MEMEVO-FCGL: STRUCTURAL MEMORY EVOLUTION FOR FEDERATED CONTINUAL GRAPH LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Federated continual graph learning (FCGL) enables distributed clients to collaboratively learn from sequential graph-task views without sharing local observations. However, clients must preserve previous knowledge while adapting to new graph patterns, and the server must coordinate structurally heterogeneous clients. Existing methods often describe task evolution using coarse task- or label-level statistics and preserve historical knowledge through raw-sample replay, limiting their ability to capture structural changes. We propose MemEvo-FCGL, a structural memory evolution framework for local knowledge retention and cross-client coordination. Each client learns memory keys that distill node representations into a compact state describing the current task's structural patterns. The server compares current states along client memory-state trajectories to adapt model aggregation and prototype-based knowledge transfer, reducing interference from structurally incompatible updates. Meanwhile, a bounded abstract key-value memory merges similar historical patterns, dynamically reorganizes novel patterns under a fixed capacity, and retrieves relevant knowledge through attention. Across six protocol-matched node-classification benchmarks, MemEvo-FCGL attains the highest final average performance on five while retaining a bounded number of abstract memory slots. These results demonstrate the effectiveness of structural memory evolution for continual learning over decentralized class-incremental graph streams. The code is available at `https://anonymous.4open.science/r/anonymous_accept`.

1 INTRODUCTION

Federated learning coordinates models trained on private client data without directly exchanging raw observations (McMahan et al., 2017). When these observations are graphs, topology also shapes the learned node representations. Federated continual graph learning (FCGL) must additionally accommodate sequential graph tasks while retaining earlier knowledge (Tang & Matteson, 2021; Serra & Buettner, 2025; Zhu et al., 2025; Wan et al., 2025). We study transductive, semi-supervised class-incremental node classification: new classes induce changing task-visible views of each private graph, only training-node labels provide supervision, and one classifier recognizes all observed classes. The graph itself is not assumed to gain or lose edges over time; features and edges of unlabeled task-visible nodes remain available for message passing.

The central difficulty is that the information used to retain and combine knowledge need not describe the graph patterns being learned. Figure 1 illustrates three connected consequences. First, clients with the same training-label counts can have different feature–neighborhood organization (Fig. 1(a)); counts therefore cannot determine whether their graph-conditioned representations are compatible. This distinction is particularly consequential on heterophilous graphs, where neighboring nodes need not share a label (Pei et al., 2020). Second, as new classes expose different task-induced views, a fixed memory budget must be reallocated among recurrent and newly observed patterns (Fig. 1(b)). Retaining individual nodes may spend capacity on redundant patterns, while loss-level rehearsal alone does not make historical information available to current-node embeddings (Zhou & Cao, 2021; Su et al., 2024). Third, clients can produce structurally incompatible updates even when their sample counts are similar (Fig. 1(c)). Aggregation based only on counts cannot assess this agreement and may mix conflicting knowledge as tasks proceed (Tan et al., 2024; Fu et al.,

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.

2025). These limitations couple local forgetting with cross-client interference rather than posing two independent problems.

Why structural memory? A GNN embedding reflects both node features and their graph neighborhoods, so its organization can distinguish patterns that label counts obscure. We ask whether a bounded representation of this organization can preserve useful history while also indicating which clients should contribute more strongly to global learning. We answer with *MemEvo-FCGL*: clients assign graph-conditioned embeddings to shared structural keys and summarize the assignments as current memory states. The states use features and connectivity from all task-visible nodes, including validation and test nodes, but not their labels. Across rounds, synchronized points on the client state trajectories guide both model aggregation and prototype-based knowledge transfer. Thus, the same structural coordinates link local evolution to cross-client coordination without communicating raw graph records.

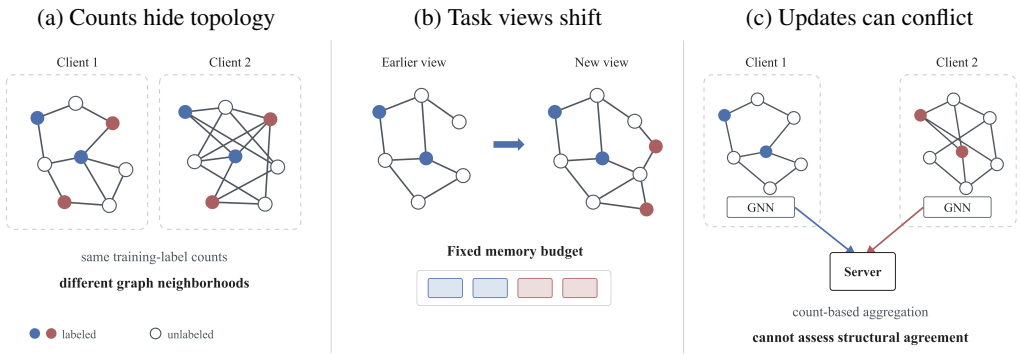


Figure 1: Three connected challenges in FCGL. **(a)** Equal training-label counts can conceal different graph structures. **(b)** Task-induced graph views change while the memory budget remains fixed. **(c)** Count-based aggregation alone cannot assess structural agreement between clients.

At task boundaries, each client condenses labeled training-node information into a bounded abstract key–value buffer. It merges compatible historical and new slots, and reorganizes the slots when capacity is reached, so that retention is not tied to a fixed set of previously selected nodes. During the next task, retrieval from the K_h most similar memory slots injects stored values into current-node representations before classification; semantic consistency, old–new margin, and encoder-stability objectives provide complementary protection against forgetting (Hinton et al., 2015; Kirkpatrick et al., 2017). This read-before-write cycle makes history part of current representation learning, not only an additional loss on past examples (Tang & Matteson, 2021; Serra & Buettner, 2025). The shared keys align clients for coordination, while memory values and semantic targets stay private; a common structural space does not require sharing historical exemplars.

Our contributions are summarized as follows:

- We identify a coupled FCGL challenge: label counts obscure graph-conditioned differences, task views shift under fixed memory, and incompatible client updates interfere. Our insight is to use shared structural coordinates for both local retention and server coordination.
- We propose MemEvo-FCGL with three linked modules: structural-key states guide aggregation and teacher selection; bounded key–value memory consolidates historical patterns; and top-K_h retrieval injects memory into current-node embeddings. Private buffer contents remain local.
- Across eight graphs and ten baselines, MemEvo-FCGL achieves the highest final average accuracy on five of six protocol-matched datasets, including Squirrel, and ranks second on Chameleon. Forgetting metrics, ablations, and task-wise analyses further assess retention and coordination.

2 Related Work

Continual graph learning. CGLB distinguishes node- from graph-level learning and task- from class-incremental protocols, underscoring the importance of the evaluation setting (Zhang et al., 2022). For node-level retention, ER-GNN replays selected experience nodes (Zhou & Cao, 2021),

whereas TWP constrains parameters important to topology-aware aggregation (Liu et al., 2021). Han et al. (2024) introduce TACO, which maintains a reduced graph through topology-aware coarsening. These retention methods preserve history on centralized task streams; they do not coordinate updates across private client graphs.

Federated graph and continual learning. FedAvg aggregates client models using local sample counts (McMahan et al., 2017). FedSSP shares spectral knowledge while accounting for client preferences (Tan et al., 2024), and FedATH addresses topology heterogeneity from a causal perspective (Fu et al., 2025). These methods study graph heterogeneity without a continual task sequence. Conversely, Powder transfers knowledge through prompts (Piao et al., 2024), FedAGC corrects conflicting gradients (Zhang et al., 2025), and uncertainty-aware memory management supports online federated continual learning (Serra & Buettner, 2025); none uses task-conditioned graph neighborhoods as a shared coordination signal.

