to transfer across clients.

Method	CIFAR-10		CIFAR-100		EMNIST		SVHN		AG News	
Backbone	*CNN*		*VGG*		*CNN*		*CNN*		*RoBERTa-base*	
	Low	High	Low	High	Low	High	Low	High	Low	High
Server-Coordinated MHFL										
FedRolex	73.85±6.21	58.30±14.17	**47.69±3.86**	**45.00±4.59**	85.28±5.91	78.74±11.48	90.27±2.78	91.46±4.03	31.70±24.66	36.91±31.53
FIARSE	**77.66±4.42**	75.18±9.93	38.93±3.28	41.09±3.16	**87.75±4.68**	**90.62±4.93**	**92.11±1.83**	**93.21±1.94**	27.93±24.57	39.49±31.83
FedMRL	60.10±5.35	**78.58±12.38**	30.86±4.29	43.93±5.05	82.78±7.81	87.73±8.62	75.33±20.45	82.17±16.88	**62.08±17.29**	**71.85±17.94**
PerFedRLNAS	69.10±5.50	77.65±13.15	26.50±4.84	39.10±6.74	85.77±5.08	90.29±3.85	86.34±3.99	92.94±4.64	–	–
pFedHR	62.79±6.25	76.62±12.31	OOM	OOM	OOM	OOM	65.24±22.24	79.38±18.66	–	–
Peer-to-Peer Heterogeneous FL										
ProxyFL	**70.23±5.52**	**81.49±9.29**	23.32±4.17	37.98±6.09	84.04±9.04	89.34±6.95	77.54±17.42	83.68±12.70	–	–
DeSA	68.80±5.72	80.88±9.90	15.68±3.77	30.42±6.23	84.83±8.35	90.41±6.29	76.73±17.70	84.74±12.94	–	–
FedJigsaw (Ours)	69.70±5.98	80.97±7.61	**31.16±2.60**	**41.96±5.97**	**88.15±4.49**	**92.54±3.98**	**86.10±4.45**	**90.35±6.18**	**94.44±3.39**	**96.20±2.73**

Table 1: Mean client-level accuracy (%) and population standard deviation across clients under low ($\alpha = 1.0$) and high ($\alpha = 0.3$) data heterogeneity.

Dataset	Variant	Local Acc. ↑	Global Acc. ↑	Client Std. ↓	FLOPs ↓
CIFAR-10	Full	63.05	**20.50**	9.74	**30.7M**
	w/o CoRe-Tune	**65.00**	20.05	11.40	72.9M
	Random Assembly	62.44	17.52	9.66	77.7M
	w/o Symbiotic Feedback	62.45	17.82	**9.58**	91.8M
EMNIST	Full	71.85	30.62	10.48	83.7M
	w/o CoRe-Tune	**72.61**	**30.71**	10.75	121.4M
	Random Assembly	71.72	29.39	**10.47**	116.5M
	w/o Symbiotic Feedback	67.80	28.43	10.80	**61.4M**
SVHN	Full	**54.60**	**18.18**	11.70	**44.7M**
	w/o CoRe-Tune	42.93	13.04	7.57	153.0M
	Random Assembly	43.28	13.38	7.52	82.4M
	w/o Symbiotic Feedback	42.84	14.63	**6.50**	105.9M

Table 2: Ablation results under high data heterogeneity ($\alpha = 0.3$). Client Std. measures the population standard deviation of local accuracy across clients.

Comparison with centralized methods Centralized methods can use server aggregation to integrate client information globally and consequently remain strong on several CNN-based tasks. Particularly, partial-training methods perform well on CIFAR-10 and SVHN. However, this advantage does not uniformly extend across tasks and model families. On EMNIST, `FedJigsaw` remains competitive with or outperforms the centralized baselines, and the difference becomes significant on AG News. Although pFedHR obtains competitive results on several CNN settings, `FedJigsaw` outperforms it across the completed comparisons. Moreover, pFedHR incurs substantially higher architecture-search overhead and encounters OOM under the EMNIST and CIFAR-100 setting. With RoBERTa-base, `FedJigsaw` achieves 94.44% and 96.20% local accuracy, compared with at most 62.08% and 71.85% among the centralized baselines. Centralized aggregation enables these methods to integrate information across clients, which often benefits learning under Non-IID distributions. However, submodel extraction can disrupt structural and parametric dependencies, while representation-level mutual learning transfers knowledge indirectly without reusing the learned transformations themselves. In contrast, `FedJigsaw` preserves the functional integrity of reusable modules, enabling more effective knowledge transfer in structurally complex models and leading to stronger performance.

4.3 Ablation Study

All ablation experiments use eight clients and run for 300 communication rounds with $\alpha = 0.3$ on CIFAR-10, EMNIST, and SVHN. We replace `AttenAssemble` with random valid assembly, disabling the source-side feedback in `SymbioArchitect`, and removing the centralized critic in `CoRe-Tune`. Tab. 2 reports the model performance, cross-client variation, and the computational overhead. Their accuracy FLOPs trade-offs is further visualized in Fig. 6 in Appendix A.4.3.

Effectiveness of AttenAssemble To test `AttenAssemble`, we replace it with random selection among valid modules while retaining the same candidate pool and capacity constraints. As shown in Tab. 2, random assembly consistently produces more computationally expensive models without improving local accuracy. The difference is mostly significant on SVHN, where local accuracy

decreases from 54.60% to 43.28%, while the average inference cost increases from 44.7M to 82.4M FLOPs. These results show that the improvement does not contributed by arbitrary module composition or simply assembly larger models. Instead, `AttenAssemble` learns more effective compositions under the available resource budget.

