MoCE: Mixture of Communitified Experts for Continual Vision-Language Learning under Task-Complexity Flux

Anonymous authors
Paper under double-blind review

Abstract

Class-incremental learning (CIL) with vision-language models is often formulated as a balanced sequence of class increments, but deployed systems rarely receive such regular updates. Some increments contain only a few closely related classes, while others introduce many visually diverse or semantically distant classes. We refer to this variation as **uneven class-incremental complexity**. It exposes a weakness of standard sparse Mixture-of-Experts (MoE) adapters: a fixed top-k router activates the same number of isolated experts for every input and every increment, even when the amount of expert knowledge needed by the current update changes. We propose **MoCE**, a **Mixture of Communitified Experts** for CLIP-based class-incremental learning. MoCE decouples the number of *executed* experts from the number of *involved* experts. It represents adapters as nodes in an input-conditioned learnable graph, lets each node exchange messages with its neighbors, and then selects sparse *community anchors*. A selected expert, also named as anchor, is therefore not a single isolated expert; it is the representative of a graph community whose state has already absorbed information from related experts. As a result, MoCE can execute a small anchor set for efficiency while involving a variable number of neighboring experts in routing, and it can optionally increase the anchor budget for increments with more classes. This gives the router a flexible capacity response to uneven task complexity without dense expert execution. Experiments under even and uneven CIL schedules show that MoCE improves over standard MoE adapters, with especially consistent gains when the number of arriving classes varies across increments.

1 Introduction

Vision-language models (VLMs), especially CLIP-like models, learn transferable visual representations by aligning images and texts at large scale Radford et al. (2021); Zhang et al. (2024). Their zero-shot recognition ability makes them attractive for open-world applications such as visual question answering, image captioning, and cross-modal retrieval Agrawal et al. (2016); Ye et al. (2024); Penamakuri et al. (2023). However, deployed recognition systems cannot remain static: new product categories, medical labels, safety events, or user-defined concepts may arrive after deployment, and the model must learn them without losing previous knowledge. Naive fine-tuning can damage pretrained image-text alignment and is expensive for large VLMs French (1999); Ding et al. (2024); Han et al. (2024).

We study class-incremental learning (CIL), where a learner receives a sequence of class increments $\{\mathcal{D}_t\}_{t=1}^{T}$ and must recognize all classes seen so far Zhou et al. (2024). Most CIL studies assume balanced streams, where each increment contains the same number of classes and is implicitly treated as having similar complexity. This assumption is convenient but unrealistic: in practice, one update may contain only a few classes, while another may contain many semantically diverse classes. We refer to this setting as *uneven class-incremental complexity*, where different increments require different amounts of adaptation capacity and cross-expert reuse.

Parameter-efficient fine-tuning (PEFT) is a natural tool for continual VLM adaptation because it freezes most pretrained parameters and learns lightweight modules such as adapters or low-rank

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.

updates Houlsby et al. (2019); Hu et al. (2022); Tang et al. (2024). Sparse Mixture-of-Experts (MoE) adapters further increase capacity by maintaining multiple experts while activating only a small subset for each input Shazeer et al. (2017); Fedus et al. (2022); Yu et al. (2024; 2025). However, standard MoE routers usually make a rigid top-k decision: the experts that are executed are also the only experts that can influence the routing decision. This is poorly matched to uneven increments, where simple increments may need compact expert evidence, while complex increments may require broader expert-community support.

The key idea of this paper is that a continual MoE should not only ask *which experts should be executed?* It should also ask *which expert communities should be involved in the routing decision?* We propose **MoCE**, a **Mixture of Communitified Experts**. In MoCE, each adapter expert is represented as a node in a learnable graph. An input-conditioned adjacency head predicts expert communication patterns, graph message passing updates expert states, and the resulting graph states produce community-aware routing logits. Top-k routing is then performed on these logits, so selected experts act as *community anchors* rather than isolated endpoints.

MoCE enables flexible expert involvement while preserving sparse execution. Even when the number of executed experts is fixed, selected anchors carry information from their graph neighborhoods; thus, the involved expert set can expand or contract according to the learned adjacency and overlap among communities. When the number of new classes in the current increment is available, MoCE can further use a task-complexity-adaptive anchor budget, assigning more anchors to complex increments and fewer to simpler ones. Concretely, the base MoE router produces logits Z_{base}, while the graph router produces per-expert graph logits Z_{graph}. The final router uses residual logit mixing

$$Z = Z_{\mathrm{base}} + \lambda Z_{\mathrm{graph}}$$

which exactly recovers the standard MoE router when $\lambda = 0$. Thus, MoCE is a conservative extension of sparse MoE: it preserves the original router while adding graph-based community evidence for uneven class-incremental learning.

Our contributions are fourfold: (a) We formulate *uneven class-incremental complexity* as a practical stress case for sparse MoE-based VLM continual learning. (b) We propose MoCE, a community-anchored MoE framework that decouples executed experts from involved experts. (c) We provide a reproducible mathematical specification of MoCE, including input-conditioned expert nodes, learnable adjacency, top-m neighborhoods, residual graph routing, flexible involved expert sets, and optional task-adaptive anchor budgets. (d) We evaluate MoCE on even and uneven CIL schedules and analyze how graph-based expert communities improve routing flexibility for continual VLM adaptation.

2 Related Work

2.1 Continual Learning for Vision and Vision-Language Models

Continual learning seeks to update a model over a stream of tasks without joint retraining on all previous data. Classical approaches include regularization methods that protect important parameters or distill old predictions, rehearsal methods that store or synthesize past examples, and expansion methods that allocate new capacity over time Kirkpatrick et al. (2017); Li & Hoiem (2018); Rebuffi et al. (2017); Douillard et al. (2022); Zhou et al. (2024). In class-incremental learning, the difficulty is stronger because the model must make a single prediction over all seen classes rather than relying on task identity at test time.

Continual learning for VLMs adds another constraint: adaptation should preserve the pretrained image-text alignment. Recent methods such as C-CLIP and CLAP4CLIP study how to retain CLIP knowledge while learning incrementally Liu et al. (2025); Jha et al. (2024). Other work analyzes interference in parameter-efficient continual learning of VLMs Tang et al. (2024). These methods focus primarily on preventing forgetting and retaining pretrained knowledge. MoCE is complementary: it targets the allocation problem that appears when the adaptation modules are sparse experts and the stream has uneven complexity.

2.2 Parameter-Efficient Fine-Tuning and Sparse Mixture of Experts

PEFT methods adapt large pretrained models by updating a small number of parameters, such as adapter modules or low-rank matrices (Houlsby et al., 2019; Hu et al., 2022; Han et al., 2024). This is attractive for CIL because frozen pretrained weights provide a stable representation while lightweight modules absorb new knowledge.

Sparse MoE layers use conditional computation: many experts are available, but each input is routed to only a few of them (Shazeer et al., 2017; Fedus et al., 2022). MoE-Adapters and MoE-Adapters++ bring this idea to continual VLM adaptation by combining PEFT experts with learned routers (Yu et al., 2024; 2025). Their main benefit is efficient capacity expansion, but standard routers usually treat experts as independent choices and use a fixed top-k execution pattern.

Existing VLM continual-learning methods mainly preserve pretrained knowledge, while sparse MoE adapters expand capacity through fixed top-k expert selection. In contrast, MoCE introduces graph-community routing, where selected experts remain sparse execution anchors but incorporate evidence from learned expert neighborhoods.

2.3 Graph Neural Networks for Expert Communication

Graph neural networks update node representations through neighborhood message passing Kipf & Welling (2017); Veličković et al. (2018). This is useful when the entities being modeled have latent relations that should influence decisions. In MoCE, the entities are adapter experts. The graph does not replace the MoE layer and does not require dense execution of all experts. Instead, it is used before routing: expert node states communicate, and the router selects anchors whose states summarize local graph communities.

This distinction is important for uneven CIL. A graph community can contain a different number of related experts depending on the input and the current adjacency. Therefore, the effective set of experts involved in routing can be larger than the set of executed experts and can vary across examples or tasks. Existing VLM continual-learning methods mainly ask how to preserve old knowledge, while existing MoE-adapter methods mainly ask how to select a fixed number of experts efficiently. MoCE asks a different question: how can the router vary the amount of expert evidence it consults as the complexity of the class increment changes?

3 Methodology

3.1 Design Principle: Sparse Execution, Flexible Evidence

Sparse MoE is attractive for continual VLM adaptation because it adds capacity without executing all modules for every input Shazeer et al. (2017); Fedus et al. (2022); Yu et al. (2024; 2025). Its weakness under uneven increments is more subtle: the same top-k mechanism controls both *computation* and *evidence*. Once the router selects k experts, only these k isolated modules influence the prediction. This coupling is reasonable when every increment has similar complexity, but it is restrictive when one increment contains a few coherent classes and another contains many diverse classes.

MoCE separates the experts that contribute information to routing from the experts that are actually executed. For each input, MoCE constructs a graph in which each node represents an expert and the learned edge weights indicate which experts should exchange routing information. For every expert, the graph retains its m most strongly connected neighbors. We refer to an expert together with these neighbors as its community. The neighboring experts exchange lightweight graph-state information before routing, but their full adapter modules are not necessarily executed. After this information exchange, the router selects k experts as anchors. Only these selected anchors process the input through their adapter modules. Thus, an anchor's routing score reflects information from several related experts, while actual expert execution remains sparse.