Federated continual graph learning. POWER mitigates local forgetting with experience-node replay and addresses global expertise conflict through pseudo-prototype reconstruction and trajectory-aware transfer (Zhu et al., 2025). MOTION retains topology through evolutionary coarsening and refines aggregation using topology-sensitive client compatibility (Wan et al., 2025). MemEvo-FCGL instead stores bounded, abstract key–value patterns rather than raw experience nodes or a coarsened graph. Shared structural keys summarize task-conditioned client states; their trajectories guide aggregation and prototype transfer, while memory retrieval brings relevant history into current-node representations.

3 Problem Formulation

Setting. We consider a server and K clients, each with a private graph $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k, \boldsymbol{X}_k, \boldsymbol{y}_k)$ (Zhu et al., 2025). Here $\mathcal{V}_k$ and $\mathcal{E}_k$ are the node and edge sets, $\boldsymbol{X}_k$ is the node-feature matrix, and $\boldsymbol{y}_k$ contains benchmark node labels. The benchmark labels $\boldsymbol{y}_k$ define an offline sequence of T tasks with disjoint class sets $\{\mathcal{C}^t\}_{t=1}^T$, but are not all available for training. At task t, the training, validation, and test nodes induce a task-visible view $\mathcal{G}_k^t$ of the fixed underlying graph; the protocol does not assume edge insertion or deletion. Let $\mathcal{V}_k^t$ be its visible nodes and $\mathcal{V}_{k,\mathrm{tr}}^t \subseteq \mathcal{V}_k^t$ its training nodes. In this transductive, semi-supervised setting, features and edges of all visible nodes may enter message passing, whereas only training-node labels enter supervised losses and memory writes. Validation and test labels are neither training targets nor uploaded. A single classifier predicts all classes seen so far without a task identifier. Each task comprises R communication rounds: the server broadcasts the model and shared structural keys, and clients return model-side quantities and derived summaries while retaining a bounded local memory $\mathcal{B}_k^{t-1}$. The initial memory is empty, $\mathcal{B}_k^0 = \varnothing$; we write $\mathcal{C}^{<t} = \bigcup_{g<t} \mathcal{C}^g$ for previously seen classes. Raw graph records are not uploaded; this does not imply a formal privacy guarantee.

Evaluation. Let $A_{t,j}$ be task-j test accuracy after learning through task t, weighted across clients by their task-j test-node counts. We report final average performance (AM) and signed average forgetting (FM):

$$\mathrm{AM} = \frac{1}{T}\sum_{j=1}^{T} A_{T,j}, \qquad \mathrm{FM} = \frac{1}{T-1}\sum_{j=1}^{T-1}(A_{j,j} - A_{T,j}). \tag{1}$$

Higher AM and lower FM are preferred. FM is defined for $T > 1$; negative FM indicates positive backward transfer (Zhu et al., 2025). The main notation is collected in Appendix A.

4 MemEvo-FCGL

MemEvo-FCGL couples local retention and cross-client coordination through a shared structural key space (Figure 2). We describe structural states for server coordination, bounded client memory, historical retrieval, and the training objective in turn. Algorithm 1 summarizes the pipeline. Key activations provide aligned summaries for server-side decisions, whereas the abstract values and targets used for retention remain in private buffers.

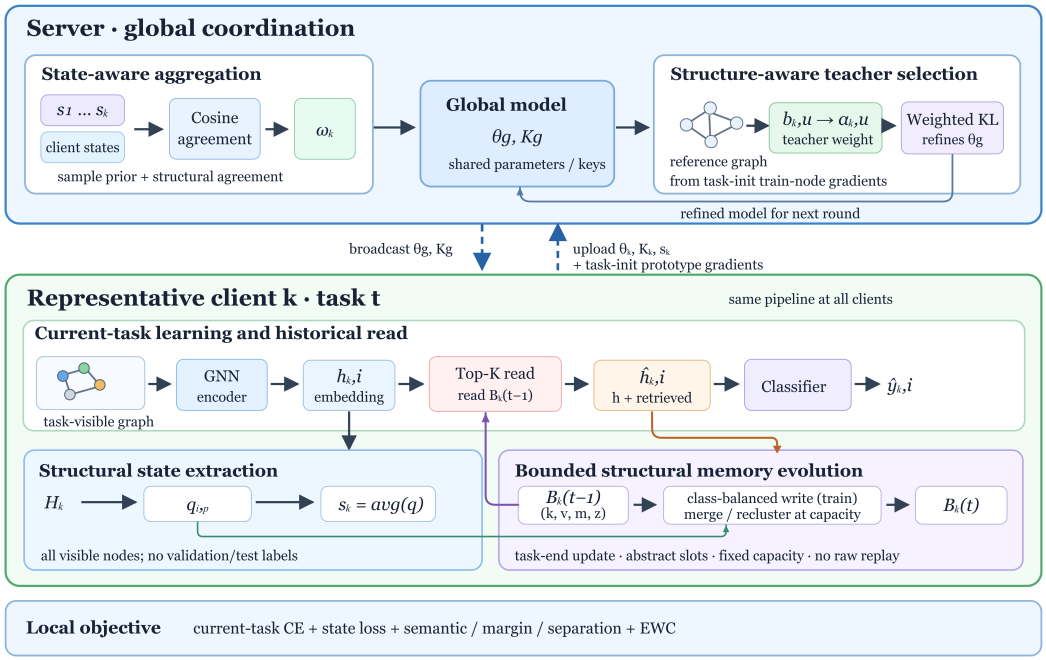


Figure 2: Overview of MemEvo-FCGL. A shared structural key space connects federated coordination at the server with continual memory consolidation and retrieval at the clients.

4.1 Structural Memory State for Federated Coordination

Aligned structural states derived from graph-conditioned embeddings guide client aggregation and reference-node teacher selection.

Memory state encoding. Let $\boldsymbol{h}_{k,i}$ be the base GNN embedding of task-visible node i at client k, and let $\{\boldsymbol{k}_{k,p}\}_{p=1}^{P}$ be its P learnable structural keys. Write $\boldsymbol{K}_k = [\boldsymbol{k}_{k,1}, \ldots, \boldsymbol{k}_{k,P}]$ and $\boldsymbol{K}_g$ for the local and server banks, and $\boldsymbol{\theta}_k, \boldsymbol{\theta}_g$ for their GNN parameters. For $\tau_s > 0$, write $u_{k,i,p} = (1 + \|\boldsymbol{h}_{k,i} - \boldsymbol{k}_{k,p}\|_2^2/\tau_s)^{-(\tau_s+1)/2}$. The Student-$t$ assignment (Xie et al., 2016) is

$$q_{k,i,p} = u_{k,i,p} / \sum_{p'=1}^{P} u_{k,i,p'}. \tag{2}$$

The assignment vector $\boldsymbol{q}_{k,i} = (q_{k,i,1}, \ldots, q_{k,i,P})$ reflects both features and connectivity through GNN message passing, without encoding edges or motifs explicitly. For labeled training nodes, let $f_{k,p} = \sum_{j \in \mathcal{V}_{k,\mathrm{tr}}^t} q_{k,j,p}$. We sharpen assignments with a detached target and learn the keys by

$$\begin{aligned} p_{k,i,p} &= \frac{q_{k,i,p}^2 / f_{k,p}}{\sum_{p'=1}^{P} q_{k,i,p'}^2 / f_{k,p'}}, \\ \mathcal{L}_{\mathrm{state}} &= \frac{1}{|\mathcal{V}_{k,\mathrm{tr}}^t|} \sum_{i \in \mathcal{V}_{k,\mathrm{tr}}^t} \mathrm{KL}(\boldsymbol{p}_{k,i} \| \boldsymbol{q}_{k,i}). \end{aligned} \tag{3}$$

Here $\boldsymbol{p}_{k,i} = (p_{k,i,1}, \ldots, p_{k,i,P})$ is detached, and frequency correction limits domination by heavily occupied keys. Training assignments use history-enhanced embeddings when available; after training, base embeddings of *all* task-visible nodes form

$$\boldsymbol{s}_k^{t,r} = \frac{1}{|\mathcal{V}_k^t|} \sum_{i \in \mathcal{V}_k^t} \boldsymbol{q}_{k,i} \in \Delta^{P-1}. \tag{4}$$

For round $r \in \{1, \ldots, R\}$, the server compares current synchronized states $\boldsymbol{s}_k^{t,r}$, not stored histories. The broadcast keys align their coordinates; state extraction uses no labels. Here Δ^{P-1} is the probability simplex over keys.