Effectiveness of SymbioArchitect To test `SymbioArchitect`, we disable the return and source-side aggregation of selected remote modules, while preserving module discovery and model assembly. Removing this feedback generally degrades both the local and global accuracy, particularly on EMNIST and SVHN. On EMNIST, local and global accuracy decrease from 71.85% to 67.80% and from 30.62% to 28.43%, separately. This degradation is more significant on SVHN, where the corresponding degradation is 11.76% and 3.55%. This result indicates that merely borrowing the structure of neighboring modules is insufficient. Returning locally trained modules to their source repositories is essential for knowledge to accumulate and propagate across the federation.

Effectiveness of CoRe-Tune We ablation `CoRe-Tune` by removing the centralized critic and training each assembly policy independently. This variant remains competitive, confirming that `CoRe-Tune` is optional, but generally produces less efficient or consistent assemblies. On CIFAR-10, it improves local accuracy by 1.95%, but increases FLOPs from 30.7M to 72.9M and the cross-client standard deviation from 9.74 to 11.40. On EMNIST, a 0.76% improvement requires 45% more FLOPs. On SVHN, removing `CoRe-Tune` reduces both local and global accuracy while increasing FLOPs from 44.7M to 153.0M. These results show that independent policy training is viable, while `CoRe-Tune` can improve assembly efficiency and robustness when cross-client coordination is beneficial.

4.4 Adapting Beyond the Initial Model Size

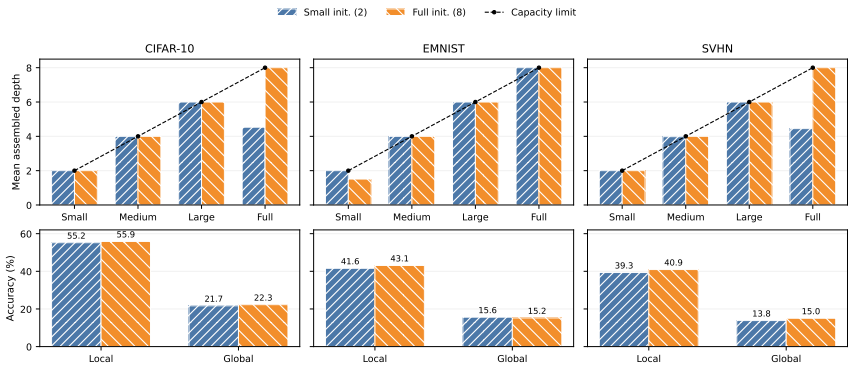


Figure 3: Top: Mean assembled depth over the final 50 rounds under uniform Small or Full initialization, where dashed lines denote capacity limits. Bottom: final local and global accuracy

As `FedJigsaw` requires a base model for initialization, we examine whether the assembled model is constrained by its initial size. We initialize all eight clients uniformly with either the smallest or biggest model, while the clients are uniformly assigned to each capacity tier. Experiments are conducted for 200 communications rounds with $\alpha = 0.5$ on CIFAR-10, EMNIST, and SVHN.

As shown in Fig. 3, Small-initialized clients can assemble substantially deeper models, whereas Full-initialized clients can construct shallower models under budget limits. The resulting depth primarily follows the capacity limit rather than the initial model size. The two initializations also achieve comparable performance, with maximum differences of only 1.59% and 1.22% in local and global accuracy. Moreover, 72.7% - 98.2% of selected modules originate from other clients. These results demonstrate that `FedJigsaw` adapts architectures through cross-client module reuse instead of inheriting the initial architecture. Results on RoBERTa-base in Appendix A.4.2 further confirm this behavior across model families.

Dataset	Method	Model	Model FLOPs (G)	Adapt./Search (s)	Round Time (s)	Peak Mem. (GiB)	Comm. (MB)
CIFAR-10	FedRolex	CNN	**0.190**	**0.014**	0.879	2.848	11.755
	FIARSE	CNN	0.304	0.501	1.405	2.923	15.261
	FedMRL	CNN	0.231	–	**0.851**	**2.628**	**0.225**
	pFedHR	CNN	0.191	50.376	52.929	73.293	31.084
	ProxyFL	CNN	0.191	–	1.170	2.632	0.536
	DeSA	CNN	0.191	–	2.522	2.904	1.256
	FedJigsaw	CNN	0.191	0.082	1.353	2.949	0.252
CIFAR-100	FedRolex	VGG	0.583	0.274	4.138	7.403	219.915
	FIARSE	VGG	0.835	10.665	15.517	9.485	255.701
	FedMRL	VGG	0.853	–	2.329	4.494	74.321
	ProxyFL	VGG	0.585	–	**1.894**	**4.199**	**1.092**
	DeSA	VGG	0.585	–	13.919	6.188	5.194
	FedJigsaw	VGG	**0.547**	**0.151**	2.212	6.057	49.220
AG News	FedRolex	RoBERTa-base	**10.570**	0.771	13.995	16.040	516.463
	FIARSE	RoBERTa-base	22.348	16.934	33.516	25.080	868.983
	FedMRL	RoBERTa-base	19.555	–	10.489	**13.110**	482.769
	FedJigsaw	RoBERTa-base	13.503	**0.113**	**8.679**	13.920	**174.834**

Table 3: System-efficiency benchmark under a four-client setting. Peak memory denotes aggregate container memory, and communication is averaged per client per round.

