PM-LoRA: Scalable Continual Learning via Progressive Merging of Low Rank Adapters

Xiaobing Yu[1,2], **Peijie Qiu**[1]*, **Jin Yang**[3], **Xiaoqi Zhao**[2], **Xuanzhao Dong**[4], **Wenhui Zhu**[4], **Xiwen Chen**[5], **Weiwei Ma**[1], **Xiaofeng Liu**[2], **Jiangpeng He**[6]†

[1]Washington University in St. Louis [2]Yale University
[3]Icahn School of Medicine at Mount Sinai
[4]Ira A. Fulton Schools of Engineering, Arizona State University
[5]Clemson University [6]Indiana University

xiaobing.yu@yale.edu, jhe2@iu.edu

Abstract

In continual learning, most parameter-efficient fine-tuning methods rely on task-specific low-rank adapters to incrementally adapt pre-trained models to new tasks. However, they overlook two critical challenges: one is the accumulation of numerous adapters that leads to linear growth in parameters and memory, and the other is the lack of an explicit mechanism to preserve previously learned knowledge during continual adaptation. In this work, we propose a simple yet scalable framework, termed Progressive Merging of Low-Rank Adapters (PM-LoRA), to address both issues simultaneously. Specifically, PM-LoRA trains a small task-specific spectral orthogonal adapter for each new task and progressively integrates it into a single evolving LoRA adapter that encodes all prior knowledge. This merging process enables continuous knowledge integration while directly mitigating catastrophic forgetting at the parameter level. Benefiting from this design, PM-LoRA achieves true scalability by maintaining constant parameter overhead throughout continual training. Experiments on four benchmark datasets demonstrate strong continual learning performance while maintaining a single LoRA adapter with a fixed rank budget. Our Code can be found in `https://github.com/VisualReasoner/pmlora`

1 Introduction

Modern continual learning (CL) aims to build models that can learn sequentially from a stream of tasks to mimic human perceptions. However, this paradigm faces a fundamental challenge known as catastrophic forgetting, where a model trained on a new task abruptly loses its ability to perform on all previous tasks. To combat forgetting, traditional CL methods [21, 16, 51] often allocate additional parameters or memory to preserve task-specific knowledge. However, this design introduces a fundamental scalability flaw: both the parameter count and storage cost grow linearly with respect to the number of tasks, which severely limits efficiency in long-term CL. To mitigate catastrophic forgetting, many traditional CL methods introduce additional parameters or memory as tasks accumulate. Representative strategies include rehearsal-based approaches that store samples from previous tasks and modular approaches [40, 45, 51, 13, 11, 49] that allocate task-specific parameters or adapters. While these designs can improve knowledge retention, they lead to model size and memory requirements that grow with the number of tasks. Such linear growth quickly becomes impractical in long-term deployment scenarios, creating a scalability bottleneck for CL systems.

*This work was done prior to joining Amazon.
†Corresponding author and project lead.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

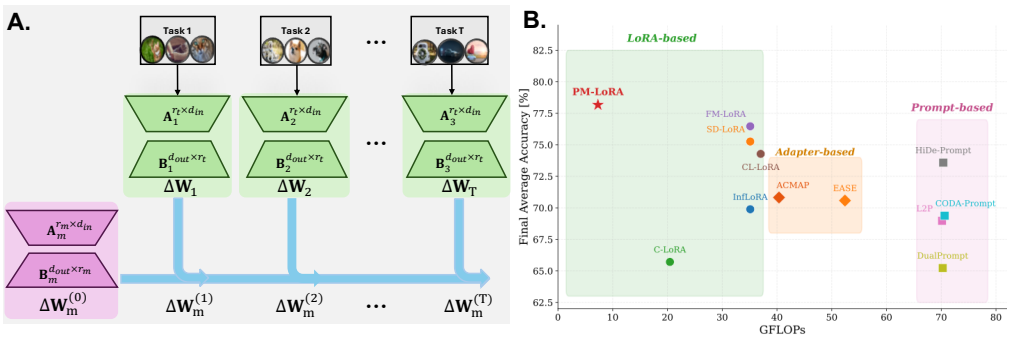


Figure 1: **A**. Overview of the proposed Progressive Merging LoRA (see details in Sec. 3). **B**. Comparison of GFLOPs versus Final Average Accuracy (%) across LoRA-based, Adapter-based, and Prompt-based continual learning methods on ImageNet-R ($N = 20$) using a pre-trained ViT-B/16 backbone. PM-LoRA achieves the highest accuracy (78.17%) while requiring significantly fewer GFLOPs (7.31) than all competing methods, demonstrating a superior efficiency-accuracy trade-off through its progressive merging framework with constant $\mathcal{O}(1)$ inference cost.