State-aware aggregation. We compare clients by cosine similarity of their current states (indices t, r omitted below). For $K > 1$, let $a_k = \frac{1}{K-1}\sum_{\ell \neq k}(1 + \cos(\boldsymbol{s}_k, \boldsymbol{s}_\ell))/2$; set $a_1 = 1$ when $K = 1$. Here $n_k^t = |\mathcal{V}_k^t|$ counts task-visible nodes. With sample-count prior $\pi_k = n_k^t / \sum_\ell n_\ell^t$, as in FedAvg (McMahan et al., 2017), the server uses

$$\omega_k = (1 - \lambda_a)\pi_k + \lambda_a \frac{a_k}{\sum_\ell a_\ell}, \quad (\boldsymbol{\theta}_g, \boldsymbol{K}_g) = \sum_k \omega_k(\boldsymbol{\theta}_k, \boldsymbol{K}_k). \tag{5}$$

With $\lambda_a \in [0, 1]$, $\lambda_a = 0$ recovers sample-count averaging; larger values favor clients whose current states agree with the federation. Thus, the update reflects both the amount of task-visible data at a client and the structural modes it shares with other clients.

Structure-conditioned knowledge transfer. On a server reference graph reconstructed from task-initial prototype gradients of training nodes (Zhu et al., 2025), each client predicts a reference-node assignment $\widetilde{\boldsymbol{q}}_{k,u}$. Its match to the client's current state determines a node-specific teacher weight:

$$b_{k,u} = \sum_p \sqrt{\widetilde{q}_{k,u,p} s_{k,p}}, \qquad \alpha_{k,u} = b_{k,u} / \sum_\ell b_{\ell,u}. \tag{6}$$

For global and local predictive distributions $\boldsymbol{\rho}_{g,u}$ and $\boldsymbol{\rho}_{k,u}$, the server minimizes a class-balanced, $\alpha_{k,u}$-weighted $\mathrm{KL}(\boldsymbol{\rho}_{g,u} \| \boldsymbol{\rho}_{k,u})$. Unlike ω_k, $\alpha_{k,u}$ varies across reference nodes. The reference graph is common to all clients, but different nodes can draw their teacher signal from different mixtures of local models.

4.2 Bounded Structural Memory Evolution

Client k retains at most B abstract slots $\mathcal{B}_k^t = \{(\boldsymbol{k}_j, \boldsymbol{v}_j, m_j, \boldsymbol{z}_j)\}$: structural key, summed value, assignment mass, and semantic target, with $m_j > 0$ and represented embedding $\boldsymbol{v}_j / m_j$. The buffer retains no raw node features or individual labels.

Class-balanced memory write. At the task boundary, current keys are allocated approximately evenly among observed training classes using class-wise assignment strengths. For key p allocated to class $o(p)$, let $\mathcal{I}_p = \{i \in \mathcal{V}_{k,\mathrm{tr}}^t : y_i = o(p)\}$ denote its labeled training nodes. These nodes produce

$$m_p^{\mathrm{cur}} = \sum_{i \in \mathcal{I}_p} q_{k,i,p}, \quad \boldsymbol{v}_p^{\mathrm{cur}} = \sum_{i \in \mathcal{I}_p} q_{k,i,p} \widehat{\boldsymbol{h}}_{k,i}, \quad \boldsymbol{z}_p^{\mathrm{cur}} = \mathbf{e}_{o(p)}. \tag{7}$$

Here y_i is the training label, $\mathbf{e}_{o(p)}$ is the one-hot class target, and $\widehat{\boldsymbol{h}}_{k,i}$ includes the historical read below. Keys describe current modes while values incorporate retrieved history.

Similarity-based consolidation. Current and historical slots are paired one-to-one in descending key cosine similarity when key and target similarities exceed δ_{key} and δ_z. For a matched historical slot j and current slot p, each field is updated as

$$x_j^+ = \mu x_j + (1 - \mu) x_p^{\mathrm{cur}}, \qquad x \in \{\boldsymbol{k}, \boldsymbol{v}, m, \boldsymbol{z}\}. \tag{8}$$

Here $\mu \in [0, 1]$ controls the historical share. The updated target is normalized; unmatched current slots are appended. Matching precedes reclustering so that recurring modes can absorb new evidence before the capacity constraint requires compression.

Capacity-triggered reorganization. If the candidate set exceeds B, slots are clustered in a joint space of normalized keys and normalized targets scaled by λ_z:

$$\mathbf{r}_j = \left[\frac{\boldsymbol{k}_j}{\|\boldsymbol{k}_j\|_2} ; \lambda_z \frac{\boldsymbol{z}_j}{\|\boldsymbol{z}_j\|_2} \right]. \tag{9}$$

The nonnegative λ_z controls semantic influence, distinguishing structurally similar slots with different targets. Clustering uses class-aware quotas where feasible. Each cluster $\mathcal{C}_b$ becomes

$$m_b = \sum_{j \in \mathcal{C}_b} m_j, \quad \boldsymbol{v}_b = \sum_{j \in \mathcal{C}_b} \boldsymbol{v}_j, \quad (\boldsymbol{k}_b, \boldsymbol{z}_b) = \frac{\sum_{j \in \mathcal{C}_b} m_j(\boldsymbol{k}_j, \boldsymbol{z}_j)}{m_b}. \tag{10}$$

Summed values and masses preserve the represented mean $\boldsymbol{v}_b/m_b$.

4.3 Historical Memory Interaction

For $t > 1$, each task-visible node retrieves from $\mathcal{B}_k^{t-1}$ without its label. Following historical-state use in dynamic GNNs (Su et al., 2024), we select the K_h keys most similar to $\boldsymbol{h}_{k,i}$ and compute

$$a_{i,j} = \mathrm{softmax}_{j \in \mathcal{N}_i^{K_h}} \left(\frac{\cos(\boldsymbol{h}_{k,i}, \boldsymbol{k}_j)}{\tau_h} \right), \quad \widehat{\boldsymbol{h}}_{k,i} = \boldsymbol{h}_{k,i} + \sigma \sum_{j \in \mathcal{N}_i^{K_h}} a_{i,j} \frac{\boldsymbol{v}_j}{m_j}. \tag{11}$$

Here $\mathcal{N}_i^{K_h}$ contains the retrieved slots, $\tau_h > 0$ is the temperature, and σ scales the readout. Classification and memory write use $\widehat{\boldsymbol{h}}_{k,i}$; dividing by m_j normalizes the slot value. Because retrieval uses the base embedding, it does not require the query node's label or a task identifier at inference.