4.5 System Efficiency

Tab. 3 reports the computational overhead, memory, and communication costs of different methods. We remove `PerFedRLNAS` because it uses DARTS which is a different backbone compare with other methods. On CIFAR-10, the additional policy and assembly operations introduce moderate overhead. FedMRL achieves the lowest round time, memory footprint, and communication cost by federating only a compact representation branch, while `FedJigsaw` maintains comparable model FLOPs and communication cost without exchanging complete models. The advantage of `FedJigsaw` becomes more evident as the backbone grows. On CIFAR-100, it achieves the lowest model FLOPs, adaptation latency, while reducing communication by 33.8% relative to FedMRL. On AG News, it reduces communication by 63.8% and round time by 17.3% relative to FedMRL, while retaining the lowest assembly latency. Overall, on-demand module retrieval scales more favorably than transferring full subnetworks or large shared representations, while compact proxy model and anchor-based methods achieve lower communication by compressing the transferred knowledge more aggressively.

5 Conclusion

In this work, we presented `FedJigsaw`, a decentralized MHFL framework that personalizes models through bottom-up reassembly of functional modules. Rather than extracting subnetworks from a predefined backbone, clients construct personalized architectures by reusing modules learned locally and by their neighbors. Context-aware assembly and module level feedback enable the transfer of both structural and parametric knowledge, while the optional `CoRe-Tune` coordinates policy learning when needed. `FedJigsaw` imrpoves local accuracy over decentralized baselines by up to 8.56 percentage points on image tasks. On text classification, it reduces per-client communication by 63.8% and round time by 17.3%, demonstrating favorable scalability to pretrained backbones. Future work will extend sequential reassembly to more flexible architectures and strengthen the privacy protection of cross-client information.

REFERENCES

Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Advances in neural information processing systems*, 35:29677–29690, 2022.

Zekai Chen, Chentao Jia, Ming Hu, Xiaofei Xie, Anran Li, and Mingsong Chen. Flexfl: Heterogeneous federated learning via apoz-guided flexible pruning in uncertain scenarios. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(11):4069–4080, 2024.

Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

Chaoyang He, Murali Annavaram, and Salman Avestimehr. Towards non-iid and invisible data with fednas: Federated deep learning via neural architecture search. *arXiv preprint arXiv:2004.08546*, 2020.

Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889, 2021.

Chun-Yin Huang, Kartik Srinivas, Xin Zhang, and Xiaoxiao Li. Overcoming data and model heterogeneities in decentralized federated learning via synthetic anchors. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Shivam Kalra, Junfeng Wen, Jesse C. Cresswell, Maksims Volkovs, and H. R. Tizhoosh. Decentralized federated learning through proxy model sharing. *Nature Communications*, 14(1):2899, May 2023. ISSN 2041-1723. doi: 10.1038/s41467-023-38569-4.

Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*, 2019.

Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10713–10722, 2021.

Peng Liao, Xilu Wang, Yaochu Jin, Wenli Du, and Han Hu. Neural architecture search driven by locally guided diffusion for personalized federated learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4222–4231, October 2025.

Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020. doi: 10.1109/COMST.2020.2986024.

Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in neural information processing systems*, 33:2351–2363, 2020.

Nurbek Tastan, Samar Fares, Toluwani Aremu, Samuel Horvath, and Karthik Nandakumar. Redefining contributions: Shapley-driven federated learning. In *33rd International Joint Conference on Artificial Intelligence, IJCAI 2024*, pp. 5009–5017. International Joint Conferences on Artificial Intelligence, 2024.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Jiaqi Wang, Xingyi Yang, Suhan Cui, Liwei Che, Lingjuan Lyu, Dongkuan DK Xu, and Fenglong Ma. Towards personalized federated learning via heterogeneous model reassembly. *Advances in Neural Information Processing Systems*, 36:29515–29531, 2023a.

Jiaqi Wang, Shenglai Zeng, Zewei Long, Yaqing Wang, Houping Xiao, and Fenglong Ma. Knowledge-enhanced semi-supervised federated learning for aggregating heterogeneous lightweight clients in iot. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*, pp. 496–504. SIAM, 2023b.

Dingzhu Wen, Ki-Jun Jeon, and Kaibin Huang. Federated dropout—a simple approach for enabling federated learning on resource constrained devices. *IEEE wireless communications letters*, 11(5): 923–927, 2022.

Feijie Wu, Xingchen Wang, Yaqing Wang, Tianci Liu, Lu Su, and Jing Gao. Fiarse: Model-heterogeneous federated learning via importance-aware submodel extraction. *Advances in Neural Information Processing Systems*, 37:115615–115651, 2024.

Luyuan Xie, Manqing Lin, Tianyu Luan, Cong Li, Yuejian Fang, Qingni Shen, and Zhonghai Wu. MH-pFLID: Model heterogeneous personalized federated learning via injection and distillation for medical data analysis. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54561–54575. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/xie24h.html.

Jian Xu, Xinyi Tong, and Shao-Lun Huang. Personalized federated learning with feature alignment and classifier collaboration. In *The Eleventh International Conference on Learning Representations*, 2023.

Jiaming Yan, Jianchun Liu, Hongli Xu, Zhiyuan Wang, and Chunming Qiao. Peaches: Personalized federated learning with neural architecture search in edge computing. *IEEE Transactions on Mobile Computing*, 23(11):10296–10312, 2024.

Dixi Yao and Baochun Li. Perfedrlnas: One-for-all personalized federated neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 16398–16406, 2024.

Liping Yi, Han Yu, Chao Ren, Gang Wang, Xiaoguang Liu, and Xiaoxiao Li. Federated model heterogeneous matryoshka representation learning. *Advances in Neural Information Processing Systems*, 37:66431–66454, 2024.

A TECHNICAL APPENDICES AND SUPPLEMENTARY MATERIAL

A.1 ATTENASSEMBLE ARCHITECTURE AND TRAINING

Fig. 4 demonstrates the architecture of `AttenAssemble`. At each assembly step, the policy evaluates the candidate modules conditioned on client's local data and partially assembled model.