CIL setup and backbone. We consider a class-incremental stream $\{\mathcal{D}_t\}_{t=1}^{T}$. Increment t introduces new label set $\mathcal{Y}_t$ with class count $C_t = |\mathcal{Y}_t|$. Balanced protocols make C_t constant. In our setting C_t and semantic diversity can vary, so C_t is used as a simple observable signal of task com-

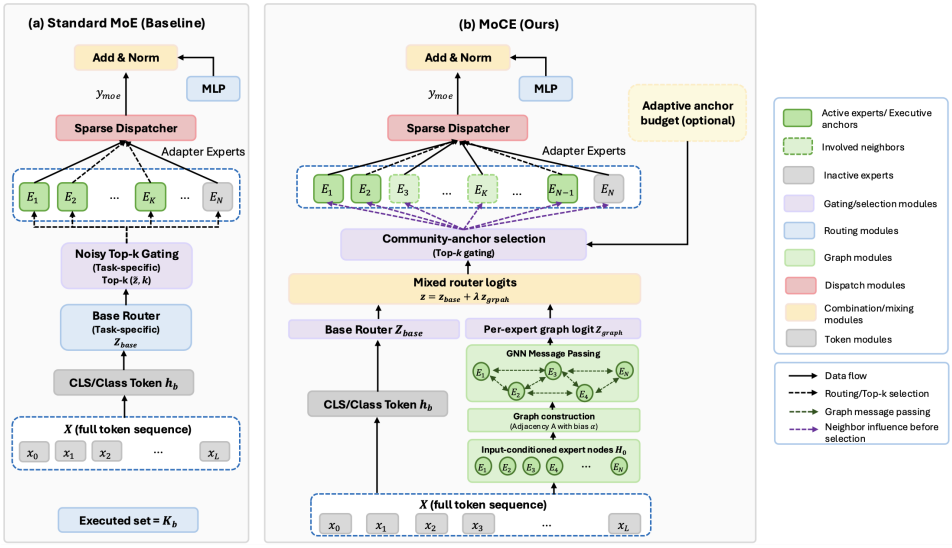


Figure 1: Architecture comparison between a standard MoE adapter and MoCE. Standard MoE selects a fixed top-k set of isolated experts. MoCE builds an input-conditioned expert graph, lets each expert state communicate with neighbors, and then selects sparse community anchors using $z = z^{\text{base}} + \lambda z^{\text{graph}}$. The executed set remains sparse, while the involved routing context can include a variable number of graph-neighbor experts.

plexity. The visual backbone is a frozen CLIP encoder Radford et al. (2021). Trainable adapter experts are inserted into transformer blocks. For a block representation $X \in \mathbb{R}^{L\times B\times D}$, where L is sequence length, B is batch size, and D is hidden dimension, the class-token feature for sample b is $h_b = X_{0,b,:} \in \mathbb{R}^D$. We use h_b rather than a task identifier because class-incremental evaluation must make a single prediction over all seen classes without being told which task generated the test image.

The adapter layer contains N experts $\{E_i\}_{i=1}^N$. We distinguish two sets throughout the method. The *executed set* contains experts whose full adapter modules process the token sequence.Full adapter modules is the actual expert adapter function $E_i(\cdot)$, not the lightweight graph-node representation used for routing. The *involved set* contains experts whose lightweight graph states contribute evidence to routing. Standard sparse MoE makes these sets essentially the same. MoCE deliberately makes the involved set more flexible than the executed set. As summarized by Figure 1, a standard router selects top-k independent experts from h_b. MoCE first builds an input-conditioned graph over expert nodes, performs message passing, and then selects sparse community anchors from graph-enhanced logits. The expensive part remains sparse adapter execution; the additional graph branch is a lightweight routing computation over expert states.

3.2 Standard Sparse MoE as the Limiting Case

A sparse MoE adapter computes base routing logits from the class-token representation:

$$z_b^{\text{base}} = h_b W_g + b_g,\ W_g \in \mathbb{R}^{D\times N},\ b_g \in \mathbb{R}^N$$

The router then selects a top-k subset of experts for sparse execution. This design is computationally efficient because only k adapters are activated for each input. However, the same k experts are also the only experts whose specialized knowledge can influence the routing decision. For uneven class increments, this fixed evidence budget becomes restrictive: increasing k can help complex increments but wastes computation on simple ones, while keeping k small preserves efficiency but suppresses useful cross-expert context. MoCE treats standard MoE as a limiting case and relaxes this bottleneck by allowing selected experts to act as graph-community anchors rather than isolated routing endpoints.

3.3 MoCE: Routing through Expert Communities

MoCE augments the standard sparse router with a graph-based routing branch before adapter execution. This branch operates on one low-dimensional state per expert rather than evaluating every expert adapter on the full token sequence. The graph therefore allows expert states to exchange information before the top-k selection is made, while the computationally expensive adapter operations remain restricted to the selected experts.

Input-conditioned expert nodes. A graph over experts is useful only if its nodes know both the current input and their own expert identity. We therefore initialize node i for sample b as $H_{b,i}^{(0)} = Ph_b + e_i,\ P \in \mathbb{R}^{H\times D},\ e_i \in \mathbb{R}^H$. The projected input Ph_b tells every node what must be recognized; the embedding e_i tells the graph which expert specialization is being considered. This construction is important. If the graph states were only expert embeddings, the graph router could collapse into a global expert prior shared by all samples. Because $H_{b,i}^{(0)}$ depends on h_b, the graph evidence is input-dependent; because it also contains e_i, it remains expert-specific.

Input-conditioned expert graph. MoCE predicts pairwise relationships among experts from the current input representation: $S_b = \mathrm{reshape}(\psi_A(h_b), N, N)$, where $S_{b,i,j}$ is the unnormalized compatibility between experts i and j for sample b. In our implementation, we symmetrize these pairwise compatibility logits before normalization: $S_b \leftarrow \frac{S_b + S_b^\top}{2}$. This operation assigns the same unnormalized pairwise compatibility to (i, j) and (j, i). The resulting logits are then normalized independently within each row:

$$A^0_{b,i,j} = \frac{\exp(S_{b,i,j})}{\sum_{r=1}^{N} \exp(S_{b,i,r})}$$

Thus, $A^0_{b,i,j}$ represents the relative strength with which expert j contributes information to the graph state of expert i for sample b. Because the normalization is row-wise, $A^0_{b,i,j}$ and $A^0_{b,j,i}$ are not required to be equal even though the corresponding unnormalized logits are symmetrized.

Graph message passing. Given A_b, MoCE updates the expert-node representations using L_g message-passing layers:

$$H_b^{(\ell+1)} = \phi\Big(\mathrm{LN}\Big(A_b H_b^{(\ell)} W_\ell + \mathbf{1} b_\ell^\top\Big)\Big)$$

with optional residual connections between graph layers. Because A_b is sparse, the updated state of expert i is determined primarily by its own state and the states of the retained neighboring experts in $\mathcal{N}_b(i)$. Let

$$Y = H^{(L_g)} \in \mathbb{R}^{B\times N\times H}$$

The vector $Y_{b,i,:}$ is therefore the graph-updated representation of expert i for sample b. These representations are used only to compute routing scores; they are not the outputs of the adapter experts E_i.

Per-expert graph logits. The graph state is converted into routing evidence separately for each expert: $z^{\mathrm{graph}}_{b,i} = \langle Y_{b,i,:}, w_i^{\mathrm{goe}}\rangle,\ W^{\mathrm{goe}} = [w_1^{\mathrm{goe}}; \ldots; w_N^{\mathrm{goe}}] \in \mathbb{R}^{N\times H}$. The expert-specific projection matters for identifiability. A single shared scalar head could behave like a generic graph bias. It instead asks whether the *community state of expert* i supports choosing expert i as an anchor.

Residual graph routing. MoCE does not discard the strong MoE router. It augments it through residual logit mixing: $z_b = z_b^{\mathrm{base}} + \lambda z_b^{\mathrm{graph}}$. This residual form makes the relationship to existing MoE adapters explicit. Setting $\lambda = 0$ exactly recovers $z_b^{\mathrm{base}} = h_b W_g + b_g,\ W_g \in \mathbb{R}^{D\times N},\ b_g \in \mathbb{R}^N$. Positive λ lets graph communities correct or reinforce the independent router when the input requires broader expert context. Thus MoCE is a conservative extension rather than a replacement of sparse MoE.

3.4 Flexible Expert Involvement

The mixed logits select community anchors, i.e., $\mathcal{K}_b = \mathrm{TopK}(z_b, k_b)$, where k_b can be fixed or stage-adaptive. The executed set is still $\mathcal{K}_b$, so inference remains sparse. The involved routing set, however, is the union of the selected anchors and the graph neighbors whose messages were used by those anchors:

$$\mathcal{U}_b = \bigcup_{i \in \mathcal{K}_b} \left(\{i\} \cup \mathcal{N}_b(i) \right), \ \mathcal{N}_b(i) = \{ j : M_{b,i,j} = 1 \}$$

The effective number of experts involved in routing is therefore $n_b^{\mathrm{inv}} = |\mathcal{U}_b|, \ k_b \le n_b^{\mathrm{inv}} \le \min(N, k_b(m+1))$ This inequality is the core capacity argument. Standard MoE has $n_b^{\mathrm{inv}} = k_b$: every executed expert is an isolated decision. MoCE has a variable n_b^{inv}. If the selected anchors share neighbors, the involved set is compact; if they occupy different graph regions, the involved set expands. Therefore the model can use a small expert community for simple increments and a broader collection of communities for complex increments, while still executing only the anchors.

When the stream provides the class count of the current increment, MoCE can also adapt the anchor budget itself:

$$k_t = \min \left(N, \max \left(1, k_0 + \left\lfloor \frac{C_t - C_0}{\Delta} \right\rfloor \right) \right)$$

Here k_0 is the base anchor budget, C_0 is a reference increment size, and Δ controls how many additional classes justify one more anchor. This stage-level rule uses only stream metadata available during incremental training; it does not provide the test-time task label of an individual image. If adaptive anchors are disabled, $k_t = k_0$ and flexibility still comes from $n_b^{\mathrm{inv}} = |\mathcal{U}_b|, \ k_b \le n_b^{\mathrm{inv}} \le \min(N, k_b(m+1))$. If enabled, complex increments can both execute more anchors and involve wider graph communities.