Semantic consistency on abstract memory. The current classifier predicts logits $\boldsymbol{l}_j$ on a nearest-neighbor graph of stored keys, using $\boldsymbol{v}_j/m_j$ as features. Assign slot j to the old task with greatest target mass; $\mathcal{J}_g$ contains its slots and $\mathcal{P}_{t-1} = \{g < t : |\mathcal{J}_g| > 0\}$. Restricting $\boldsymbol{z}_j$ to task g and normalizing at temperature $\tau_d > 0$ gives the zero-padded target $\widetilde{\boldsymbol{z}}_j^g$ (Hinton et al., 2015). Task-balanced distillation is

$$\mathcal{L}_{\mathrm{sem}} = \frac{1}{|\mathcal{P}_{t-1}|} \sum_{g \in \mathcal{P}_{t-1}} \frac{\tau_d^2}{|\mathcal{J}_g|} \sum_{j \in \mathcal{J}_g} \mathrm{KL}\big(\widetilde{\boldsymbol{z}}_j^g \big\| \mathrm{softmax}(\boldsymbol{l}_j/\tau_d)\big), \tag{12}$$

and set it to zero if no old task is represented.

Old–new boundary protection. To protect old-class targets, let $\mathcal{J}_{\mathrm{old}} = \{j : \sum_{c \in \mathcal{C}^{<t}} z_{j,c} > 0\}$ and $c_j^* = \arg\max_{c \in \mathcal{C}^{<t}} z_{j,c}$ for $j \in \mathcal{J}_{\mathrm{old}}$. We impose

$$\mathcal{L}_{\mathrm{margin}} = \frac{1}{|\mathcal{J}_{\mathrm{old}}|} \sum_{j \in \mathcal{J}_{\mathrm{old}}} \left[\gamma - \ell_{j,c_j^*} + \max_{c \in \mathcal{C}^t} \ell_{j,c} \right]_+, \tag{13}$$

where $\gamma > 0$, $[x]_+ = \max(x, 0)$, and the term is zero if either set is empty.

4.4 Training Objective

The objective combines current-task cross-entropy $\mathcal{L}_{\mathrm{cur}}$ on training nodes, structural-state learning, retention, key separation, and encoder stability. $\mathcal{L}_{\mathrm{sep}}$ penalizes squared cosine similarity between keys; following EWC (Kirkpatrick et al., 2017), $\mathcal{L}_{\mathrm{enc}}$ penalizes drift from task-boundary anchors using diagonal squared-gradient importance. We optimize

$$\mathcal{L}_{\mathrm{local}} = \mathcal{L}_{\mathrm{cur}} + \lambda_s \mathcal{L}_{\mathrm{state}} + \widehat{\mathcal{L}}_{\mathrm{sem}} + \lambda_m \mathcal{L}_{\mathrm{margin}} + \lambda_{\mathrm{sep}} \mathcal{L}_{\mathrm{sep}} + \widehat{\mathcal{L}}_{\mathrm{enc}}. \tag{14}$$

The nonnegative λ_s, λ_m, and λ_{sep} weight their respective terms; hats indicate optional detached rescaling relative to $\mathcal{L}_{\mathrm{cur}}$. After each task, clients update anchors, importance, and private buffers. Communication carries model-side quantities, states, and training-derived prototype gradients, but no abstract values, masses, or targets. Inference uses one classifier and available memory without task selection; validation/test labels never serve as training targets.

5 Experiments

We evaluate MemEvo-FCGL from three complementary perspectives: final performance across graph datasets (Table 1), the contribution of each component (Table 2), and the evolution of accuracy and forgetting over a task stream (Figure 3). Together, these experiments ask whether the method learns new classes without losing earlier ones, whether its three modules are necessary, and when differences from representative baselines emerge.

Dataset	Cora		CiteSeer		Computers		Amazon Photo		PubMed		Chameleon		Squirrel		Coauthor CS	
Method	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓
FedAvg	25.74 ±0.75	94.55 ±1.21	29.48 ±0.61	80.66 ±0.98	18.36 ±0.71	97.98 ±0.20	21.68 ±0.72	98.39 ±0.25	52.11 ±3.71	72.66 ±14.80	34.43 ±1.35	68.88 ±1.88	33.84 ±0.54	53.15 ±1.54	22.60 ±4.02	87.11 ±4.51
FedDyn	26.02 ±4.56	74.40 ±5.39	20.26 ±4.98	78.36 ±1.97	17.93 ±6.35	73.77 ±7.33	19.26 ±4.16	73.62 ±11.78	37.86 ±16.33	56.31 ±27.29	40.64 ±3.23	51.53 ±8.15	35.63 ±2.02	52.08 ±2.88	14.51 ±5.43	45.90 ±4.41
FedSSP	34.70 ±2.62	73.68 ±3.92	32.98 ±2.17	69.60 ±4.17	17.68 ±0.03	88.11 ±2.50	30.96 ±7.66	81.91 ±11.74	55.46 ±6.03	48.34 ±20.35	38.53 ±1.25	51.42 ±1.66	37.10 ±1.75	41.18 ±5.71	50.86 ±2.66	51.91 ±3.18
FedATH	38.76 ±2.30	73.98 ±2.97	31.32 ±0.34	77.51 ±2.02	17.91 ±2.94	88.82 ±8.91	23.88 ±2.39	94.33 ±3.15	50.35 ±0.24	92.72 ±0.69	37.80 ±1.20	53.31 ±3.07	33.59 ±0.08	58.64 ±1.53	43.88 ±10.17	61.86 ±11.88
Fed-ERGNN	56.04 ±2.72	40.91 ±3.78	59.96 ±2.96	29.99 ±3.82	11.38 ±12.17	25.90 ±3.30	52.80 ±9.54	16.46 ±8.01	45.58 ±0.81	**3.11 ±1.64**	36.26 ±12.55	21.21 ±4.45	48.37 ±1.13	29.23 ±3.25	64.41 ±8.15	35.98 ±9.93
Fed-TWP	33.63 ±3.49	83.36 ±4.68	48.37 ±4.36	52.02 ±6.74	35.70 ±8.37	75.95 ±10.46	29.10 ±7.49	88.51 ±10.63	50.01 ±0.01	94.08 ±0.12	**69.84 ±1.73**	**3.42 ±1.93**	47.86 ±8.76	37.87 ±13.60	64.16 ±10.31	38.86 ±11.91
Powder	34.73 ±11.04	49.30 ±10.84	30.12 ±0.25	81.17 ±0.66	19.42 ±0.44	98.90 ±0.23	24.23 ±1.06	97.74 ±1.58	50.17 ±0.26	89.43 ±4.85	21.94 ±13.99	58.13 ±3.83	39.86 ±7.65	41.48 ±12.15	14.45 ±6.16	47.08 ±5.00
FedAGC	42.71 ±3.62	70.41 ±5.09	32.84 ±1.26	75.82 ±2.62	25.22 ±4.03	89.39 ±3.92	32.22 ±3.71	84.76 ±4.10	50.26 ±0.17	92.80 ±0.26	39.24 ±0.93	63.34 ±2.28	33.83 ±0.15	63.71 ±1.64	61.27 ±9.23	42.17 ±10.70
POWER	44.75 ±4.83	19.69 ±5.03	15.41 ±2.69	44.78 ±4.24	27.38 ±5.92	**6.63 ±6.29**	17.86 ±5.96	28.87 ±11.58	44.91 ±1.14	3.63 ±2.24	17.74 ±1.99	24.41 ±9.34	20.72 ±3.99	**14.62 ±4.03**	53.68 ±6.70	37.77 ±6.00
MOTION	57.19 ±3.68	8.55 ±2.42	50.74 ±2.40	8.61 ±1.76	63.51 ±8.94	10.42 ±7.38	82.91 ±1.83	12.52 ±2.21	46.96 ±8.45	7.70 ±1.76	20.01 ±1.79	5.36 ±1.23	19.92 ±1.99	23.49 ±1.15	25.62 ±4.50	7.93 ±5.30
MemEvo-FCGL	**79.65 ±2.64**	**7.09 ±1.97**	**64.23 ±1.83**	**8.08 ±2.67**	**74.27 ±2.44**	9.73 ±0.61	**83.17 ±1.29**	**3.63 ±0.33**	**75.30 ±0.59**	8.33 ±1.94	57.63 ±0.96	6.42 ±0.77	**53.14 ±4.26**	21.39 ±2.63	**88.38 ±0.19**	**4.28 ±0.12**

Table 1: Overall performance under federated class-incremental node classification. AM (↑) and signed FM (↓) are reported as mean ± standard deviation over five runs. Bold and underline denote the best and second-best displayed means, respectively; these descriptive rankings include runs with protocol differences as detailed in Section 5.1.