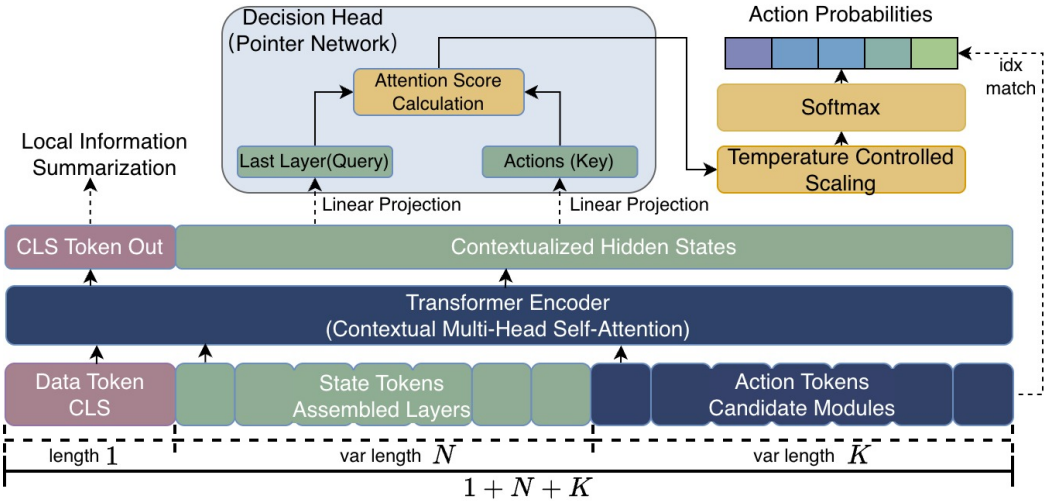


Figure 4: Architecture of `AttenAssemble`. A local data token, state tokens, and candidate action tokens are contextualized by a Transformer Encoder. A pointer head scores the candidate actions using the current assembly state as its query.

Token Construction The pointer network takes three types of tokens as input: a data token, token representing the current assembly state, and candidate-action tokens. To construct the data token, each client encodes a batch of local samples and averages their representations. The current assembly state is represented by the ordered sequence of previously selected modules, where each module is encoded from its metadata $m_{i,j}$ together with a positional embedding indicating its position in the assembled model. Each currently feasible candidate module is similarly encoded as an action token. Thus, at assembly step t, the Transformer input consists of the data token, the ordered tokens of the $t-1$ selected modules, and one token for each action in $\mathcal{A}_{i,t}$, including a dedicated `STOP` token.

Pointer Network-based Scoring As shown in Fig. 4, after all three types of token are concatenated as a sequence, the Transformer Encoder will jointly contextualizes these tokens. Its output for the most recently selected modules will serve as the query, because we want to know which candidate layer match the last layer best while considering all the currently assembled layers and the data distributions. The outputs corresponding to the candidate actions form the keys. Concretely, the scoring function s_{ϕ_i} in Eq. 2 is implemented as:

$$s_{\phi_i}(q^k_{i,t}, z^k_{i,t,a}) = \frac{(W_Q q^k_{i,t})^\top (W_K z^k_{i,t,a})}{\lambda\sqrt{d}}, \tag{5}$$

where d stands for the embedding dimension, W_Q and W_K are learned projections, and $\lambda > 0$ is a learned logit scale. The sampling temperature τ in Eq. 2 further controls exploration during the policy training.

Action Constraints and execution Candidate modules are filtered according to their input/output compatibility, prior selections in the current assembly, and the client's capacity limitation. The `STOP` action is masked until at least one module is selected. During the policy training, the client samples

an action from the resulting probability distribution. During the decentralized execution, the action will be chosen from the highest probability. Assembly terminates when client selects `STOP` token or reach its capacity limit. The client will then retrieve the selected modules and appends the model with a task-specific classification head when necessary. The terminal evaluation feedback is used to train the policy as detailed in the following subsection.

A.2 CoRe-Tune and Reward Design

Centralized Critic `CoRe-Tune` is an optional training component. The centralized critic described below is used only when it is enabled. Otherwise, each actor is optimized directly from its observed Monte Carlo returns. During the policy training, each participant sends the centralized server its encoded data token and the state representation producing its assembly history. Throughout this procedure, no raw data is being transmitted. Let $c_{i,t}^k$ denote client i's state representation after its t-th assembly decision. The critic first encodes the assembly history to obtain the local history representation $h_{i,t}^k$. Then, it collects the same representations of other participants:

$$Z_{i,t}^k = \left\{ c_{u,\min(t,H_u)}^k \mid u \in \mathcal{P}^k \setminus \{i\} \right\}, \tag{6}$$

where $\mathcal{P}^k$ is the set of the clients participating in round k, and H_u is the trajectory length of client u. The final available state is used when another client's trajectory is shorter than t, as they have different capacity limits.

The critic also employs an attention mechanism to contextualize the local assembly trajectory $h_{i,t}^k$ with cross-client context $Z_{i,t}^k$, and estimates the expected return as $V_\psi(h_{i,t}^k, Z_{i,t}^k)$. As each client only observes a partial view of the global Federated Learning system status, similar local observations may corresponding to different global conditions. The centralized critic uses the joint context to provide a better-conditioned training signal, guiding the actor toward decisions that perform well across the global contexts associated with its local observation. During decentralized execution, the critic and joint context are no longer required, and each clients selects module only based on their local observation.

Reward Design Although intermediate assemblies could in principle be evaluated, repeatedly training and evaluating them would introduce additional overhead on resource constrained devices. Therefore, we provide a terminal reward only after the completed model has been locally trained and evaluated on the client's validation set.