With anchors selected, MoCE uses the same sparse adapter aggregation as standard MoE,

$$Y^{\mathrm{moe}}_{:,b,:} = \sum_{i \in \mathcal{K}_b} g_{b,i} E_i(X_{:,b,:})$$

and the transformer block output is

$$X^{\mathrm{out}} = X + \mathrm{MLP}(\mathrm{LN}_2(X)) + Y^{\mathrm{moe}}$$

The graph branch therefore changes *who is consulted* before routing, not the sparse-execution contract of the adapter layer.

3.5 Training Objective and Algorithm

MoCE is trained with the classification objective of the underlying CIL framework. As in sparse MoE, we retain a load-balancing term to prevent expert collapse and a router-logit regularizer to stabilize gating. The loss is

$$\mathcal{L} = \mathcal{L}_{\mathrm{ce}} + \beta \mathcal{L}_{\mathrm{lb}}(g, \ell) + \gamma \mathcal{L}_z(z)$$

where g are sparse gates, ℓ denotes expert load, and z are the final mixed router logits. The load-balancing term is computed from the final routing decisions, because those decisions include both base and graph evidence.

4 Experiments

4.1 Experimental Goals and Protocol

The experiments address four questions: whether MoCE improves standard sparse MoE under uneven class increments, whether the gain remains under balanced streams, how MoCE compares with representative continual-learning baselines such as fine-tuning, LwF (Li & Hoiem, 2018), LwF-VR (Ding et al., 2022), iCaRL (Rebuffi et al., 2017), ZSCL (Zheng et al., 2023), and MoE-Adapters (Yu et al., 2024; 2025), and how sensitive the graph router is to λ and α. The key comparison is the controlled MoE-versus-MoCE setting, where the backbone, optimizer, data order, training duration, and expert pool are fixed. MoE disables the graph mixer and adaptive community routing,

Table 1: Results under balanced class-incremental schedules on CIFAR100 and TinyImageNet-100 with $N \in \{8, 16\}$ experts. Entries report mean ± standard deviation of accuracy (%) over three runs.

	CIFAR100				**TinyImageNet-100**			
	MoE		**MoCE**		**MoE**		**MoCE**	
Steps	**Last**	**Average**	**Last**	**Average**	**Last**	**Average**	**Last**	**Average**
***N* = 8**								
10	77.0 ± 0.2	85.1 ± 0.1	**78.3** ± 3.1	**87.3** ± 1.7	78.7 ± 0.4	86.6 ± 0.1	**79.2** ± 2.9	**88.2** ± 1.2
20	79.8 ± 0.1	86.8 ± 0.1	**84.6** ± 2.0	**90.5** ± 1.6	82.0 ± 0.4	87.7 ± 0.0	**86.5** ± 0.7	**90.9** ± 0.3
50	84.9 ± 0.3	88.4 ± 0.3	**89.9** ± 0.5	**91.2** ± 0.3	85.2 ± 0.3	88.8 ± 0.2	**87.8** ± 0.4	**90.1** ± 0.2
***N* = 16**								
10	75.2 ± 0.1	84.3 ± 0.1	**78.6** ± 1.1	**88.1** ± 0.6	78.0 ± 0.1	85.9 ± 0.1	**79.0** ± 1.5	**88.0** ± 1.0
20	79.1 ± 0.2	86.3 ± 0.2	**82.8** ± 1.6	**89.3** ± 0.7	81.7 ± 0.4	87.5 ± 0.1	**84.5** ± 0.3	**90.2** ± 0.4
50	84.7 ± 0.4	87.9 ± 0.2	**87.2** ± 1.5	**89.7** ± 0.8	85.4 ± 0.2	88.8 ± 0.3	**86.4** ± 0.0	**89.4** ± 0.3

while MoCE lets selected experts act as graph-community anchors and, for uneven streams, adjusts the anchor budget according to the number of arriving classes. External baselines provide context, but the controlled comparison most directly isolates the proposed routing policy.

4.2 Datasets, Splits, and Evaluation Metrics

We evaluate MoCE on CIFAR100 and TinyImageNet-100 under both uneven and balanced class-incremental learning protocols. At increment t, the model receives examples from a new class set $\mathcal{Y}_t$ and is evaluated over all classes observed so far, $\cup_{s \le t} \mathcal{Y}_s$. The uneven protocols use 10-step streams with non-uniform class counts, while the balanced protocols use 10-, 20-, and 50-step streams with equal class counts per increment. Let A_t denote the top-1 accuracy after finishing increment t, evaluated on all seen classes. We report final accuracy after the last increment, $\mathrm{Last} = A_T$, and average accuracy over the learning trajectory, $\mathrm{Avg} = \frac{1}{T} \sum_{t=1}^{T} A_t$. For controlled MoE-versus-MoCE comparisons, the dataset, class order, backbone, prompt, optimizer, training duration, and total expert pool are fixed. MoCE differs from MoE only in the proposed graph-community routing branch.

4.3 Balanced Task Results

Table 1 reports results under balanced class-incremental schedules on CIFAR100 and TinyImageNet-100 with $N \in 8, 16$ experts. MoCE consistently outperforms the standard MoE baseline in both Last and Average accuracy across all 10-, 20-, and 50-step settings. On CIFAR100, Last-accuracy gains range from $+1.3$ to $+5.0$ points for $N = 8$ and from $+2.5$ to $+3.7$ points for $N = 16$, with corresponding Average-accuracy improvements of up to $+3.7$ and $+3.8$ points. Similar gains are observed on TinyImageNet-100. For $N = 8$, MoCE improves Last accuracy by $+0.5$, $+4.5$, and $+2.6$ percentage points across the 10-, 20-, and 50-step schedules, respectively, together with Average-accuracy gains of $+1.6$, $+3.2$, and $+1.3$ points. For $N = 16$, Last accuracy increases by $+1.0$, $+2.8$, and $+1.0$ points, while Average accuracy improves by $+2.1$, $+2.7$, and $+0.6$ points. These consistent improvements across datasets, task granularities, and expert counts show that graph-community routing remains beneficial even under evenly partitioned class-incremental streams.

4.4 Controlled Uneven-Task Results

Table 2 summarizes results averaged separately over the 10-step (U1–U4) and 20-step (U5–U8) uneven schedules with $N \in \{8, 16\}$ experts. MoCE outperforms standard MoE in both Last and Average accuracy across both datasets, schedule groups, and expert counts. On CIFAR100, Last-accuracy gains range from $+6.25$ to $+6.52$ percentage points under the 10-step schedules and from

Table 2: Average and last accuracy (%) across U1-U8 uneven schedules.

	CIFAR100				TinyImageNet			
	MoE		MoCE		MoE		MoCE	
Setting	L	A	L	A	L	A	L	A
10 steps (U1–U4)								
$N = 8$	76.33	85.40	**82.85**	**90.05**	79.95	86.73	**83.93**	**90.20**
$N = 16$	75.00	84.50	**81.25**	**89.30**	79.08	86.10	**82.40**	**89.45**
20 steps (U5–U8)								
$N = 8$	80.28	86.80	**86.10**	**90.40**	82.03	87.53	**85.95**	**90.10**
$N = 16$	79.33	86.13	**84.50**	**89.78**	81.68	87.20	**84.65**	**89.38**

L = Last; A = Avg.

Table 3: Average and last accuracy (%) on the descending U4 schedule.

Method	CIFAR100		TinyImageNet	
	L	A	L	A
Fine-tune	77.91	81.83	80.02	83.40
Fine-tune + CE-ext	75.79	81.28	79.62	83.38
LwF (Li & Hoiem, 2018)	65.81	73.50	70.82	77.83
LwF-VR (Ding et al., 2022)	64.31	73.71	67.26	76.58
ZSCL (Zheng et al., 2023)	79.01	84.25	80.58	84.88
iCaRL (Rebuffi et al., 2017)	62.30	72.62	67.76	77.75
MoE-Adapters (Yu et al., 2024; 2025)	75.16	82.21	77.90	84.10
MoCE	**81.53**	**87.50**	**82.35**	**87.23**

L = Last; A = Avg. CE-ext denotes classifier expansion; LwF-VR denotes vocabulary replay.

+5.17 to +5.82 points under the 20-step schedules. On TinyImageNet-100, the corresponding gains range from +3.32 to +3.98 and from +2.97 to +3.92 points, respectively. Average-accuracy gains range from +3.60 to +4.80 points on CIFAR100 and from +2.18 to +3.47 points on TinyImageNet-100. These results support the effectiveness of graph-community routing under uneven class arrivals across both schedule lengths.

4.5 Contextual Comparison with Continual-Learning Baselines

Table 3 reports contextual results under the descending U4 schedule, where class increments are highly non-uniform. MoCE achieves the best performance on both datasets, improving over ZSCL by +2.52 Last and +3.25 Avg points on CIFAR100, and by +1.77 Last and +2.35 Avg points on TinyImageNet-100. Compared with MoE-Adapters, MoCE gains +6.37 Last and +5.29 Avg points on CIFAR100, and +4.45 Last and +3.13 Avg points on TinyImageNet-100. These results suggest that graph-community routing complements existing VLM continual-learning methods by adding explicit expert communication under uneven class increments.

4.6 Sensitivity to Graph-Logit Scale and Identity Bias

We sweep the graph-logit scale λ and identity-bias coefficient α to test whether MoCE depends on a fragile hyperparameter choice. The MoE baseline removes the graph branch, so λ and α are undefined; in this setting, it obtains 77.90% Last and 84.08% Avg accuracy. Figure 2 shows that most MoCE settings outperform MoE. λ has a stronger effect than α: larger λ values allow graph evidence to contribute more strongly, with the best setting reaching 87.53% Avg and 82.71% Last at $\lambda = 100, \alpha = 0.3$. The identity-bias coefficient α controls how much each expert preserves its own state during message passing. Moderate self-preservation is generally effective: $\alpha = 0.3$ and $\alpha = 0.7$ perform strongly for large λ, while $\alpha = 0.5$ remains competitive in the middle of the sweep. These results support identity-biased expert communication, which avoids both oversmoothing from excessive mixing and isolated routing from overly strong self-preservation.