5.1 Experimental Setup

Datasets and baselines. We use eight node-classification graphs spanning citation networks (Cora, CiteSeer, and PubMed (Yang et al., 2016)), product and coauthor graphs (Computers, Photo, and Coauthor CS (Shchur et al., 2018)), and heterophilous graphs (Chameleon and Squirrel (Pei et al., 2020)). This selection tests the method across different graph structures rather than a single citation benchmark. Louvain partitioning yields three client subgraphs (Blondel et al., 2008), and each task introduces at most two new classes. We compare with ten baselines covering standard FL (FedAvg, FedDyn), FGL (FedSSP, FedATH), federated CGL adaptations (Fed-ERGNN, Fed-TWP), FCL (Powder, FedAGC), and FCGL (POWER, MOTION). These groups distinguish gains over generic federation from gains over methods designed for graph structure or continual learning; task splits and baseline adaptations are specified in Appendices B and C.

Evaluation protocol. We report final average performance (AM) and signed forgetting (FM) as defined in Section 3; each cell is the mean ± standard deviation over five runs. MemEvo-FCGL uses a two-layer, 64-dimensional GAT (Veličković et al., 2018) with 15 communication rounds and five local epochs per task. On Computers, Photo, PubMed, Chameleon, Squirrel, and Coauthor CS, six baselines (FedATH, Fed-ERGNN, Fed-TWP, FedAGC, Powder, and POWER) share its client, task, backbone, and training protocol and use paired seeds. The other entries in Table 1 are retained for context but are not controlled comparisons. FM is calculated from saved task-accuracy matrices without clipping; optimizer, memory, and protocol details appear in Appendix B.

Variant	A	B	C	Cora		CiteSeer		Computers		Chameleon		Squirrel	
				AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓	AM ↑	FM ↓
MemEvo-FCGL	✓	✓	✓	**79.65 ±2.64**	**7.09 ±1.97**	**64.23 ±1.83**	**8.08 ±2.67**	**74.27 ±2.44**	**9.73 ±0.61**	**57.63 ±0.96**	**6.42 ±0.77**	**53.14 ±4.26**	**21.39 ±2.63**
w/o A	×	✓	✓	31.25 ±0.95	93.97 ±0.55	30.02 ±0.32	80.00 ±0.84	20.42 ±1.71	88.04 ±5.82	33.33 ±0.00	70.71 ±2.16	30.95 ±1.30	62.81 ±1.92
w/o C	✓	✓	×	50.45 ±5.39	37.21 ±7.05	33.99 ±8.96	52.06 ±12.15	21.03 ±3.59	46.95 ±11.77	39.55 ±5.38	33.31 ±18.13	27.67 ±1.94	51.52 ±22.07
w/o A,C	×	✓	×	44.72 ±8.67	41.93 ±8.77	32.45 ±10.32	49.76 ±11.20	21.98 ±6.05	41.16 ±13.04	38.99 ±4.94	40.93 ±19.25	27.68 ±1.65	62.95 ±3.72
w/o B,C	✓	×	×	45.08 ±12.26	14.02 ±1.10	33.31 ±13.97	32.55 ±13.02	17.09 ±0.92	20.23 ±0.52	18.82 ±3.24	14.13 ±2.15	27.46 ±1.46	31.00 ±4.75
w/o A,B,C	×	×	×	40.64 ±10.87	13.35 ±5.07	36.02 ±14.92	33.69 ±11.18	18.62 ±2.24	31.79 ±1.83	33.33 ±0.00	73.06 ±2.81	28.51 ±1.97	59.81 ±5.75

Table 2: Contribution of the three MemEvo-FCGL components. A denotes coordination based on current points along client state trajectories, B denotes bounded structural memory evolution, and C denotes historical memory interaction. Results are reported as mean ± standard deviation over paired seeds.

5.2 Overall Performance

Table 1 reports endpoint AM and FM. We interpret method differences on the six datasets with a matched protocol; the remaining entries provide context, not controlled rankings. MemEvo-FCGL obtains the highest AM on five of these six datasets. The clearest joint gain occurs on Coauthor CS: compared with the strongest matched baseline in AM, Fed-ERGNN, our method improves AM from 64.41% to 88.38% while reducing FM from 35.98% to 4.28%. The result matters because high final accuracy here is accompanied by retention of earlier classes, rather than improved new-class learning at their expense.

The comparison also probes the structural issue raised in the Introduction. On heterophilous Squirrel, MemEvo-FCGL reaches 53.14% AM and 21.39% FM, versus 48.37% and 29.23% for the strongest matched baseline in AM, Fed-ERGNN. This is consistent with using graph-conditioned memory states when label counts alone may obscure neighborhood differences. The advantage is not universal. Lower FM alone can be misleading: on PubMed, Fed-ERGNN and POWER have lower FM (3.11% and 3.63% versus 8.33%), but their AM is only 45.58% and 44.91% versus our 75.30%; their task-accuracy matrices show zero accuracy on the newly introduced task. POWER likewise has lower FM but much lower AM on Computers and Squirrel. Chameleon is a genuine exception: Fed-TWP exceeds our method in both AM (69.84% versus 57.63%) and FM (3.42% versus 6.42%). The gap is especially large on the final task (87.18% versus 49.74%, averaged over seeds), indicating weaker new-class acquisition as well as retention on this split. The ablation below tests which proposed components contribute to the overall pattern.

5.3 Ablation Study

We ablate (A) state-aware coordination, (B) bounded structural memory evolution, and (C) historical memory interaction. The letters A–C label modules and are distinct from the slot capacity B. The five datasets in Table 2 include two citation graphs (Cora and CiteSeer), a product graph (Computers), and two heterophilous graphs (Chameleon and Squirrel). This subset tests whether each module matters across different graph structures, including settings where neighboring nodes need not share labels. All six variants use paired seeds and the same training protocol. Because C reads the memory constructed by B, removing B also removes C.

Table 2 shows that the complete model has the highest AM on all five datasets, including both heterophilous graphs. Removing A while retaining B and C lowers Cora AM from 79.65% to 31.25% and Chameleon AM from 57.63% to 33.33%. Thus, retaining local patterns is not enough when client updates must also be coordinated in a shared structural space. Conversely, removing C lowers Computers AM from 74.27% to 21.03% even though B still stores memory: a bounded buffer is most useful when its historical values can inform current-node representations. Removing B and C together yields only 17.09% AM on Computers despite 0.23% FM. This illustrates the key

evaluation caveat: little measured forgetting can merely reflect failure to learn the tasks. Since C depends on B and the variants do not have additive effects, these results support the three modules as a coupled design rather than assigning an independent effect size to B.