Let Acc_i^k denote the resulting validation accuracy of client i in the round k. To encourage clients with similar data contexts to explore diverse model compositions, we define the cross-client exploration score as Eq. 7, where $\mathcal{P}^k$ is the set of the participating clients in this round, d_i^k denotes the data token, c_i^k is the finally assembly-state representation, and $\mathrm{sim}(\cdot,\cdot)$. Intuitively speaking, if there are two clients with a quite similar data distribution, a higher reward will be granted if they come up with different structures.

$$E_i^k = \frac{1}{|\mathcal{P}^k| - 1} \sum_{\substack{j \in \mathcal{P}^k \\ j \neq i}} \mathrm{sim}(d_i^k, d_j^k) \left[1 - \mathrm{sim}(c_i^k, c_j^k) \right], \tag{7}$$

Combining with the model performance, the terminal reward is then defined as Eq. 8, where $\lambda_E \in [0, 1]$ balances predictive performance and cross-client exploration.

$$r_i^k = \begin{cases} \lambda_E \, \mathrm{Acc}_i^k + (1 - \lambda_E) \, E_i^k, & \text{if } \texttt{CoRe-Tune} \text{ is enabled,} \\ \mathrm{Acc}_i^k, & \text{otherwise.} \end{cases} \tag{8}$$

For a trajectory containing T_i^k selections, the return assigned to step t is defined following the discounted Monte Carlo return with a terminal reward, defined as Eq. 9, where $\gamma \in (0, 1]$ is the discount factor, and the corresponding advantage is then computed as Eq. 10.

$$G_{i,t}^k = \gamma^{T_i^k - t} r_i^k, \tag{9}$$

$$\hat{A}_{i,t}^k = \begin{cases} G_{i,t}^k - V_\psi(h_{i,t}^k, Z_{i,t}^k), & \text{if } \texttt{CoRe-Tune} \text{ is enabled,} \\ G_{i,t}^k, & \text{otherwise.} \end{cases} \tag{10}$$

Thus, disabling `CoRe-Tune` removes both the cross-client exploration term and the centralized value baseline, allowing each actor to be optimized entirely from local feedback.

Entropy-Regularized Policy Optimization At this stage, the designed reward motivates the `AttenAssemble` to construct a high-performing model while exploring structurally diverse compositions. However, as the reward is provided after the model is completely assembled and trained, the delayed feedback may cause the policy to prematurely concentrate on a small subset of actions, particularly the `STOP` action, due to the noisy advantages during early training, influencing the chance of sample other module compositions. Thus, we introduce entropy regularization to maintain sufficient step-wise exploration during training stage.

The entropy of the action distribution at step t is defined as Eq. 11. In this definition, as the summation is taken over all the current valid actions. A higher entropy indicates that the policy distributes its probability mass across multiple candidate modules, whereas a low entropy indicates that it concentrates on a small number of actions. With this definition, maximizing the entropy discourages premature policy collapse and preserves a better opportunity to evaluate alternative model compositions in an early age.

$$\mathcal{H}_{i,t}^k = -\sum_{a \in \mathcal{A}_{i,t}^k} \pi_{\phi_i}(a \mid o_{i,t}^k) \log \pi_{\phi_i}(a \mid o_{i,t}^k). \tag{11}$$

Specifically, the entropy regularization is defined as Eq. 12, where $\lambda_{\mathcal{H}} \geq 0$ controls the strength of the exploration. The first term increases the probability of the actions associated with positive advantages and suppresses actions with negative advantages, while the second term prevents the policy from prematurely concentrating on a small subset of actions. Together, these terms balance the exploitation of the promising model composition and the explorations.

$$\mathcal{L}_{\text{actor},i}^k = -\frac{1}{T_i^k} \sum_{t=1}^{T_i^k} \left[\log \pi_{\phi_i}(a_{i,t}^k \mid o_{i,t}^k) \hat{A}_{i,t}^k + \lambda_{\mathcal{H}} \mathcal{H}\big(\pi_{\phi_i}(\cdot \mid o_{i,t}^k)\big) \right], \tag{12}$$

Combining the terminal reward with entropy regularization, the overall policy objective is defined as Eq. 13, where L_i^k denotes the number of assembly decisions made by the `AttenAssemble` of client i in round k. In contrast, T_i^k denotes the number of modules included in the resulting model.

$$\max_{\phi_i} \; J_i(\phi_i) = \mathbb{E}_{\pi_{\phi_i}} \left[\underbrace{\lambda_E \, \text{Acc}_i^k + (1 - \lambda_E) \, E_i^k}_{\text{terminal reward}} + \underbrace{\lambda_{\mathcal{H}} \frac{1}{L_i^k} \sum_{t=1}^{L_i^k} \mathcal{H}\big(\pi_{\phi_i}(\cdot \mid o_{i,t}^k)\big)}_{\text{entropy regularization}} \right], \tag{13}$$

The structural exploration reward and entropy regularization serve complementary roles: The former encourages diverse module compositions at the trajectory level, while the latter prevents the step-wise action distribution from prematurely collapsing, particularly to the `STOP` action. Thus, the reward guides what structures to explore, while entropy preserves the action diversity for discovery.

A.3 Complete Algorithm

Algorithm 1 summarizes the training and execution procedure of `FedJigsaw`. In each communication round, `AttenAssemble` constructs a personalized model, `SymbioArchitect` supports module sharing between neighbours. When enabled, `CoRe-Tune` additionally coordinates policy optimization using cross-client information. During execution, each client independently executes its assembly policy using only locally available observations.