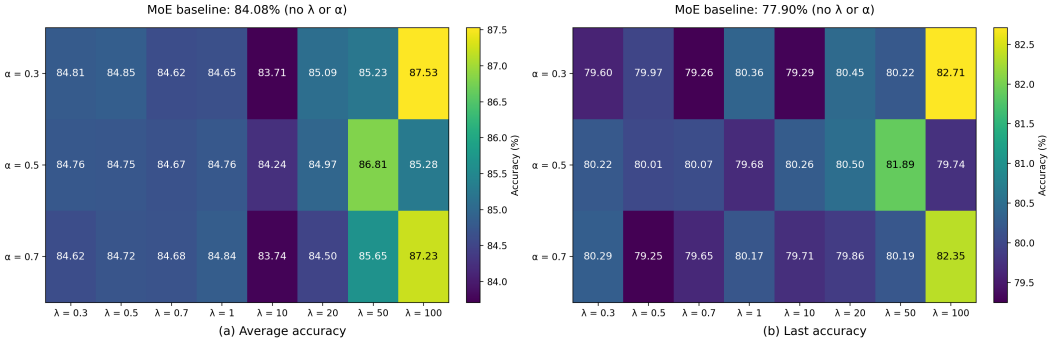


Figure 2: Sensitivity of MoCE to the residual graph-logit scale λ and identity-bias coefficient α.

Computational Cost We evaluated the computational efficiency of MoCE(N = 8) in the uneven CIFAR-100 setting using a single NVIDIA GeForce RTX 5080 GPU. MoCE contains 199.8M parameters in total, of which only 50.2M (25.1%) are trainable. During training, the model reaches a peak GPU memory usage of 11,427 MiB (∼11.2 GiB). The complete continual learning run, consisting of 10 tasks with 15 epochs per task, takes approximately 1.5 hours, with an average training time of approximately 0.238 s/iteration.These results in Table 4 indicate that MoCE achieves parameter-efficient adaptation by updating only about one quarter of the full model parameters while maintaining a moderate memory and runtime footprint.

Table 4: Computational cost comparison. For MoCE, computational cost is measured using a single NVIDIA GeForce RTX 5080 GPU.

Method	**Train Params** ↓	**GPU** ↓	**Time** ↓
LWF Li & Hoiem (2018)	149.6M	32172 MiB	1.54 s/it
LWF-VR Ding et al. (2022)	149.6M	32236 MiB	1.51 s/it
ZSCL Zheng et al. (2023)	149.6M	26290 MiB	3.94 s/it
MoE-Adapters	51.1M	19898 MiB	1.37 s/it
DDAS	8.7M	2461 MiB	0.21 s/it
Vanilla MoE Yu et al. (2024)	59.8M	22358 MiB	1.58 s/it
Ours	**50.2M**	**11427 MiB**	**0.238 s/it**
Δ	−66.44%	−56.53%	−93.96%

5 Conclusion

MoCE reformulates sparse MoE adaptation for class-incremental vision-language learning under uneven task complexity. Instead of routing each input to k isolated experts, MoCE selects k graph-community anchors: only these anchors execute their adapters, while their learned neighborhoods provide flexible routing evidence before sparse execution. Across uneven and balanced streams on CIFAR100 and TinyImageNet, controlled comparisons show that MoCE improves over standard MoE adapters by involving broader expert evidence through graph communities rather than densely executing more experts. The uneven results highlight the value of an adaptive routing evidence budget when class increments vary in size, while the balanced results show that graph-community routing remains useful even under regular class-increment streams. Together with the contextual baseline comparison, these findings suggest that effective continual VLM adaptation requires not only expert selection, but also structured communication among experts.

AI use statement

In this work, we used generative AI tools for grammar checking and polishing. We have not used generative AI tools for text or image generation, for running experiments, or for generating ideas. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts.

5.1 Reproducibility Statement

We have taken several steps to support the reproducibility of our results. The main paper describes the MoCE architecture, expert construction and routing mechanism, class-incremental learning protocol, and experimental settings. The appendix provides additional implementation details, including training configurations, hyperparameter, dataset and task construction procedures for both even and uneven task schedules, backbone settings, computational requirements, and supplementary ablation and sensitivity analyses. We also report results across multiple random seeds where applicable and provide detailed definitions of the evaluation metrics and experimental protocols. The supplementary materials include the code, configuration files, and experiment scripts needed to reproduce the reported results.

5.1.1 Acknowledgments

The authors declare that no funding is supported for this research. No conflict of interest is declared as well.

REFERENCES

Aishwarya Agrawal, Jiasen Lu, Stanislaw Antol, Margaret Mitchell, C. Lawrence Zitnick, Dhruv Batra, and Devi Parikh. Vqa: Visual question answering, 2016. URL https://arxiv.org/abs/1505.00468.

Fengqian Ding, Chen Xu, Han Liu, Bin Zhou, and Hongchao Zhou. Bridging pre-trained models to continual learning: A hypernetwork based framework with parameter-efficient fine-tuning techniques. *Information Sciences*, 674:120710, 2024. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2024.120710. URL https://www.sciencedirect.com/science/article/pii/S0020025524006236.

Yuxuan Ding, Lingqiao Liu, Chunna Tian, Jingyuan Yang, and Haoxuan Ding. Don't stop learning: Towards continual learning for the clip model, 2022. URL https://arxiv.org/abs/2207.09248.

Arthur Douillard, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord. Dytox: Transformers for continual learning with dynamic token expansion, 2022. URL https://arxiv.org/abs/2111.11326.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. doi: 10.1016/S1364-6613(99)01294-2.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey, 2024. URL https://arxiv.org/abs/2403.14608.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2790–2799, 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Saurav Jha, Dong Gong, and Lina Yao. Clap4clip: Continual learning with probabilistic finetuning for vision–language models. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024) Main Conference Track*, 2024. doi: 10.52202/079017-4102. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e94c57064dd5740c117b453bde8404c9-Abstract-Conference.html.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

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.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2018. doi: 10.1109/TPAMI.2017.2773081.

Wenzhuo Liu, Fei Zhu, Longhui Wei, and Qi Tian. C-CLIP: Multimodal continual learning for vision-language model. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sb7qHFYwBc.

Abhirama Subramanyam Penamakuri, Manish Gupta, Mithun Das Gupta, and Anand Mishra. Answer mining from a pool of images: Towards retrieval-based visual question answering, 2023. URL https://arxiv.org/abs/2306.16713.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 8748–8763, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017. doi: 10.1109/CVPR.2017.587.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Longxiang Tang, Zhuotao Tian, Kai Li, Chunming He, Hantao Zhou, Hengshuang Zhao, Xiu Li, and Jiaya Jia. Mind the interference: Retaining pre-trained knowledge in parameter efficient continual learning of vision-language models, 2024. URL https://arxiv.org/abs/2407.05342.

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.

Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, Chenliang Li, Yuanhong Xu, Hehong Chen, Junfeng Tian, Qi Qian, Ji Zhang, Fei Huang, and Jingren Zhou. mplug-owl: Modularization empowers large language models with multimodality, 2024. URL https://arxiv.org/abs/2304.14178.

Jiazuo Yu, Yunzhi Zhuge, Lu Zhang, Ping Hu, Dong Wang, Huchuan Lu, and You He. Boosting continual learning of vision-language models via mixture-of-experts adapters, 2024. URL https://arxiv.org/abs/2403.11549.

Jiazuo Yu, Zichen Huang, Yunzhi Zhuge, Lu Zhang, Ping Hu, Dong Wang, Huchuan Lu, and You He. Moe-adapters++: Toward more efficient continual learning of vision-language models via dynamic mixture-of-experts adapters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(12):11912–11928, 2025. doi: 10.1109/TPAMI.2025.3597942.

Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. Vision-language models for vision tasks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(8):5625–5644, August 2024. ISSN 0162-8828. doi: 10.1109/TPAMI.2024.3369699. URL https://doi.org/10.1109/TPAMI.2024.3369699.

Zangwei Zheng, Mingyuan Ma, Kai Wang, Ziheng Qin, Xiangyu Yue, and Yang You. Preventing zero-shot transfer degradation in continual learning of vision-language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19125–19136, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Zheng_Preventing_Zero-Shot_Transfer_Degradation_in_Continual_Learning_of_Vision-Language_Models_ICCV_2023_paper.html.

Da-Wei Zhou, Qi-Wei Wang, Zhi-Hong Qi, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Class-incremental learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9851–9873, December 2024. ISSN 1939-3539. doi: 10.1109/tpami.2024.3429383. URL http://dx.doi.org/10.1109/TPAMI.2024.3429383.

A Appendix

A.1 Detailed Standard MoE Routing

During training, the standard MoE router can use noisy top-k gating to encourage exploration and load balancing:

$$\tilde{z}_{b,i} = z_{b,i}^{\text{base}} + \epsilon_{b,i}\,\text{softplus}\big((h_b W_{\text{noise}})_i\big),$$

where $\epsilon_{b,i} \sim \mathcal{N}(0,1)$. The router selects

$$\mathcal{K}_b = \text{TopK}(\tilde{z}_b, k),$$

and normalizes scores only over the selected experts:

$$g_{b,i} = \begin{cases} \dfrac{\exp(\tilde{z}_{b,i})}{\sum_{j\in\mathcal{K}_b}\exp(\tilde{z}_{b,j})}, & i \in \mathcal{K}_b, \\ 0, & i \notin \mathcal{K}_b. \end{cases}$$

Only the selected experts are executed:

$$Y^{\text{moe}}_{:,b,:} = \sum_{i\in\mathcal{K}_b} g_{b,i}\, E_i(X_{:,b,:}).$$

A.2 Dataset and Split Details

We evaluate on CIFAR100 and TinyImageNet-100. CIFAR100 contains 100 classes. For TinyImageNet, we use a reproducible 100-class subset, denoted TinyImageNet-100, so that the number of classes matches CIFAR100 and the same class-increment schedules can be applied to both datasets. The TinyImageNet-100 subset is specified by the class-order file `class_orders/tinyimagenet100.yaml`, which selects the first 100 class IDs from the fixed TinyImageNet order file. All class orders are fixed before training and are shared by MoE and MoCE.