The emergence of parameter-efficient fine-tuning (PEFT) initially aims to alleviate these concerns. In particular, Low-Rank adaptive (LoRA) [14] alleviates this by decomposing weight updates into low-rank factors, dramatically reducing trainable parameters in each task while maintaining model expressiveness. However, when applied to CL, existing LoRA-based methods [45, 42, 11, 49] typically allocate a separate adapter for each task. Although each adapter contains only a small fraction of the model parameters (e.g., a LoRA module with rank r introduces roughly $r(d_{\text{in}} + d_{\text{out}})$ parameters, often about 1–5% of the backbone), the total parameter footprint grows linearly with the number of tasks. For instance, storing 20 task-specific adapters can already approach the parameter size of the original model. Consequently, both storage and inference costs increase with task count, undermining the parameter-efficiency advantage of LoRA in long-horizon CL (see Fig. 1**B**).

Another promising direction explores *model merging*, where independently trained task models are integrated into a single model for inference [31, 33, 43]. By consolidating multiple task adapters into one representation, this "merge-and-compress" paradigm has the potential to maintain constant inference cost. However, most existing merging approaches operate in a *post-hoc* manner: task adapters are first trained and stored independently, and only merged offline afterward. This design requires retaining all task-specific adapters during training, which leads to growing storage cost and makes the approach incompatible with CL scenarios that require *online* updates under a strict budget.

A natural extension is to perform *progressive merging*, where each new adapter is immediately integrated into the cumulative adapter after training, eliminating the need to retain previous task weights. Yet, naïvely averaging or summing adapters in this progressive manner fails catastrophically due to the following two coupled challenges that amplify over time:

Spectral Interference: In progressive merging, every new adapter interacts with the accumulated spectral structure of the merged adapter. When the new adapter is not spectrally orthogonal to the existing one, their principal components overlap, causing interference in the shared low-rank subspace. As this process repeats over tasks, these misaligned updates accumulate destructively, where correlated directions reinforce or cancel one another, producing uncontrolled energy inflation in the merged parameters [22, 42, 11]. The spectral geometry of the unified adapter thus becomes increasingly distorted, making each subsequent compression step more lossy and resulting in catastrophic forgetting of earlier tasks.

Magnitude Domination: Even when spectral orthogonality is approximately preserved, differences in adapter magnitudes create an implicit bias. In a progressive merging setting, the accumulated adapter carries the energy of all previous tasks, while newly trained adapters are inherently smaller residual updates. Consequently, earlier tasks tend to numerically dominate the merged representation, where high-magnitude task vectors disproportionately contribute to the merged weights [25, 33]. Over successive merges, this imbalance compounds, monopolizing the limited rank budget and diminishing the effective contribution of later tasks, ultimately freezing the model's capacity to absorb new knowledge.

Consequently, interference corrupts the directional structure of the spectral basis, while domination distorts energy allocation across singular modes. Together, they render the continual accumulation of low-rank adapters unstable under a fixed-rank constraint.

Building on this insight, we propose PM-LoRA (Progressive Merging of LoRA adapters), a framework to achieve scalable, constant-cost CL via progressively merged LoRA adapters to solve both of these core challenges, and eliminate the need for storing task-specific adapters (Fig.1**A**). Our method operates in two stages. The first stage, which occurs during training, is a **Spectral Compatibility Regularizer**. To solve spectral interference, we introduce this novel regularizer that forces the factors of a new adapter to be learned in a subspace orthogonal to the frozen, accumulated adapter. This precondition the adapter for a low-loss merge by eliminating destructive interference before it occurs. The second stage, applied at the end of each stage, is a **Scaled Progressive Merging** operator. To address magnitude domination, we propose a merge operator that first normalizes all adapters to a common scale, ensuring equal contribution from each task. It then applies an optimal yet efficient SVD-based compression on the scaled composite to form an adapter, preserving balanced task influence while maintaining computational scalability (see Fig. 1**B**).