Algorithm 1 `FedJigsaw` Training and Decentralized Execution

Require: Client graph $\mathcal{G}$, local datasets $\{\mathcal{D}_i\}_{i=1}^N$, initial repositories $\{\mathcal{R}_i^0\}_{i=1}^N$, and capacity budgets $\{B_i\}_{i=1}^N$
Ensure: Personalized assembled models $\{f_i\}_{i=1}^N$

Initialize local actors $\{\pi_{\phi_i}\}_{i=1}^N$ and centralized critic V_ψ
for communication round $k = 1, \ldots, K$ **do**
  for all client $i \in \mathcal{N}$ **in parallel do**
    Advertise the metadata of $\mathcal{R}_i^k$ to neighboring clients
    Construct candidate pool $\mathcal{C}_i^k \leftarrow \bigcup_{u \in \{i\} \cup \mathcal{N}_i} \mathcal{R}_u^k$
    Initialize partial assembly $\mathcal{S}_i^k \leftarrow \emptyset$ and step $t \leftarrow 1$
    repeat
      Construct observation $o_{i,t}^k$ from the local data token, current assembly, and candidate metadata
      Determine compatible actions $\mathcal{A}_{i,t}^k \subseteq \mathcal{C}_i^k \cup \{\texttt{STOP}\}$
      Sample $a_{i,t}^k \sim \pi_{\phi_i}(\cdot \mid o_{i,t}^k)$
      if $a_{i,t}^k \neq \texttt{STOP}$ **then**
        Append the selected module to $\mathcal{S}_i^k$
      end if
      $t \leftarrow t + 1$
    until $a_{i,t-1}^k = \texttt{STOP}$ or capacity B_i is reached
    Retrieve the parameters of selected remote modules
    Construct and locally train the assembled model f_i^k
    Evaluate f_i^k and compute terminal reward r_i^k
    Return updated module copies to their source clients
  end for
  for all source module (u, j) **do**
    Aggregate the returned copies of (u, j) and update $\theta_{u,j}^{k+1}$
  end for
  if b_{CTDE} **then**
    Collect encoded cross-client summaries
    Construct cross-client context $Z_{i,t}^k$
    Update critic V_ψ using the observed returns
    $\hat{A}_{i,t}^k \leftarrow G_{i,t}^k - V_\psi(h_{i,t}^k, Z_{i,t}^k)$
  else
    $\hat{A}_{i,t}^k \leftarrow G_{i,t}^k$
  end if
end for

Decentralized execution: each client selects the highest-probability valid action using only its local observation; the centralized critic is discarded.

A.4 Supplementary Experiments

A.4.1 Detailed Experiment Setup

Implementation and Environment We implement all methods in PyTorch 2.7.1 and execute them in isolated Docker containers based on Ubuntu 24.04. Full-scale experiments are conducted on a Linux server equipped with NVIDIA RTX PRO 6000 GPU with 96GB GRAM and 256 GB

Table 4: Reference model configurations and relative capacity ratios. Within each model family, $\rho_q = F_q/F_{\text{Full}}$, where F_q denotes the FLOPs of tier q. FLOPs are measured per input at a fixed image resolution or sequence length.

Model	Quantity	Small	Medium	Large	Full
CNN	Conv. modules	2	4	6	8
	FLOPs (M)	39.53	152.80	266.08	303.83
VGG	Variant	11	13	16	19
	FLOPs (M)	343.36	494.35	664.22	834.09
RoBERTa	Encoder blocks	3	6	9	12
	FLOPs (G)	5.59	11.17	16.76	22.35

Dataset / Backbone	α	**Small**	**Medium**	**Large**	**Full**	**Util. (%)**	**Remote (%)**
CIFAR-10 / CNN	0.3	1.8/2	3.2/4	3.3/6	3.0/8	65.6	89.6
CIFAR-10 / CNN	1.0	1.4/2	4.0/4	4.4/6	5.1/8	76.8	93.3
AG News / RoBERTa-base	0.3	3.0/3	6.0/6	9.0/9	12.0/12	100.0	89.9
AG News / RoBERTa-base	1.0	3.0/3	6.0/6	9.0/9	12.0/12	100.0	96.3

Table 5: Learned assembly behavior across capacity tiers. Each depth entry reports the mean assembled depth and its corresponding capacity limit. Utilization is the ratio between assembled depth and capacity limit. Remote denotes the proportion of modules originating from neighbors.

of host memory, using CUDA 12.8 and cuDNN9. Mininet assigns each client and independent network namespace and carries inter-client and client-server communication over emulated TCP links. Auxiliary ablation and sensitivity experiments use the same implementation on an Apple M2 MAX CPU macbook with 64 GB RAM for an isolated environment for accurate statistical information.

Client Capacity Assignment Tab. 4 summarizes the four client-capacity tiers used in our experiments. Clients are evenly assigned to these tiers and retain the same assignments throughout training. For all baselines, we preserve the same correspondence between client and computational resources, and configure their method-specific capacity settings using the corresponding tier in Tab. 4 as the reference.

Baselines The original repositories of the baselines are listed as follows:

- FedRolex: `https://github.com/AIoT-MLSys-Lab/FedRolex`
- FIARSE: `https://github.com/HarliWu/FIARSE`
- FedMRL: `https://github.com/LipingYi/FedMRL`
- ProxyFL: `https://github.com/layer6ai-labs/ProxyFL`
- DeSA: `https://github.com/ubc-tea/DESA`
- PerFedRLNAS: `https://github.com/TL-System/plato/tree/main/examples/model_search/pfedrlnas/DARTS`
- pFedHR: `https://github.com/JackqqWang/pfedHR`

ProxyFL, DeSA, pFedHR, and PerFedRLNAS are not evaluated with RoBERTa-base because their release implementations do not provide a compatible model configuration. ProxyFL and DESA target vision backbones, while PerFEDRLNAS uses a DART-based CNN search space. We therefore do not extend these methods beyond their originally supported model families.