At increment t, the model receives training examples only from the new class set $\mathcal{Y}_t$ and is evaluated on all classes seen so far, $\cup_{s\le t}\mathcal{Y}_s$. We consider both uneven and balanced class-incremental protocols. The uneven protocols are 10-step streams with non-uniform numbers of classes per increment. The three main uneven schedules are:

$$U_1 = [46, 2, 2, 20, 20, 2, 2, 2, 2, 2],$$
$$U_2 = [13, 5, 4, 19, 8, 11, 28, 6, 3, 3],$$
$$U_3 = [3, 3, 4, 5, 6, 8, 11, 13, 19, 28].$$

Here, U_1 represents a bursty stream, where large increments appear early and in the middle; U_2 represents a shuffled stream, where smaller and larger increments alternate; and U_3 represents an ascending stream, where the number of new classes gradually increases. We also use a descending schedule for contextual comparison with other continual-learning baselines:

$$U_4 = [28, 19, 13, 11, 8, 6, 5, 4, 3, 3].$$

Similarly, for the 20-step setting, the remaining four schedules are:

$$U_5 = [6, 9, 14, 24, 47],$$
$$U_6 = [18, 23, 19, 34, 6],$$
$$U_7 = [48, 22, 22, 4, 4].$$
$$U_8 = [47, 24, 14, 9, 6].$$

The balanced protocols use 10, 20, or 50 increments, corresponding to 10, 5, or 2 classes per increment on the 100-class datasets. For all protocols, evaluation is conducted after each increment on the cumulative set of seen classes. Let A_t denote the top-1 accuracy after increment t. We report two main metrics: $\text{Last} = A_T$ and $\text{Avg} = \frac{1}{T}\sum_{t=1}^{T} A_t$. Last accuracy measures the final retained recognition performance after the full stream, while average accuracy measures the quality of the learning trajectory across increments.

For controlled MoE-versus-MoCE comparisons, the dataset, class order, backbone, prompt, optimizer, training duration, and total expert pool are identical. MoCE differs from MoE only in the graph-community routing branch. The implementation also logs per-task accuracy, forgetting, backward transfer, and forward transfer, but the main tables focus on Last and Avg accuracy because these metrics are consistently available across all recorded runs.

A.3 Implementation Details for Replication

Backbone and prediction head. All experiments use CLIP ViT-B/16 (Radford et al., 2021) with the prompt template `a bad photo of a {}.`. At increment t, text prompts are generated for all classes seen so far, and image-text cosine logits are used for single-head classification over the seen label set. The CLIP backbone and logit-scale parameter are frozen. The trainable parameters are the adapter experts, router/noise matrices, and, for MoCE, the graph mixer and graph-routing projections.

Expert architecture and routing. Experts are lightweight adapter MLPs inserted in CLIP transformer residual blocks, initialized with the LoRA-style option used by the codebase (Hu et al., 2022). Unless otherwise stated, the adapter bottleneck is 64, adapter dropout is 0.1, and the adapter scalar is 0.1. The code follows the MoE-Adapters protocol and maintains router/noise matrices for each incremental stage; MoE and MoCE share this protocol, while classification is always performed over the single text-prompt set of all seen classes. The controlled tables evaluate $N \in \{8, 16\}$ experts. The MoE baseline uses fixed top-k sparse routing with $k = 2$. MoCE uses a base anchor budget $k_0 = 2$ and, when the current increment contains C_t classes, the effective anchor count is

$$k_t = \min\left(N, \max\left(1, k_0 + \left\lfloor \frac{C_t - 10}{5} \right\rfloor\right)\right),$$

so small increments keep a compact anchor set while large increments can execute more community anchors. For MoCE, the graph branch has one graph-message-passing layer, hidden dimension 512, a one-hidden-layer adjacency head with width 512, symmetric adjacency logits, explicit self loops, row-wise normalization, and top-m adjacency sparsification with $m = \min(N, 8)$. The graph router uses per-expert graph logits and residual fusion $z = z^{\text{base}} + \lambda z^{\text{graph}}$. The default graph regularizers are load-balancing weight 0.01 and router z-loss weight 10^{-4}.

As illustrated in Figure 3(a), each expert is implemented as a bottleneck adapter that maps a D-dimensional token representation to a lower-dimensional space of size r through a linear down-projection, followed by ReLU activation and dropout. A zero-initialized linear up-projection restores the original dimensionality, and the output is multiplied by a fixed scaling factor s. No additional layer normalization is applied within the adapter.(b) shows the placement of the MoE adapter module in parallel with the frozen FFN in each transformer block of both the image and text towers. The CLS token determines the top-k routing weights, whereas the selected experts process the full token sequence. Their outputs are combined through a gated weighted sum and incorporated into the block output through residual addition alongside the FFN contribution.

Specifications at a Glance. The specified configuration uses a ViT-B/16 backbone with a hidden dimension of $D = 768$, an adapter bottleneck dimension of $r = 64$, and $N = 8$ experts per transformer block. Each adapter uses ReLU activation, a dropout rate of 0.1, and a fixed output scaling factor of $s = 0.1$. Including the biases of both linear projections, each expert contains $2Dr + D + r = 99{,}136$ parameters, yielding $793{,}088$ adapter parameters per block. The adapters are inserted into all 12 transformer blocks of each tower, resulting in approximately 9.52 million adapter parameters per tower and 19.03 million across the image and text towers combined. These counts cover only the bottleneck adapters and exclude the backbone and routing components.

Optimization. We train each increment for 15 epochs using AdamW with learning rate 10^{-3}, zero weight decay, and zero label smoothing. A cosine learning-rate schedule is used after a 30-iteration warmup. The batch size in the released CIL configuration files is 32. Each result is averaged over three runs produced by the runner script with three repeats.

Computational Resources. Experiments were conducted across one server and Ubuntu workstation. The main server used for experimental runs was equipped with an Intel(R) Core(TM) Ultra 7 265 processor at 2.40 GHz, 128 GB RAM, and an NVIDIA RTX 6000 Ada Generation GPU with 47.5 GB dedicated GPU memory, running Windows 11 Enterprise version 25H2. The Ubuntu workstation was equipped with an Intel Core Ultra 7 265K processor with 20 cores, 48.0 GiB RAM, and an NVIDIA GeForce RTX 5080 GPU with approximately 16 GB dedicated GPU memory, running Ubuntu 24.04.3 LTS. One 10-task experimental run required slightly over 3 hours on the all computational resources, with runtime varying by dataset, task split, and number of experts.

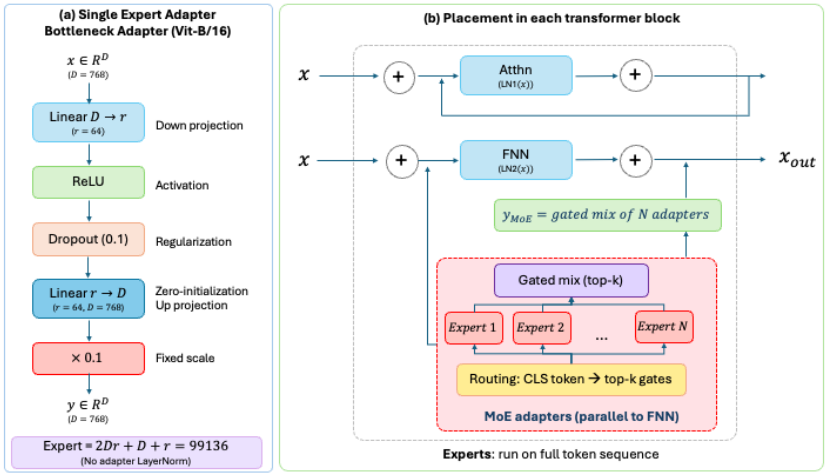


Figure 3: Expert-adapter architecture. (a) Each expert is a bottleneck adapter comprising a down-projection, ReLU activation, dropout, a zero-initialized up-projection, and fixed output scaling. (b) MoE adapter module operates in parallel with the frozen feed-forward network (FFN) in each transformer block of the image and text towers. The CLS token determines top-k routing, while the selected experts process the full token sequence. Their outputs are combined through a gated weighted sum and incorporated into the block output through residual addition.

A.4 MoCE Training Algorithm.