In summary, our contributions are fourfold: *i)* We study CL under a *fixed LoRA capacity*, where task knowledge must be accumulated within a single bounded adapter rather than by allocating task-specific modules. We formulate CL in this setting as a recursive rank-constrained consolidation problem. *ii)* We propose **PM-LoRA**, a training and merging protocol that enables CL with a single LoRA adapter. PM-LoRA trains task updates to be compatible with future consolidation and integrates them online through rank-constrained merging, maintaining constant adapter capacity throughout learning. *iii)* We introduce the **Spectral Compatibility Regularizer (SCR)**, a theoretically motivated objective derived from an interference bound between LoRA updates, together with an efficient consolidation operator that computes the optimal rank-constrained approximation in a reduced low-rank core space. *iv)* Extensive experiments on multiple benchmarks demonstrate that PM-LoRA achieves strong CL performance while maintaining constant adapter capacity across tasks.

2 Related Work

Continual Learning (CL). Continual Learning (CL) aims to enable models to sequentially acquire new knowledge while retaining previously learned information, addressing the problem of catastrophic forgetting. Existing CL approaches can be broadly categorized into three families: rehearsal-based methods [29, 3, 23, 22, 10], which store and replay samples from previous tasks; regularization-based approaches [16, 1, 46, 21, 24, 9], which constrain parameter updates to protect previously learned knowledge; and architecture-based methods [7, 50, 44, 39, 48], which expand network capacity to isolate task representations. While effective in certain scenarios, these approaches often incur increasing memory or computational costs as tasks accumulate, limiting their scalability in long-term CL settings.

Parameter-Efficient Fine-Tuning-based approach in CL.Parameter-efficient fine-tuning (PEFT) [4] has emerged as a promising direction for adapting large pretrained models with minimal parameter updates. Representative PEFT methods include adapters, prompt tuning [18], prefix tuning [20], and LoRA [14]. Among them, LoRA introduces low-rank parameter updates to pretrained weights, significantly reducing training and storage costs. However, most PEFT-based CL approaches allocate a separate adapter for each task, causing the number of stored modules to grow linearly with the number of tasks. As tasks accumulate, this expanding storage footprint gradually undermines the parameter-efficiency advantage of PEFT.

Continual Learning with Model Merging. Model merging offers an alternative paradigm by combining multiple task-specific models into a unified representation without rehearsal data [43, 15, 25, 32, 26, 33, 19, 8]. For example, Fisher-weighted merging [26] improves over uniform averaging by weighting parameters according to their estimated importance. More recent work such as OPCM [33] introduces projection-based merging to reduce parameter interference between models. Despite these advances, most merging methods assume that task models are trained independently and merged post hoc, which leads to growing compute and storage costs as tasks accumulate.

Recent works attempt to mitigate interference within parameter-efficient settings. C-LoRA [49] reduces interference through orthogonal parameterization but does not explicitly address knowledge retention under rank-bounded consolidation. ACMap [8] merges independently trained task adapters

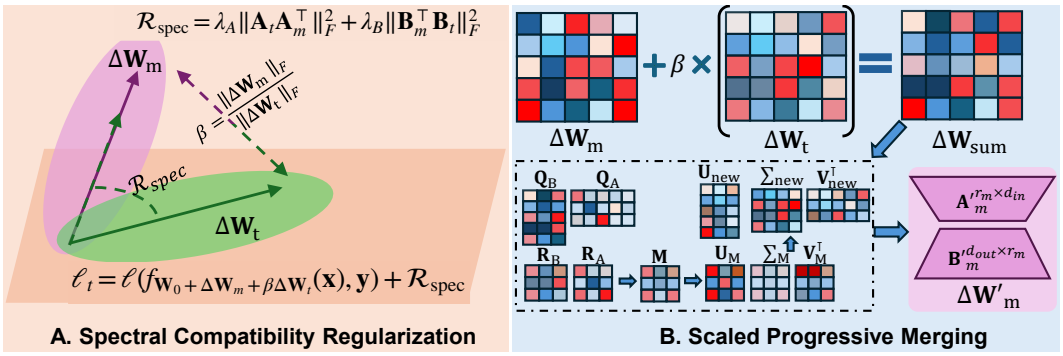


Figure 2: **A.** Spectral Compatibility Regularization penalizes factor cross terms between the temporary and frozen merged adapters during task training. The displayed norm ratio is the nominal scaling rule; initialization and near-zero norms are handled as described in Appendix B. **B.** Scaled Progressive Merging combines the two updates and computes a truncated-SVD approximation through a low-rank core, retaining one persistent adapter with rank at most $R_{\max}$ after each task.

as tasks arrive, but the adapters are not trained for compatibility with the merged state. InfLoRA [22] constrains the adaptation subspace without retaining all historical LoRA branches. PEARL [2] allocates task-specific LoRA ranks according to the proximity of task weights to reference task weights. Consequently, these methods do not directly address CL under a fixed adapter capacity. Our focus is training each incoming update for immediate consolidation into a rank-bounded LoRA state, where task updates must be both interference-aware and consolidation-compatible.