A.4.2 Analysis of Learned Assembly Behavior

We further examine what assembly behavior is learned by `AttenAssemble`. Specifically, we investigate whether the policy adapts the assembled depth to the available capacity and whether the

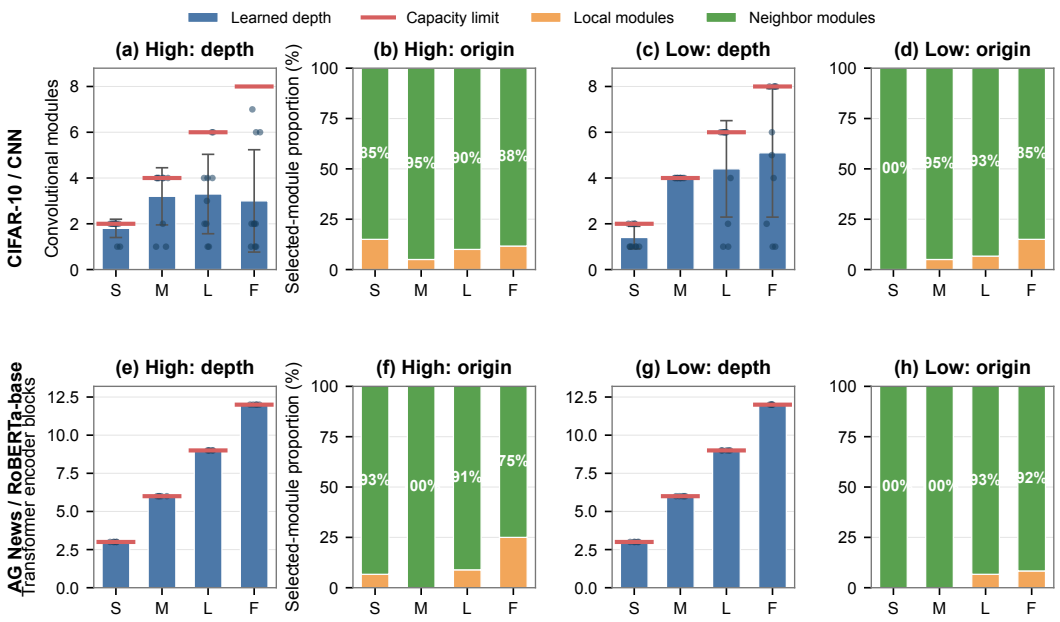


Figure 5: Learned assembly depth and module sharing rate. Bars and error bars report the mean and standard deviation, dots denote individual clients, and red markers indicate capacity limits.

resulting models actually reuse modules from other clients. For each client, we record the assembled depth and module provenance of its selected model, grouped by capacity tiers. Fig. 5 reports the resulting distributions under both data-heterogeneity settings for CNN and RoBERTa-base backbones.

Tab. 5 summarizes the corresponding mean depth, capacity utilizaation, and remote-module proportion. The capacity limit bounds the feasible depth without but does not necessarily determine its final depth. On high non-iid CIFAR-10, the mean assembled depth are 1.8, 3.2, 3.3, and 3.0 across 4 tiers, corresponding to 65.6% average utilization. When the partition becomes more balanced in a low non-iid setting, the assembled depths increase to 1.4, 4.0, 4.4, and 5.1 modules, with 76.8% utilization. This increase is most evident in the Medium, Large and Full tiers. A more balanced distribution exposes the clients to a broader label distribution and therefore presents a more complex local classification problem, for which the policy tends to retain deeper CNNs. Nevertheless, the selected depths generally remain below their capacity limits, consistent with CIFAR-10 being adequately handled by relatively compact CNNs.

The behavior becomes different on AG News as it is a more challenging task. RoBERTa-base clients select all 3, 6, 9, and 12 available encoder blocks under both $\alpha = 0.3$ and $\alpha = 1.0$. This suggests that additional Transformer depth remain useful for the text task regardless of the degree of data heterogeneity, as the classification is a challenging task itself. Together, these results show that the policy does not indiscriminately fill the available capacity, it terminates early for many CNN clients while using the full depth allowance for RoBERTa-base clients.

The provenance results further demonstrate extensive cross-client reuse. The remote-module proportion increases from 89.6% to 93.3% on CIFAR-10 and from 89.9% to 96.3% on AG News as α increase from 0.3 to 1.0. Thus, the policy relies more strongly on neighboring for knowledge sharing when client distributions become more similar. Hence, `FedJigsaw` does not merely rearrange modules from each client's local repository, but also utilize knowledge from neighbors with the requirement for knowledge sharing growth.

Study	Setting	Local Acc.	Shared Acc.	Round (s)	Comm. (MB)
Clients	5	68.16±5.26	22.85±1.73	1.37	0.631
	10	70.38±14.18	18.45±4.79	1.57	0.943
	15	66.03±11.12	20.05±4.39	1.77	0.710
	20	62.78±11.20	21.18±3.96	2.07	0.464
Degree	2	63.14±9.85	21.21±4.07	1.71	0.431
	4	62.78±11.20	21.18±3.96	**0.86**	0.349
	8	62.17±10.62	21.24±4.31	1.31	**0.283**
	19	61.42±9.93	21.08±3.82	1.17	1.237

Table 6: Sensitivity to federation size and neighborhood degree. Degree results are averaged over the final 50 rounds of the 300-round runs. Communication is reported per client per round.

A.4.3 Additional Ablation Analysis

Fig. 6 visualizes the accuracy efficiency trade-off corresponding to the ablation results in Tab. 2. Points closer to the upper-left corner are preferred, as they represent higher local accuracy with lower inference FLOPs.