Algorithm 1 MoCE Training with Flexible Expert-Community Routing

Require: CIL tasks $\{\mathcal{D}_t\}_{t=1}^{T}$ with class counts C_t, frozen CLIP backbone, experts $\{E_i\}_{i=1}^{N}$
Require: base anchor budget k_0, graph-neighbor budget m, graph weight λ, identity bias α

for $t = 1$ to T **do**
  Set anchor budget $k_t = \min\left(N, \max\left(1, k_0 + \left\lfloor \frac{C_t - C_0}{\Delta} \right\rfloor\right)\right)$, or use $k_t = k_0$ if adaptive anchors are disabled.
  for mini-batch $(X, y) \sim \mathcal{D}_t$ **do**
    Extract class-token features $h_b = X_{0,b,:}$
    Compute base logits $z_b^{\text{base}} = h_b W_g + b_g$
    Build input-conditioned expert nodes $H_{b,i}^{(0)} = P h_b + e_i$
    Predict adjacency logits $S_b = \text{reshape}(\psi_A(h_b), N, N)$
    Compute row-normalized adjacency $A_{b,i,j}^0 = \frac{\exp(S_{b,i,j})}{\sum_{r=1}^{N} \exp(S_{b,i,r})}$
    Keep top-m neighbors with $M_{b,i,j} = \mathbb{I}\left[j \in \text{TopM}(A_{b,i,:}^0, m)\right]$ and $\bar{A}_{b,i,j} = \frac{A_{b,i,j}^0 M_{b,i,j}}{\sum_{r=1}^{N} A_{b,i,r}^0 M_{b,i,r} + \varepsilon}$
    Apply identity-biased mixing $A_b = \alpha I_N + (1-\alpha)\bar{A}_b$, where $\alpha \in [0, 1]$
    Pass messages over the expert graph: $H_b^{(\ell+1)} = \phi\Big(\text{LN}\Big(A_b H_b^{(\ell)} W_\ell + \mathbf{1} b_\ell^\top\Big)\Big)$, and set $Y = H^{(L_g)}$
    Compute per-expert graph evidence $z_{b,i}^{\text{graph}} = \langle Y_{b,i,:}, w_i^{\text{goe}} \rangle$
    Mix evidence $z_b = z_b^{\text{base}} + \lambda z_b^{\text{graph}}$
    Select community anchors $\mathcal{K}_b = \text{TopK}(z_b, k_t)$ and sparse gates $g_{b,i}$
    Form involved communities $\mathcal{U}_b = \bigcup_{i \in \mathcal{K}_b} (\{i\} \cup \mathcal{N}_b(i))$, where $\mathcal{N}_b(i) = \{j : M_{b,i,j} = 1\}$; execute only anchors E_i, $i \in \mathcal{K}_b$
    Update trainable experts, router, and graph parameters using $\mathcal{L} = \mathcal{L}_{\text{ce}} + \beta \mathcal{L}_{\text{lb}}(g, \ell) + \gamma \mathcal{L}_z(z)$
  end for
end for

A.5 Completed Controlled Uneven-Task Results

Table 5 summarizes uneven-task results on CIFAR100 and TinyImageNet-100 across U1–U8 with $N \in \{8, 16\}$ experts. MoCE achieves higher mean Last and Average accuracy than standard MoE in every evaluated configuration. On CIFAR100, Last-accuracy gains peak on U2 at $+9.99$ and $+9.83$ percentage points for $N = 8$ and $N = 16$, respectively. On TinyImageNet-100, the largest gains occur on U3, reaching $+7.39$ and $+6.77$ points, respectively. These results support the effectiveness of graph-community routing across diverse uneven class-arrival patterns, although the magnitude of improvement varies by configuration.

Table 5: Results under uneven class arrivals on CIFAR100 and TinyImageNet-100 across schedules U1–U8. Entries report mean ± standard deviation of accuracy (%) over three runs. Bold indicates the higher mean between MoE and MoCE for each dataset, schedule, expert count, and metric.

	CIFAR100				**TinyImageNet-100**			
	MoE		**MoCE**		**MoE**		**MoCE**	
Schedule	**Last**	**Avg**	**Last**	**Avg**	**Last**	**Avg**	**Last**	**Avg**
$N = 8$								
U1	76.8 ± 1.2	83.9 ± 0.3	**79.9** ± 3.0	**87.1** ± 1.0	80.2 ± 0.7	84.7 ± 0.2	**82.2** ± 3.2	**87.9** ± 1.2
U2	74.2 ± 0.4	85.3 ± 0.1	**84.1** ± 1.6	**90.9** ± 0.9	79.0 ± 0.1	86.6 ± 0.0	**83.8** ± 2.3	**90.2** ± 1.5
U3	79.1 ± 0.3	90.2 ± 0.1	**85.9** ± 0.8	**94.7** ± 0.3	80.8 ± 0.3	90.7 ± 0.1	**88.2** ± 0.8	**95.2** ± 0.4
U4	75.2 ± 0.5	82.2 ± 0.3	**81.5** ± 2.2	**87.5** ± 1.7	79.8 ± 0.2	84.9 ± 0.1	**81.5** ± 2.2	**87.5** ± 1.7
U5	82.8 ± 0.4	91.3 ± 0.2	**88.5** ± 0.3	**94.6** ± 0.0	83.9 ± 0.5	91.1 ± 0.2	**89.0** ± 1.1	**94.5** ± 0.5
U6	79.3 ± 0.6	86.7 ± 0.2	**84.1** ± 0.4	**89.3** ± 0.4	81.7 ± 0.2	87.5 ± 0.2	**85.7** ± 1.1	**90.4** ± 0.9
U7	80.3 ± 0.1	84.9 ± 0.0	**87.3** ± 1.3	**89.7** ± 0.7	81.2 ± 0.3	85.8 ± 0.2	**85.9** ± 1.4	**88.8** ± 0.6
U8	78.7 ± 0.2	84.3 ± 0.1	**84.5** ± 1.0	**88.0** ± 0.9	81.3 ± 0.6	85.7 ± 0.1	**83.2** ± 0.6	**86.7** ± 0.8
$N = 16$								
U1	75.9 ± 1.1	82.6 ± 0.2	**77.8** ± 4.9	**85.7** ± 1.6	79.4 ± 0.5	84.1 ± 0.2	**80.2** ± 1.2	**86.1** ± 0.9
U2	73.3 ± 0.6	84.5 ± 0.1	**83.1** ± 1.7	**90.7** ± 1.0	77.7 ± 0.6	86.1 ± 0.2	**82.0** ± 0.8	**89.6** ± 0.6
U3	77.8 ± 0.4	89.6 ± 0.1	**85.2** ± 0.7	**94.7** ± 0.1	79.8 ± 0.7	89.7 ± 0.1	**86.6** ± 2.3	**95.1** ± 0.5
U4	73.0 ± 0.4	81.3 ± 0.3	**78.9** ± 0.5	**86.1** ± 1.5	79.4 ± 0.4	84.5 ± 0.2	**80.8** ± 3.4	**87.0** ± 1.8
U5	82.0 ± 0.5	91.0 ± 0.2	**87.1** ± 1.1	**94.3** ± 0.3	84.0 ± 0.4	91.1 ± 0.1	**87.9** ± 1.0	**94.2** ± 0.3
U6	77.6 ± 0.3	85.9 ± 0.2	**84.1** ± 1.9	**89.6** ± 0.9	81.0 ± 0.4	87.3 ± 0.1	**84.1** ± 1.4	**89.5** ± 0.6
U7	79.5 ± 0.7	84.1 ± 0.1	**83.1** ± 1.3	**87.1** ± 0.9	81.0 ± 0.1	85.1 ± 0.2	**81.3** ± 0.7	**85.9** ± 0.7
U8	78.2 ± 0.5	83.5 ± 0.0	**83.7** ± 1.7	**88.1** ± 0.7	80.7 ± 0.5	85.3 ± 0.0	**85.3** ± 0.6	**87.9** ± 0.5

A.6 Ablation Study

To examine the graph-routing components, we ablate the identity-bias coefficient α and the residual graph-logit scale λ. We isolate each factor by fixing one to zero while varying the other, and report averaged last and average accuracy across available runs. Figure 4 shows that, when $\lambda = 0$, $\alpha = 0$ achieves the best performance. Larger α values lead to slightly lower results, suggesting that identity bias alone does not improve performance when graph-logit residuals are disabled. In contrast, varying λ with $\alpha = 0$ produces clearer differences. The graph-logit scale has a non-monotonic effect, with intermediate settings providing stronger performance than either very small or overly large values.

A.6.1 Graph-Related Ablation

We conduct ablations over the number of graph neighbors K_g, the number of expert nodes N, the identity-bias coefficient α, and the residual graph-logit scale λ. We vary K_g, α, and λ jointly under the standard expert configuration ($N = 8$) and repeat the analysis with a larger expert pool ($N = 16$). Average accuracy is reported for each configuration. These experiments evaluate whether

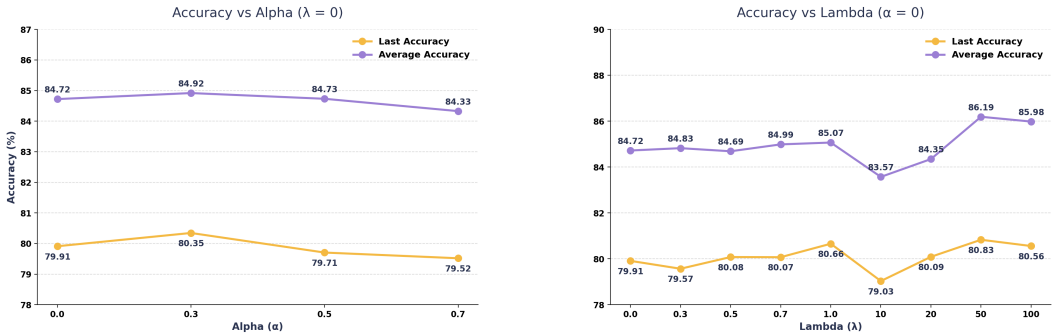


Figure 4: Sensitivity analysis of the graph-routing hyperparameters α and λ. We vary α with $\lambda = 0$ and λ with $\alpha = 0$, and report averaged last accuracy and average accuracy across runs.

the gains of MoCE depend on a specific graph topology or parameter setting and how the graph components interact with the size of the expert pool.