In contrast to prior methods, PM-LoRA targets CL under a *fixed adapter capacity*. Instead of allocating new adapters or storing multiple task models, PM-LoRA maintains a single LoRA adapter whose rank is bounded by a fixed budget. By training task updates to be compatible with future consolidation and integrating them through an online rank-constrained merging process, PM-LoRA enables continual knowledge accumulation while maintaining constant memory cost and mitigating interference between task updates.

3 Methodology

Let $\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_N\}$ denote a sequence of N tasks. Each task $\mathcal{T}_t$ is associated with a dataset $\mathcal{D}_t = \{\mathbf{x}_t^{(i)}, \mathbf{y}_t^{(i)}\}_{i=1}^{|\mathcal{D}_t|}$, where $\mathbf{x}_t^{(i)}$ is the input and $\mathbf{y}_t^{(i)}$ is the supervisory signal. The model $f(\cdot)$ is trained sequentially, and when learning task $\mathcal{T}_t$ we assume that data from previous tasks $\{\mathcal{T}_1, \ldots, \mathcal{T}_{t-1}\}$ are unavailable, following the rehearsal-free CL setting. Let $\mathbf{W}_0$ denote the weight matrix of a layer in the pretrained model. For task t, LoRA introduces a low-rank update $\Delta\mathbf{W}_t = \mathbf{B}_t\mathbf{A}_t$, where $\mathbf{A}_t$ and $\mathbf{B}_t$ are learnable low-rank matrices. The layer output $\mathbf{h}_t$ for the t-th task reads

$$\mathbf{h}_t := \mathbf{W}_0\mathbf{x}_t + \Delta\mathbf{W}_t\mathbf{x}_t = (\mathbf{W}_0 + \mathbf{B}_t\mathbf{A}_t)\mathbf{x}_t. \tag{1}$$

In this work, we maintain only a *single* LoRA adapter throughout CL. The rank of this adapter determines both its representational capacity and its computational footprint: a LoRA layer with rank r introduces $r(d_{\text{in}} + d_{\text{out}})$ additional parameters and incurs proportional inference cost. Allowing the rank to grow with the number of tasks would therefore increase model size and runtime linearly, degenerating into storing multiple task-specific adapters and defeating the parameter-efficiency objective of LoRA. However, CL requires accumulating knowledge from multiple tasks, which naturally increases the effective rank of the adapter. This creates an inherent dilemma: the model must incorporate new task knowledge while keeping the adapter's capacity fixed and bounded. Existing methods either merge independently trained adapters (e.g., ACMap [8] and OPCM [33] merge them sequentially) or share one adapter via routing (e.g., C-LoRA [49] uses self-regularization for continual adaptation), but neither supports continual consolidation under a strict fixed-rank budget. Here we study how to train each update for its subsequent rank-constrained consolidation. The core challenge is to continually consolidate new updates into a bounded adapter without corrupting prior knowledge: regular LoRA adapter merging causes subspace interference with $\Delta\mathbf{W}_m$ and rank truncation that suppresses later updates.

We propose PM-LoRA, a CL protocol that trains each task update to be consolidation-compatible and absorbs it online via rank-constrained merging, enabling continual updates under fixed LoRA

capacity. Concretely, before task t, the model stores a merged adapter $\Delta\mathbf{W}_m = \mathbf{B}_m\mathbf{A}_m$ whose rank r_m satisfies $r_m \leq R_{\max}$. For the t-th task, we attach a small temporary adapter $\Delta\mathbf{W}_t$ with rank $r_t \ll R_{\max}$ and train it on the current dataset $\mathcal{D}_t$. After training, the temporary adapter is consolidated into the merged adapter and then discarded. This ensures that after every task the model returns to the same compact state: a single LoRA adapter whose rank never exceeds the budget $R_{\max}$.