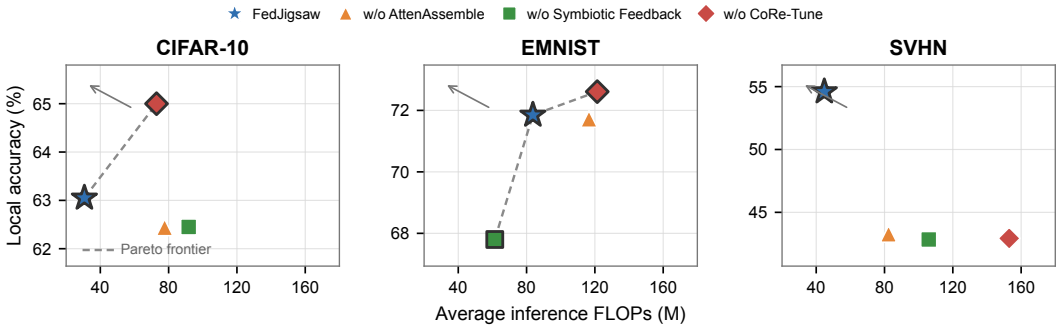


Figure 6: Local accuracy and inference-FLOPs trade-offs of `FedJigsaw` and its ablated variants.

A.4.4 Scalability and System Robustness

We further evaluate `FedJigsaw` under cahnges in federation scale, neighbourhood connectivty, client avaliability, and link condition. All experiments use CIFAR-10 with $\alpha = 0.3$. These robustness experiments use 300 communication rounds and are intended for relative sensitivity analysis.

Number of Clients and Neighborhood Degree. Tab. 6 reports both predictive performance and system cost. When the federation grows from 5 to 20 clients, local accuracy varies non-monotonically and shared-distribution accuracy remains between 18.45% and 22.85%. Meanwhile, independently measured round time increases from only 1.37 to 2.07 seconds rather than proportionally with the fourfold increase in clients. This result suggests that the round time does not grow linearly with the federation size under our execution setting. Per-client communication remains between 0.464 and 0.943 MB per round and does not increase monotonically with the federation size, since each client exchanges only its selected modules with neighbors.

Then with 20 clients fixed, increasing the degree from 2 to 19 changes local accuracy by 1.72 percentage points and shared-distribution accuracy by only 0.16 points. Full connectivity therefore provides no corresponding accuracy gain and incurs the largest communication cost, while degree four gives the lowest observed round time. In particular, full connectivity increases per-client communication to 1.237MB per round, approximately 3.5 times of degree four, without improving accuracy. Therefore, `FedJigsaw` does not require dense connectivity for effective knowledge sharing, and a moderately sparse topology offers a favorable balance between model quality and system overhead.

Study	Setting	Local Acc.	Shared Acc.	Round (s)
Participation	20%	59.33±12.63	19.04±3.14	0.99
	40%	60.68±10.13	20.06±3.63	1.20
	60%	61.32±10.44	19.68±3.93	0.99
	80%	61.87±10.00	20.36±3.92	1.11
	100%	62.78±11.20	21.18±3.96	0.86
Bandwidth	Unlimited	62.78±11.20	21.18±3.96	0.86
	100 Mbps	61.67±10.97	20.63±4.31	0.95
	20 Mbps	63.14±10.20	21.01±4.28	2.57
	5 Mbps	62.85±11.45	21.25±3.68	16.05

Table 7: Robustness to partial client participation and constrained link bandwidth with 20 clients and degree four. System measurements are averaged over the final 50 rounds.

Partial Participation and Link Bandwidth. We next sample 20%–100% of the 20 clients per round. As shown in Tab. 7, local accuracy improves from 59.33% to 62.78% and shared-distribution accuracy from 19.04% to 21.18% as participation increases. Nevertheless, the method remains functional with only 20% participation, and round time stays within 0.86–1.20 seconds because only active clients execute. Varying bandwidth has little effect on model quality: local and shared-distribution accuracy vary by at most 1.47 and 0.62 points, respectively. In contrast, reducing bandwidth from 100 to 5 Mbps increases round time from 0.95 to 16.05 seconds, showing that optimization is stable but execution becomes network-bound on slow links.

A.5 Known Limitations

First, the current implementation assembles modules sequentially into a chain-structured network. Although this formation simplifies compatibility checking and policy training, practical neural architectures are more naturally represented as directed acyclic graphs (DAGs) containing skip connections, parallel branches, and multi-input operators. Extending model reassembly from sequential planning to graph construction would introduce more flexibility.

Second, the current module zoo is inherited from fixed base models. While module parameters can be shared and updated, the underlying structural vocabulary, including module boundaries and connectivity patterns, remains predefined. A promising direction is to jointly optimize module reuse and structural evolution, allowing the federation to create, refine, and reorganize modules over time.

Finally, the data token is a latent summary rather than raw client data, but its use by the centralized critic does not provide a formal privacy guarantee and may still reveal information about local information. Further work could incorporate differential privacy, secure aggregation, or privacy-preserving representation learning into the policy training procedure.

A.6 Reproducibility statement

We provide the implementation, experimental configurations, and data-partitioning procedures required to reproduce our results. The appendix details the model architectures, client-capacity assignments, network settings, and evaluation protocols. All experiments use fixed random seeds, and the datasets are publicly available.

A.7 AI Use Statement

We used Chatgpt as assistive tools for manuscript polishing, improving the visual presentation of figures, and implementation debugging. It was used to improve clarity, grammar, and organization of the author's original writing. The research idea, method design, algorithm design and implementation, experimental protocal and discussion of experiment results were conducted by the authors. All LLMs suggestions are reviewed and verified by the authors, and the authors take the responsibility of the content of this paper.