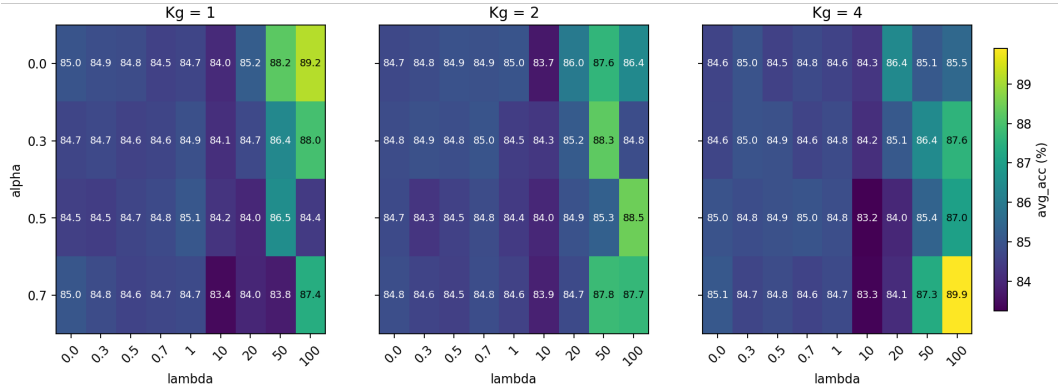


Figure 5: Average accuracy across the graph-neighbor budget K_g, graph-logit scale λ, and identity-bias coefficient α for the $N = 8$ expert configuration on TinyImageNet.

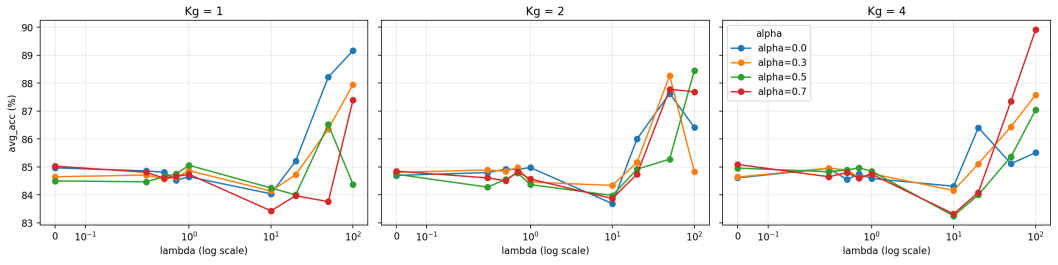


Figure 6: Average accuracy as a function of λ for different α and K_g settings under the $N = 8$ expert configuration on TinyImageNet.

Number of graph neighbors. Figures 5 and 6 shows that performance is not monotonic with respect to the graph-neighbor budget K_g. Under the $N = 8$ configuration, all three neighborhood sizes ($K_g \in \{1, 2, 4\}$) produce competitive results for small values of λ, with most configurations remaining around 84–85% average accuracy. Larger differences emerge when the graph contribution is strengthened. The best result, 89.9% average accuracy, is obtained with $K_g = 4$, $\alpha = 0.7$,

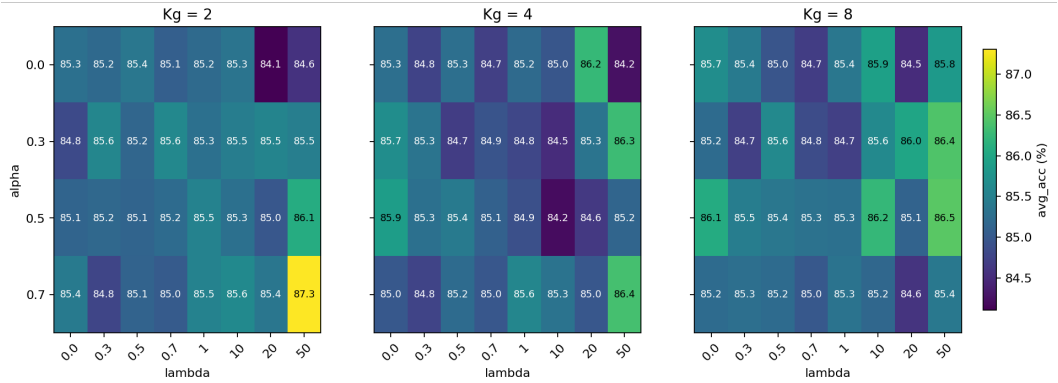


Figure 7: Average accuracy across the graph-neighbor budget K_g, graph-logit scale λ, and identity-bias coefficient α for the $N = 16$ expert configuration on TinyImageNet.

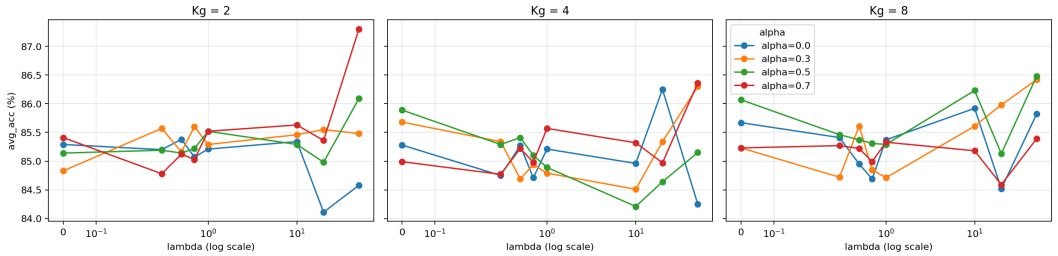
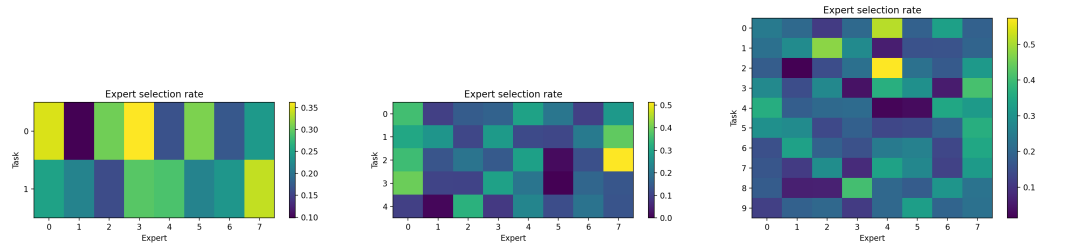


Figure 8: Average accuracy as a function of λ for different α and K_g settings under the $N = 16$ expert configuration on TinyImageNet.

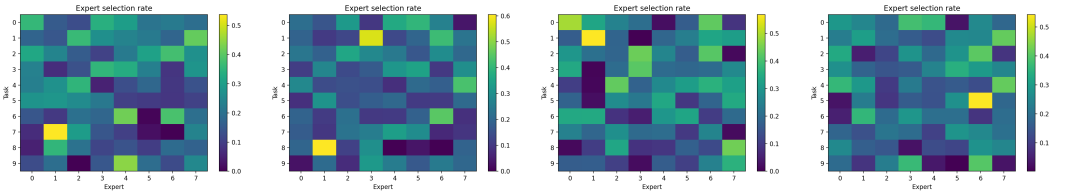
and $\lambda = 100$. However, strong performance is also observed with fewer neighbors; for example, $K_g = 1$, $\alpha = 0$, and $\lambda = 100$ reaches 89.2%. These results indicate that increasing the graph neighborhood does not uniformly improve performance. Instead, the appropriate neighborhood size depends on the strength of graph evidence and the degree of identity preservation.

Number of expert nodes. We next increase the expert pool from $N = 8$ to $N = 16$ and evaluate $K_g \in \{2, 4, 8\}$, as shown in Figures 7 and 8. Expanding the expert pool does not lead to a corresponding increase in accuracy. With $N = 16$, the best observed configuration reaches 87.3% average accuracy at $K_g = 2$, $\alpha = 0.7$, and $\lambda = 50$, which is below the best configuration obtained with $N = 8$. Moreover, most $N = 16$ configurations remain within a relatively narrow range of approximately 84–86%. This suggests that simply introducing more expert nodes is insufficient to improve continual-learning performance. Rather, the benefit of the expert graph depends on selective routing and effective communication among a suitably sized subset of experts.

Graph components. The joint ablation over α and λ further clarifies the roles of the two graph components. When λ is zero or small, changing α generally produces only modest differences, indicating that identity-biased mixing alone provides limited benefit. In contrast, nonzero graph-logit residuals can substantially improve performance, but their effect is highly non-monotonic. In the $N = 8$ experiments, intermediate-to-large values of λ produce the largest gains, while some configurations at $\lambda = 10$ result in a temporary performance decrease. Performance then increases sharply for selected combinations at $\lambda = 50$ or 100. The effect of α also depends on K_g and λ: no single identity-bias value is optimal across all configurations. For example, the strongest $K_g = 1$ result occurs at $\alpha = 0$, whereas the overall best $K_g = 4$ configuration uses $\alpha = 0.7$. Thus, α primarily modulates how strongly expert-specific information is preserved during message passing, while λ controls how much the graph-derived evidence contributes to routing.

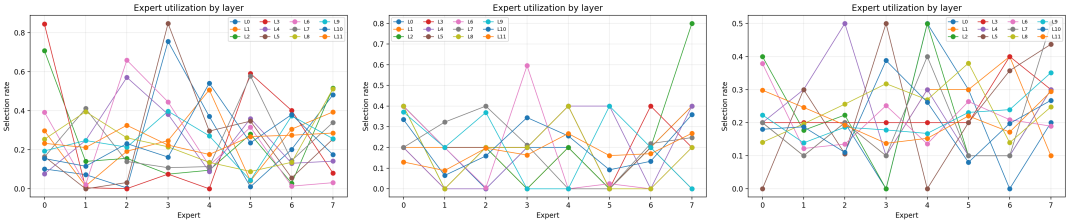


(a) Expert selection under even task sizes.