PM-LoRA achieves this through a two-stage design. Stage 1 (Sec. 3.1) trains each task update to be *mergeable* with the current merged adapter by reducing interference between their update directions. Stage 2 (Sec. 3.2) consolidates the balanced old and new updates through an online rank-constrained merging step, allowing knowledge to accumulate across tasks while maintaining the fixed adapter capacity. The algorithmic taxonomy is detailed in Algorithm 1.

3.1 Stage 1: Learning a Mergeable Update

At task t, the backbone parameters and the current merged adapter $\Delta\mathbf{W}_m$ are frozen, and only a temporary task adapter $\Delta\mathbf{W}_t$ is optimized on the dataset $\mathcal{D}_t$. The goal of this stage is not merely to fit the current task but to learn an update that can later be consolidated into the shared merged adapter with minimal loss of previously stored information. A key challenge arises from interference between task updates. When two LoRA adapters modify similar directions in parameter space, their combination may overwrite previously stored knowledge.

Proposition 1 (Interference Bound)**.** *The interference between the current merged adapter $\Delta\mathbf{W}_m$ and $\Delta\mathbf{W}_t$ can be measured by the trace of their dot product, which is upper-bounded:*

$$\left|\mathrm{tr}(\Delta\mathbf{W}_m^\top \Delta\mathbf{W}_t)\right| \leq \|\mathbf{B}_m^\top\mathbf{B}_t\|_F \|\mathbf{A}_t\mathbf{A}_m^\top\|_F. \tag{2}$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix.

Proof. The proof follows the Cauchy–Schwarz inequality and is detailed in the Appendix A □

This bound shows that interference between LoRA updates is governed by the alignment between their input and output subspaces. Therefore, instead of enforcing strict orthogonality on the weight matrices, we penalize these alignments through a novel **Spectral Compatibility Regularizer (SCR)**:

$$\mathcal{R}_{\text{spec}} = \lambda_A \|\mathbf{A}_t\mathbf{A}_m^\top\|_F^2 + \lambda_B \|\mathbf{B}_m^\top\mathbf{B}_t\|_F^2. \tag{3}$$

where λ_A and λ_B control the strength of the input and output subspace penalties and $\|\cdot\|_F$ denotes the Frobenius norm. A geometric illustration of the proposed SCR is shown in Fig. 2**A**.

In addition to directional interference, a second issue arises from magnitude imbalance between the two adapters. Because $\Delta\mathbf{W}_m$ aggregates information from multiple tasks, its magnitude is typically larger than that of the temporary adapter. If the temporary update is trained in isolation but later merged with a much stronger merged adapter, its contribution can be suppressed during rank-constrained consolidation. To remove this train–merge mismatch, we scale the temporary adapter during training by $\beta_t := \frac{\|\Delta\mathbf{W}_m\|_F}{\|\Delta\mathbf{W}_t\|_F}$, where β_t is computed from the low-rank factors and detached from the computation graph. Unlike methods such as SD-LoRA [42], which learn task-specific scaling coefficients to gate adapter contributions, the scaling factor here is a deterministic normalization derived from the relative magnitudes of the two adapters. Its role is not to gate task importance but to align the training objective with the consolidation operator used in Stage 2.

Final objective. The final training objective for the t-th task reads

$$\mathcal{L}_t = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}_t}\left[\ell(f_{\theta+\Delta\mathbf{W}_m+\beta_t\Delta\mathbf{W}_t}(\mathbf{x}),\mathbf{y})\right] + \mathcal{R}_{\text{spec}}. \tag{4}$$

Together, these components train the temporary adapter to be *mergeable* with the existing merged adapter. Stage 1 therefore prepares task updates that are compatible with bounded-capacity consolidation in Stage 2.

3.2 Stage 2: Online Fixed-Budget merging

After finishing task t, PM-LoRA immediately absorbs the temporary adapter $\Delta\mathbf{W}_t$ into the merged adapter $\Delta\mathbf{W}_m$, and then discards $\Delta\mathbf{W}_t$. This design ensures that the model maintains a single LoRA

adapter throughout CL. To improve clarity, we define an intermediate merged update as

$$\Delta \mathbf{W}_{\text{sum}} = \Delta \mathbf{W}_m + \beta_t \Delta \mathbf{W}_t. \tag{5}$$