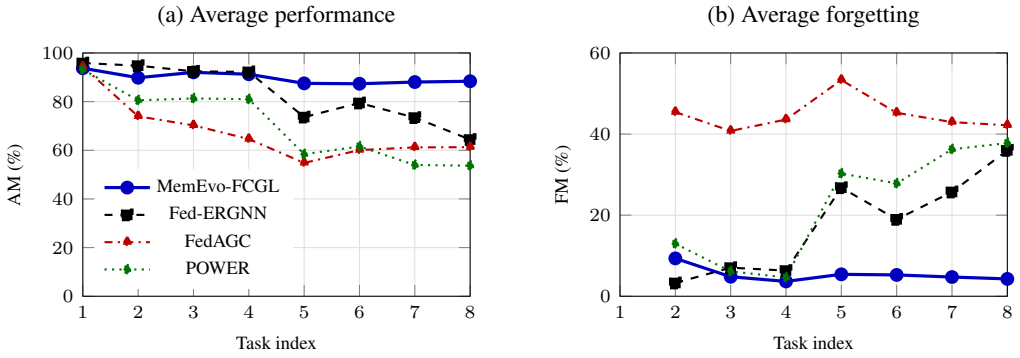


Figure 3: Task-wise performance on Coauthor CS. (a) Average accuracy over tasks seen through t; higher is better. (b) Signed average forgetting over earlier tasks; lower is better, and the metric is undefined at $t = 1$. Curves show means over five paired seeds with the same task order and training budget. The final-task standard deviations are reported in Table 1; colors and line styles are shared across the two panels.

5.4 Task-wise Performance

The endpoint scores in Table 1 do not show when performance changes as new classes arrive. We therefore use Coauthor CS for the task-wise view: its eight-task sequence provides a longer horizon over which retention and new-task learning can be observed. We select three protocol-matched baselines with distinct continual-learning mechanisms: Fed-ERGNN uses experience-node replay, FedAGC combines rehearsal with gradient correction, and POWER is an FCGL method that replays selected experience nodes. Figure 3 compares their trajectories under the same task order and training budget. Panel (a) averages accuracy over all tasks seen by each step, while panel (b) measures signed forgetting on earlier tasks. At task t, the two metrics, with FM defined only for $t > 1$, are

$$\mathrm{AM}_t = \frac{1}{t}\sum_{j\le t} A_{t,j}, \qquad \mathrm{FM}_t = \frac{1}{t-1}\sum_{j<t}(A_{j,j} - A_{t,j}). \tag{15}$$

Figure 3 reveals when the endpoint advantage arises. As the fifth task arrives, POWER's AM drops from 81.00% to 58.41% and its FM rises from 4.65% to 30.28%; Fed-ERGNN likewise falls from 92.21% to 73.65% AM. Over the same transition, MemEvo-FCGL changes from 91.25% to 87.51% AM, while FM remains low (3.66% to 5.41%). It maintains AM above 87% and FM below 10% throughout the measured stream, finishing at $88.38\%/4.28\%$ AM/FM versus $64.41\%/35.98\%$ for Fed-ERGNN, $61.27\%/42.17\%$ for FedAGC, and $53.68\%/37.77\%$ for POWER. The pattern is consistent with the proposed insight that, as new classes alter task-visible graph views, useful history must remain available to current representations under a fixed memory budget. The curves show this temporal effect; Table 2 provides the component-level evidence.

6 Conclusion

We presented MemEvo-FCGL for federated class-incremental learning on private graphs with changing task-induced views. Round-wise structural states guide server coordination, while a bounded abstract key–value memory consolidates past patterns and supplies relevant context to current nodes. Across the six protocol-matched datasets, the method attains the highest final AM on five and the second-highest on Chameleon; the full eight-dataset table additionally documents outcomes under broader, partly unmatched configurations. Our study does not model temporal edge insertion or deletion, nor does a bound on memory slots alone establish lower runtime or storage cost.

Ethics Statement

This work uses existing public graph benchmarks and does not involve newly recruited human participants or animal experiments. In keeping with the ICLR Code of Ethics, we report the privacy limitations of the evaluated setting. Raw graph records are not uploaded to the server. Clients retain bounded abstract key–value memory slots rather than raw node features or individual node labels for rehearsal; labeled training nodes contribute to slot statistics and semantic targets locally. Model updates and task-initial prototype gradients are communicated. These exchanges may reveal information about client data; we do not claim a formal privacy guarantee. Applications to sensitive graphs would require appropriate data-use permissions and additional safeguards. We do not make claims about demographic fairness, which is not evaluated here.

Reproducibility Statement

We describe the learning setting and evaluation metrics in Section 3, the method in Section 4 and Algorithm 1, and the experimental protocol in Section 5.1. Appendix B details task construction, implementation settings, and baseline adaptations. All eight benchmark datasets are public. We report means and standard deviations over five runs and retain the per-run configurations and task-accuracy matrices used to calculate AM and FM. The executable code is publicly available at `hhttps://anonymous.4open.science/r/anonymous_accept`.

References

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul N. Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=B7v4QMR6Z9w.

Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008. doi: 10.1088/1742-5468/2008/10/P10008. URL https://doi.org/10.1088/1742-5468/2008/10/P10008.

Lele Fu, Bowen Deng, Sheng Huang, Tianchi Liao, Shirui Pan, and Chuan Chen. Less is more: Federated graph learning with alleviating topology heterogeneity from a causal perspective. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 17727–17745. PMLR, 2025. URL https://proceedings.mlr.press/v267/fu25a.html.

Xiaoxue Han, Zhuo Feng, and Yue Ning. A topology-aware graph coarsening framework for continual graph learning. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-4212. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ef283d62b4bce30854a8d4827f331229-Abstract-Conference.html.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. doi: 10.48550/arXiv.1503.02531. URL https://arxiv.org/abs/1503.02531.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://openreview.net/forum?id=8gmWwjFyLj.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL https://www.pnas.org/doi/10.1073/pnas.1611835114.

Huihui Liu, Yiding Yang, and Xinchao Wang. Overcoming catastrophic forgetting in graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8653–8661, 2021. doi: 10.1609/aaai.v35i10.17049. URL https://ojs.aaai.org/index.php/AAAI/article/view/17049.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 1273–1282. PMLR, 2017. URL https://proceedings.mlr.press/v54/mcmahan17a.html.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Hongming Piao, Yichen Wu, Dapeng Wu, and Ying Wei. Federated continual learning via prompt-based dual knowledge transfer. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 40725–40739. PMLR, 2024. URL https://proceedings.mlr.press/v235/piao24a.html.

Giuseppe Serra and Florian Buettner. Federated continual learning goes online: Uncertainty-aware memory management for vision tasks and beyond. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=f65RuQgVlp.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS 2018*, 2018. doi: 10.48550/arXiv.1811.05868. URL https://arxiv.org/abs/1811.05868.

Junwei Su, Difan Zou, and Chuan Wu. PRES: Toward scalable memory-based dynamic graph neural networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/f5dbd98d426772945099ccace06418ba-Abstract-Conference.html.

Zihan Tan, Guancheng Wan, Wenke Huang, and Mang Ye. FedSSP: Federated graph learning with spectral knowledge and personalized preference. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-1090. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/3d226fb8fbd6ee6ec70d0427f1319707-Abstract-Conference.html.