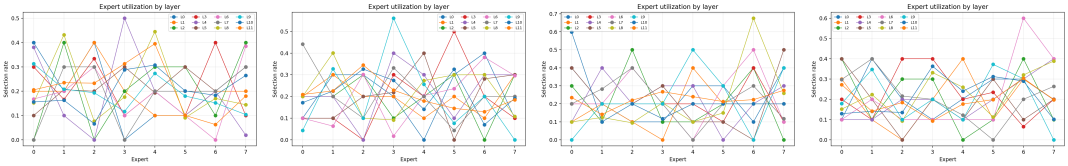


(b) Expert selection under uneven task sizes.

Figure 9: Expert selection patterns under even and uneven class-incremental learning schedules. Each heatmap reports the expert selection rate across tasks and experts. The first row shows the three even-task 50 steps, 20 steps, 10 steps configurations, while the second row shows the four uneven-task U1-U4 configurations.



(a) Expert utilization under even task sizes.

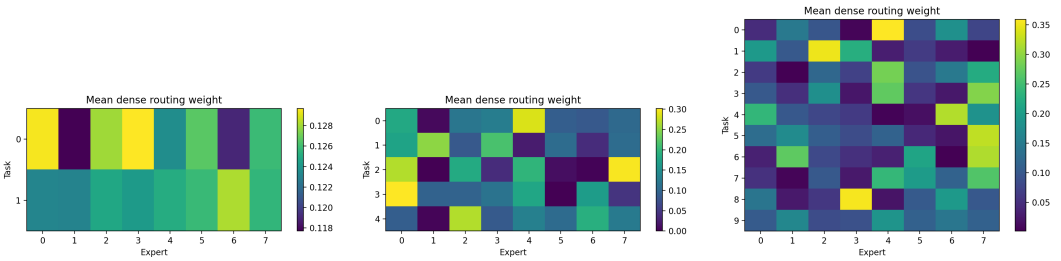


(b) Expert utilization under uneven task sizes.

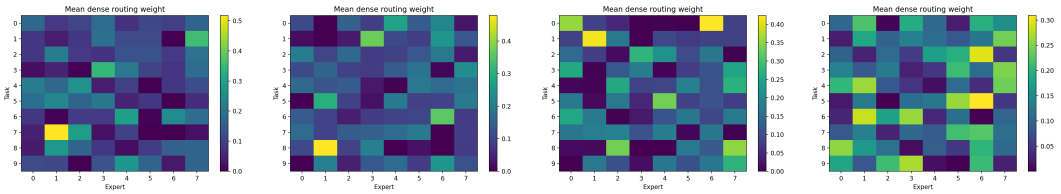
Figure 10: Expert utilization patterns under even and uneven class-incremental learning schedules. Each line chart reports the expert selection rate across tasks and experts. The first row shows the three even-task 50 steps, 20 steps, 10 steps configurations, while the second row shows the four uneven-task U1-U4 configurations.

A.7 Routing behavior analysis

The expert-selection (Fig. 9) and utilization patterns (Fig. 10) reveal how the expert pool is distributed across the stream. Although some experts are selected more frequently for particular increments, expert usage changes across tasks, with different experts becoming prominent at different stages. This behavior suggests that the router develops differentiated routing preferences while avoiding collapse onto a single expert or a small invariant subset. Importantly, the changing utilization pattern is consistent with the heterogeneous class composition of the uneven stream, where successive increments impose different routing demands.



(a) Routing weight under even task sizes.



(b) Routing weight under uneven task sizes.

Figure 11: Routing weight patterns under even and uneven class-incremental learning schedules. Each heatmap reports the expert selection rate across tasks and experts. The first row shows the three even-task 50 steps, 20 steps, 10 steps configurations, while the second row shows the four uneven-task U1-U4 configurations.

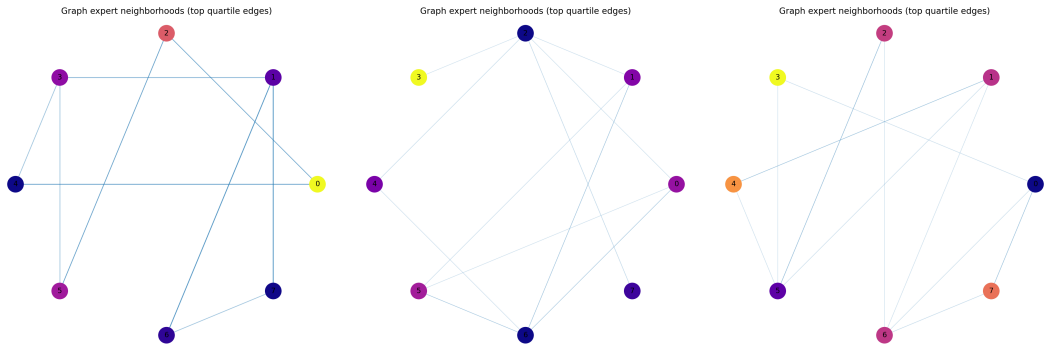
The routing-weight visualizations (Fig. 11) provide a finer-grained view of this adaptation. Rather than treating selected experts as independent units, GoE combines the base routing signal with graph-derived evidence. Consequently, the relative routing weights vary across experts and increments, showing that the graph branch modifies the strength of individual expert assignments rather than merely reproducing the original sparse router. The selected anchors are therefore determined using both their own routing evidence and information propagated from other experts in the learned graph.

Finally, the graph-neighborhood visualizations (Fig. 12) show that a selected expert can draw routing evidence from multiple neighboring experts before sparse execution. These neighborhoods are input-dependent and vary across the continual-learning stream, so the experts contributing evidence to a routing decision need not be identical to the experts that are ultimately executed. In other words, MoCE maintains sparse expert execution while allowing a broader expert community to participate in the routing decision. Taken together, these analyses demonstrate that the proposed graph component produces substantive changes in expert selection, utilization, routing weights, and neighborhood structure, providing mechanistic evidence for the performance improvements of GoE over conventional isolated top-k routing.

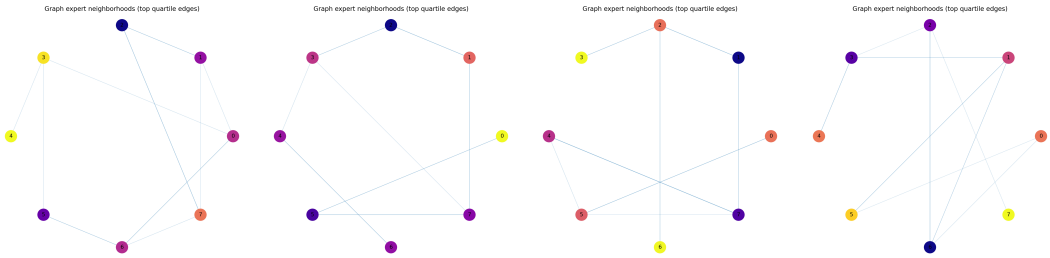
A.8 Additional backbone

To examine whether the benefit of MoCE generalizes beyond the primary backbone, we further evaluate the method using an additional backbone under the descending uneven class-incremental schedule $[28, 19, 13, 11, 8, 6, 5, 4, 3, 3]$ using CLIP ViT-B/32. The encoder is loaded from the official OpenAI CLIP checkpoints and remain frozen during continual adaptation. Only the MoCE adapter experts and routing modules are trained. ViT-B/16 and ViT-B/32 share the same transformer residual-block interface, so the same adapter insertion points, expert architecture, and graph-based community routing can be reused without changing the method. The backbone is selected through the configuration field `model_name` (default: `ViT-B/16`). Thus, any reported difference between ViT-B/16 and ViT-B/32 isolates the effect of patch size and feature granularity under an otherwise identical CIL protocol.

As shown in Table 6, MoCE consistently outperforms the standard MoE baseline in both Last and Average accuracy across CIFAR100 and TinyImageNet-100 and for both $N = 8$ and $N = 16$



(a) Graph neightbords under even task sizes.



(b) Graph neighborhoods under uneven task sizes.

Figure 12: Graph neighborhood patterns under even and uneven class-incremental learning schedules. Each structure chart reports the expert selection rate across tasks and experts. The first row shows the three even-task 50 steps, 20 steps, 10 steps configurations, while the second row shows the four uneven-task U1-U4 configurations.

Table 6: Additional-backbone results under the descending uneven class-incremental schedule U4. Entries report mean ± standard deviation of accuracy (%) over three runs. Δ denotes the improvement of MoCE over MoE.

		MoE		MoCE		Δ	
Dataset	Experts	Last	Avg	Last	Avg	Last	Avg
CIFAR100	$N = 8$	76.40 ± 1.00	83.27 ± 0.35	$\mathbf{80.59 \pm 0.58}$	$\mathbf{86.86 \pm 1.06}$	$+4.19$	$+3.59$
	$N = 16$	75.60 ± 1.05	82.53 ± 0.21	$\mathbf{81.79 \pm 1.83}$	$\mathbf{88.34 \pm 0.26}$	$+6.19$	$+5.81$
TinyImageNet-100	$N = 8$	79.88 ± 0.49	85.06 ± 0.08	$\mathbf{83.03 \pm 2.40}$	$\mathbf{86.96 \pm 2.16}$	$+3.15$	$+1.91$
	$N = 16$	79.60 ± 0.20	84.56 ± 0.08	$\mathbf{80.75 \pm 1.65}$	$\mathbf{87.00 \pm 1.12}$	$+1.15$	$+2.45$

experts. On CIFAR100, MoCE improves Last accuracy by $+4.19$ and $+6.19$ percentage points for $N = 8$ and $N = 16$, respectively, with corresponding Average-accuracy gains of $+3.59$ and $+5.81$ points. On TinyImageNet-100, the Last-accuracy improvements are $+3.15$ and $+1.15$ points, while Average accuracy increases by $+1.91$ and $+2.45$ points, respectively. These results indicate that the advantage of graph-based expert communication is not limited to the primary backbone and remains observable under a different backbone architecture and a strongly non-uniform class-arrival sequence.