Because the merged adapter must respect the fixed rank budget $R_{\max}$, the combined update in Eq. (5) cannot be stored directly. Instead, PM-LoRA consolidates it by computing the optimal rank-constrained projection $\text{Proj}_{R_{max}}(\cdot)$:

$$\Delta \mathbf{W}_m^+ = \text{Proj}_{R_{\max}}(\Delta \mathbf{W}_{\text{sum}}) = \underset{\text{rank}(\widehat{\Delta \mathbf{W}_m^+}) \leq R_{\max}}{\arg\min} \|\Delta \mathbf{W}_{\text{sum}} - \widehat{\Delta \mathbf{W}_m^+}\|_F^2. \tag{6}$$

This projection defines the consolidation operator of PM-LoRA. Rather than storing task-specific adapters, the method repeatedly merges the current task update with the existing merged adapter and compresses the result back into a bounded representation. Consequently, knowledge from successive tasks is accumulated through a recursive consolidation process in which each update is integrated into the shared adapter while preserving its most informative directions.

This consolidation step is theoretically justified by the Eckart–Young theorem [6], which states that truncated SVD provides the optimal rank-constrained approximation under the Frobenius norm. Therefore, Eq. (6) retains as much information as possible from the merged update while respecting the fixed memory capacity. However, a direct computation of Eq. (6) would require performing SVD on the full matrix $\Delta \mathbf{W}_{\text{sum}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, which is computationally expensive for large models. PM-LoRA instead exploits the low-rank structure of LoRA adapters through QR factorization to perform consolidation in a reduced space.

First, we substitute the factorized forms into Eq. (5)

$$\Delta \mathbf{W}_{\text{sum}} = \mathbf{B}_m \mathbf{A}_m + \beta_t \mathbf{B}_t \mathbf{A}_t = [\, \mathbf{B}_m \mid \beta_t \mathbf{B}_t \,] \begin{bmatrix} \mathbf{A}_m \\ \mathbf{A}_t \end{bmatrix}. \tag{7}$$

which shows that the merged update has a rank at most $r = r_m + r_t$. Instead of forming the dense matrix $\Delta \mathbf{W}_{\text{sum}}$, we compute thin QR decompositions of the concatenated factors

$$\mathbf{B}_{\text{cat}} = \mathbf{Q}_B \mathbf{R}_B, \qquad \mathbf{A}_{\text{cat}}^\top = \mathbf{Q}_A \mathbf{R}_A. \tag{8}$$

This yields the compact representation

$$\Delta \mathbf{W}_{\text{sum}} = \mathbf{Q}_B \mathbf{M} \mathbf{Q}_A^\top, \qquad \mathbf{M} = \mathbf{R}_B \mathbf{R}_A^\top. \tag{9}$$

where the small matrix $\mathbf{M} \in \mathbb{R}^{r \times r}$ captures the entire interaction between the merged updates. We then compute the SVD of the core matrix $\mathbf{M} = \mathbf{U}_M \Sigma_M \mathbf{V}_M^\top$. Retaining the smallest number of singular values that preserve an energy ratio τ while enforcing $r_{\text{new}} \leq R_{\max}$, the updated LoRA factors are reconstructed as

$$\mathbf{B}'_m = \mathbf{Q}_B \mathbf{U}_M^{(r_{\text{new}})} (\Sigma_M^{(r_{\text{new}})})^{1/2}, \quad \mathbf{A}'_m = (\Sigma_M^{(r_{\text{new}})})^{1/2} (\mathbf{V}_M^{(r_{\text{new}})})^\top \mathbf{Q}_A^\top. \tag{10}$$

Finally, the merged adapter is updated by

$$\Delta \mathbf{W}'_m \leftarrow \mathbf{B}'_m \mathbf{A}'_m. \tag{11}$$

after which the temporary adapter $\Delta \mathbf{W}_t$ is discarded. The full consolidation process is illustrated in Fig. 2**B**. This procedure guarantees that after every task, the model returns to the same compact state: a single LoRA adapter whose rank never exceeds the fixed budget $R_{\max}$. Because all expensive operations are performed in the reduced core space rather than on the full weight matrix, the consolidation step can be computed efficiently with complexity $O(dr^2 + r^3)$ where $r = r_m + r_t \ll d$.