Binh Tang and David S. Matteson. Graph-based continual learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=HHSEKOnPvaO.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Frank Wan, Fengyuan Ran, Ruikang Zhang, Wenke Huang, Xuankun Rong, Guibin Zhang, Yuxin Wu, Bo Du, and Mang Ye. MOTION: Multi-sculpt evolutionary coarsening for federated continual graph learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-2363. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/659afd140447b7d50fadf8bde6a2240d-Abstract-Conference.html.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 478–487. PMLR, 2016. URL https://proceedings.mlr.press/v48/xieb16.html.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 40–48. PMLR, 2016. URL https://proceedings.mlr.press/v48/yanga16.html.

Chengchao Zhang, Fanhua Shang, Hongying Liu, Liang Wan, and Wei Feng. FedAGC: Federated continual learning with asymmetric gradient correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3841–3850, 2025. doi: 10.1109/ICCV51701.2025.00366. URL https://openaccess.thecvf.com/content/ICCV2025/papers/Zhang_FedAGC_Federated_Continual_Learning_with_Asymmetric_Gradient_Correction_ICCV_2025_paper.pdf.

Xikun Zhang, Dongjin Song, and Dacheng Tao. CGLB: Benchmark tasks for continual graph learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. doi: 10.52202/068431-0945. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/548a41b9cac6f50dccf7e63e9e1b1b9b-Abstract.html.

Fan Zhou and Chengtai Cao. Overcoming catastrophic forgetting in graph neural networks with experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 4714–4722, 2021. doi: 10.1609/aaai.v35i5.16602. URL https://ojs.aaai.org/index.php/AAAI/article/view/16602.

Yinlin Zhu, Miao Hu, and Di Wu. Federated continual graph learning. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 4203–4213, 2025. doi: 10.1145/3711896.3736956. URL https://doi.org/10.1145/3711896.3736956.

A NOTATION

Table 3 lists the principal notation used in the problem formulation, method, and evaluation. Intermediate quantities are defined where they first appear.

Table 3: Notation used throughout the manuscript.

Notation	Description
K, T, R	Numbers of clients, tasks, and communication rounds per task.
$\mathcal{G}_k$	Private graph of client k.
$(\mathcal{V}_k, \mathcal{E}_k, \boldsymbol{X}_k, \boldsymbol{y}_k)$	Node set, edge set, feature matrix, and benchmark labels of $\mathcal{G}_k$.
$\mathcal{G}_k^t, \mathcal{V}_{k,\mathrm{tr}}^t$	Task-visible graph and its labeled training nodes.
$\mathcal{V}_k^t$	Nodes visible to client k at task t, including unlabeled nodes.
$\mathcal{C}^t, \mathcal{C}^{<t}$	New classes at task t and classes from earlier tasks.
$A_{t,j}$	Test accuracy on task j after learning through task t.
AM, FM	Final average performance and signed average forgetting.
$\mathrm{AM}_t, \mathrm{FM}_t$	Average performance and signed forgetting through task t.
$P, \boldsymbol{K}_k, \boldsymbol{K}_g$	Number of structural keys; client and server key banks.
$\boldsymbol{k}_{k,p}, \boldsymbol{h}_{k,i}$	Client key p and base node embedding.
$q_{k,i,p}, p_{k,i,p}$	Node-to-key assignment probability and its sharpened training target.
$\boldsymbol{q}_{k,i}, \boldsymbol{s}_k^{t,r}$	Node-to-key assignment vector and client structural state.
$\boldsymbol{\theta}_k, \boldsymbol{\theta}_g$	Client and global GNN parameters.
$\omega_k, \alpha_{k,u}$	Client aggregation weight and reference-node teacher weight.
$\boldsymbol{\rho}_{g,u}, \boldsymbol{\rho}_{k,u}$	Global and client predictive distributions on reference node u.
$B, \mathcal{B}_k^t$	Memory capacity and client k's memory after task t.
$(\boldsymbol{k}_j, \boldsymbol{v}_j, m_j, \boldsymbol{z}_j)$	Memory-slot key, summed value, mass, and semantic target.
$K_h, \mathcal{N}_i^{K_h}$	Retrieval count and selected memory slots for node i.
τ_h, σ	Retrieval temperature and scale of the memory readout.
$\widehat{\boldsymbol{h}}_{k,i}$	History-enhanced node embedding.
$\mathcal{J}_{\mathrm{old}}, \gamma$	Old-class memory slots and old–new logit margin.
$\mathcal{L}_{\mathrm{state}}, \mathcal{L}_{\mathrm{sem}}$	Structural-state and semantic-consistency losses.
$\mathcal{L}_{\mathrm{margin}}, \mathcal{L}_{\mathrm{enc}}$	Old–new class-boundary and encoder-stability losses.
$\mathcal{L}_{\mathrm{sep}}$	Structural-key separation loss.
$\mathcal{L}_{\mathrm{cur}}, \mathcal{L}_{\mathrm{local}}$	Current-task and full client-side training objectives.

B ADDITIONAL EXPERIMENTAL DETAILS

Task construction and label access. The MemEvo-FCGL runs define the reference task construction. Each client receives disjoint blocks of at most two new classes per task. The train/validation/test split is $20\%/40\%/40\%$ except on Squirrel, where it is $48\%/32\%/20\%$. Features and induced connections of all task-visible nodes are available for transductive message passing; only training-node labels serve as instance-level training targets or enter memory construction. Predefined task class sets specify the shared output partitions used by the old–new margin, not the labels of individual validation or test nodes.

Implementation and reporting. MemEvo-FCGL uses a two-layer GAT (Veličković et al., 2018) with hidden dimension 64 and dropout 0.5, trained with Adam (Kingma & Ba, 2015) at learning rate 0.01 and weight decay 5×10^{-4}. All three clients participate in every communication round. Table 4 reports the dataset-specific task counts, memory settings, and training budgets. The retrieval temperature τ_h and interaction scale σ are fixed at 0.05 and 0.5, respectively. We report the mean and standard deviation over five saved runs, using seeds $\{77, 2034, 2024, 1024, 33\}$ except on Cora, which uses $\{1024, 2034, 2023, 3407, 2024\}$. Federation-level task accuracy is weighted by each client's test-node count. Signed FM is computed from the saved task-accuracy matrices without clipping negative forgetting values.

Comparison scope. The main results include all ten baselines on eight datasets. For datasets marked "Matched" in Table 4, FedATH, Fed-ERGNN, Fed-TWP, FedAGC, Powder, and POWER share the MemEvo-FCGL client count, class increment, task count, backbone, hidden dimension, and training budget. The Cora baselines instead use four tasks and a different seed set, while the CiteSeer runs differ in training budget. FedAvg, FedDyn, FedSSP, and MOTION are not uniformly matched across the remaining six datasets. Method-specific optimization and memory hyperparameters may differ even within the matched subset. We therefore interpret table-wide rankings as recorded outcomes and restrict comparisons under a common protocol to the matched subset.

Table 4: Public full-graph statistics and saved MemEvo-FCGL configurations. Edges count unique undirected connections, excluding self-loops, before client and task partitioning; Chameleon and Squirrel use the original, unfiltered Geom-GCN versions. T: tasks; P: structural keys per client; B: abstract-memory slots per client; K_h: retrieval top-K_h; R: rounds per task; E: local epochs per round. Comparison refers to the six-baseline subset defined in the text.

Dataset	Nodes	Features	Edges	T	P	B	K_h	$R \times E$	Comparison
Cora	2,708	1,433	5,278	3	20	40	3	15×5	Task/seed mismatch
CiteSeer	3,327	3,703	4,552	3	10	20	3	10×3	Budget mismatch
PubMed	19,717	500	44,324	2	20	40	3	15×5	Matched
Computers	13,752	767	245,861	5	20	40	3	15×5	Matched
Photo	7,650	745	119,081	4	20	40	3	15×5	Matched
Coauthor CS	18,333	6,805	81,894	8	20	40	3	15×5	Matched
Chameleon	2,277	2,325	31,371	3	20	40	3	15×5	Matched
Squirrel	5,201	2,089	198,353	3	20	40	3	15×5	Matched

C Baseline Methods

FL and FGL. FedAvg weights client parameters by sample count (McMahan et al., 2017); FedDyn uses dynamic regularization to reduce client drift (Acar et al., 2021). FedSSP shares generic spectral knowledge while retaining client-specific graph preferences (Tan et al., 2024). FedATH separates causal from biased topology to address structural heterogeneity (Fu et al., 2025).

Continual-learning adaptations. Fed-ERGNN combines FedAvg with coverage-based experience-node replay from ER-GNN (Zhou & Cao, 2021). Fed-TWP combines FedAvg with topology-aware parameter preservation from TWP (Liu et al., 2021). Powder adapts prompt-based dual knowledge transfer (Piao et al., 2024), while FedAGC combines representative rehearsal with asymmetric gradient correction (Zhang et al., 2025).

FCGL. POWER uses experience-node replay, pseudo-prototype reconstruction, and knowledge transfer (Zhu et al., 2025). MOTION combines topology-preserving graph coarsening with conflict-aware model aggregation (Wan et al., 2025).

D Training and Memory Update Algorithms

This section presents the training and memory update procedures of MemEvo-FCGL, complementing the formulation in Section 4. The two algorithms describe the overall federated workflow and the client-side consolidation of a bounded memory.

Training procedure. Algorithm 1 organizes local learning and server coordination over tasks and communication rounds. Clients train with historical memory, while the server aggregates the shared GNN and structural keys and refines the GNN on reconstructed reference nodes. Each task ends with local memory consolidation and parameter-anchor updates.

Algorithm 1 Training and task-boundary memory evolution in MemEvo-FCGL

Require: Task graphs $\{\mathcal{G}_k^t\}$ and training-node masks; initial global GNN and keys $(\boldsymbol{\theta}_g, \boldsymbol{K}_g)$; task count T, rounds R, slot limit B.

Algorithm 1 Training and task-boundary memory evolution in MemEvo-FCGL *(continued)*

Ensure: Trained $\boldsymbol{\theta}_g$ and client memories $\{\mathcal{B}_k^T\}$.

for $t = 1, \ldots, T$ **do**
  for each client k **do**
    Initialize the historical read layer from $\mathcal{B}_k^{t-1}$ if $t > 1$.
  end for
  for $r = 1, \ldots, R$ **do**
    Server: broadcast $(\boldsymbol{\theta}_g, \boldsymbol{K}_g)$.
    for each client k **in parallel do**
      Encode $\mathcal{G}_k^t$; retrieve historical values if available (Eq. equation 11).
      Update the GNN and keys with Eq. equation 14 using training-node labels only.
      Compute $\boldsymbol{s}_k^{t,r}$ from base embeddings of all task-visible nodes (Eq. equation 4).
      Upload $(\boldsymbol{\theta}_k, \boldsymbol{K}_k, \boldsymbol{s}_k^{t,r})$; at $r = 1$, also upload training-derived prototype gradients.
    end for
    Server: compute ω_k and aggregate GNN and keys (Eq. equation 5).
    if $r = 1$ **then**
      Reconstruct reference nodes from prototype gradients.
    end if
    Compute teacher weights (Eq. equation 6) and refine the global GNN on the reference graph.
  end for
  for each client k **do**
    Update encoder importance; form class-balanced candidate slots from training nodes (Eq. equation 7).
    Merge compatible candidates with $\mathcal{B}_k^{t-1}$.
    if the candidate count exceeds B **then**
      Recluster slots in joint key–semantic space (Eq. equation 10).
    end if
    Save $\mathcal{B}_k^t$ and update task-boundary parameter anchors.
  end for
end for

Bounded memory update. Algorithm 2 expands the task-boundary consolidation into candidate generation, one-to-one merging, insertion of unmatched slots, and compression when the combined bank exceeds capacity. For client k after task t, the notation follows Section 4.2. Write $\xi_j = (\boldsymbol{k}_j, \boldsymbol{v}_j, m_j, \boldsymbol{z}_j)$ for a historical slot and ξ_p^{cur} for a candidate with $\boldsymbol{k}_p^{\mathrm{cur}} = \boldsymbol{k}_{k,p}$. The update uses training-node labels only and is performed without backpropagation.

Algorithm 2 Bounded key–value memory update

Require: Previous memory $\mathcal{B}_k^{t-1}$; learned keys $\{\boldsymbol{k}_{k,p}\}_{p=1}^P$; training-node embeddings and labels $\{(\boldsymbol{h}_{k,i}, y_i) : i \in \mathcal{V}_{k,\mathrm{tr}}^t\}$; historical read layer; capacity B; parameters $(\tau_s, \delta_{\mathrm{key}}, \delta_z, \mu, \lambda_z)$.

Ensure: Updated memory $\mathcal{B}_k^t$ with $|\mathcal{B}_k^t| \le B$.

1. Generate candidate slots.

Compute $q_{k,i,p}$ from base embeddings (Eq. equation 2); obtain $\widehat{\boldsymbol{h}}_{k,i}$ by historical retrieval if available (Eq. equation 11), otherwise set $\widehat{\boldsymbol{h}}_{k,i} = \boldsymbol{h}_{k,i}$.
Allocate the P keys nearly uniformly across observed training classes using class-wise assignment strengths; obtain owners $o(p)$.
Form candidates $\{\xi_p^{\mathrm{cur}}\}_{p=1}^P$ from training nodes (Eq. equation 7).
if $|\mathcal{B}_k^{t-1}| = 0$ **then**
  return the $\min(B, P)$ candidates with largest masses as $\mathcal{B}_k^t$.
end if

2. Match and merge.

Copy $\mathcal{B} \leftarrow \mathcal{B}_k^{t-1}$; mark all candidate and historical slots unmatched.
Before any updates, identify pairs (p, j) satisfying $\cos(\boldsymbol{k}_p^{\mathrm{cur}}, \boldsymbol{k}_j) \ge \delta_{\mathrm{key}}$ and $\cos(\boldsymbol{z}_p^{\mathrm{cur}}, \boldsymbol{z}_j) \ge \delta_z$.
for each eligible pair (p, j) in descending key cosine similarity **do**
  if both slots are unmatched **then**
    Merge the candidate into historical slot j by EMA (Eq. equation 8); normalize $\boldsymbol{z}_j$ to unit sum and mark both slots matched.
  end if
end for

3. Append new slots.

Append all unmatched candidates to $\mathcal{B}$, retaining unmatched historical slots.

4. Compress if over capacity.

if $|\mathcal{B}| > B$ **then**

Algorithm 2 Bounded key–value memory update *(continued)*

16: Form joint key–semantic representations $\mathbf{r}_j$ (Eq. equation 9).
17: Cluster all slots into B groups $\{\mathcal{C}_b\}_{b=1}^{B}$ using Euclidean k-means: within-class clustering with near-uniform class quotas if the represented class count is at most B; global clustering otherwise.
18: Replace each cluster with its aggregated slot (Eq. equation 10); normalize each semantic target to unit sum.
19: **end if**
20: **return** $\mathcal{B}_k^t \leftarrow \mathcal{B}$.

Candidate allocation proceeds round-robin over training classes, each choosing the remaining key with the largest class-wise assignment mass. For compression, slot classes are determined by $\arg\max_c z_{j,c}$; cluster quotas are capped by class slot counts, and unused quotas are redistributed to classes with remaining capacity. The implementation uses $0 \leq \mu < 1$; all reported settings satisfy $P \leq B$.