4 Experiments

4.1 Experimental Setups

We evaluate PM-LoRA on several CL benchmarks under the class-incremental setting: CIFAR-100 [17] (100 classes), ImageNet-R [12] (200 classes of artistic renditions), CUB-200 [34] (200 bird species), and VTAB [47] (50 classes). Each dataset is partitioned into multiple tasks with equal numbers of classes per task. During training, each task-specific LoRA adapter is optimized using only its current task dataset without storing or replaying past samples. During evaluation, a single

Table 1: Performance comparison on Split CIFAR-100, CUB200, and Split ImageNet-R under N incremental tasks. We report Final Average Accuracy (FA%), Cumulative Average Accuracy (CA%), and Average Forgetting Measure (FM%) of all methods using the same pre-trained model (**ViT-B/16-IN21K**). For FA and CA, higher is better; for FM, lower is better. The best and second-best results for each metric are highlighted in **red** and **blue**, respectively. More experiments with larger backbone can be found in **Appendix**.

Method	CIFAR-100 ($N = 10$)			CUB200 ($N = 20$)			ImageNet-R ($N = 40$)		
	FA (↑)	**CA** (↑)	**FM** (↓)	**FA** (↑)	**CA** (↑)	**FM** (↓)	**FA** (↑)	**CA** (↑)	**FM** (↓)
Fine-tune	34.17 ± 7.55	37.51 ± 8.20	58.37 ± 9.63	30.47 ± 7.36	36.69 ± 8.37	53.21 ± 8.42	27.35 ± 8.31	32.17 ± 9.42	62.18 ± 14.27
EWC [16]	86.01 ± 1.86	89.42 ± 1.71	4.57 ± 0.56	65.71 ± 2.31	71.02 ± 1.79	8.42 ± 1.12	64.35 ± 2.47	70.34 ± 1.36	8.43 ± 1.85
SI [46]	86.51 ± 0.61	90.15 ± 0.52	4.54 ± 0.15	66.45 ± 2.11	73.12 ± 1.45	8.65 ± 1.07	63.02 ± 2.35	69.02 ± 1.41	8.21 ± 1.42
L2P [41]	80.71 ± 1.47	86.61 ± 0.91	6.29 ± 1.17	61.29 ± 2.27	66.31 ± 1.75	9.53 ± 1.84	58.21 ± 2.75	65.28 ± 1.81	11.62 ± 2.34
DualPrompt [40]	85.51 ± 0.48	88.59 ± 0.81	4.32 ± 0.86	65.47 ± 1.41	69.98 ± 1.71	7.21 ± 1.43	62.07 ± 2.71	67.83 ± 2.39	9.43 ± 1.62
S-Prompt [38]	86.02 ± 0.76	88.97 ± 0.69	3.41 ± 0.53	66.21 ± 2.46	71.53 ± 2.16	6.85 ± 1.21	63.21 ± 2.46	68.03 ± 2.16	8.85 ± 1.47
CODA-Prompt [30]	86.13 ± 0.53	89.62 ± 1.07	4.59 ± 0.26	66.32 ± 1.51	70.21 ± 1.37	6.42 ± 0.87	63.77 ± 1.33	68.16 ± 0.98	7.62 ± 0.85
HiDe-Prompt [36]	**91.67** ± 0.28	**97.61** ± 0.21	2.91 ± 0.36	70.11 ± 0.57	74.07 ± 0.35	4.83 ± 0.67	68.29 ± 0.82	73.62 ± 0.69	7.13 ± 0.91
MOS [32]	87.12 ± 0.37	91.72 ± 0.25	2.61 ± 0.32	73.13 ± 1.21	76.35 ± 0.87	4.62 ± 0.71	71.22 ± 0.81	77.02 ± 0.62	6.42 ± 0.74
SEMA [35]	85.17 ± 0.35	88.25 ± 0.44	2.46 ± 0.51	75.31 ± 0.64	76.17 ± 0.58	3.97 ± 0.62	73.25 ± 0.47	78.27 ± 0.59	7.85 ± 0.83
EASE [50]	88.32 ± 0.51	92.16 ± 0.21	1.94 ± 0.27	74.35 ± 0.76	78.11 ± 0.45	4.21 ± 0.58	72.06 ± 0.83	75.11 ± 0.63	6.32 ± 0.92
InfLoRA [22]	82.17 ± 0.49	86.88 ± 0.72	3.56 ± 0.36	69.22 ± 1.88	74.56 ± 0.83	5.17 ± 1.05	64.15 ± 1.32	73.28 ± 0.51	9.84 ± 1.32
C-LoRA [49]	82.97 ± 2.34	88.81 ± 0.77	2.82 ± 0.95	66.58 ± 2.07	72.16 ± 1.73	7.84 ± 1.42	62.55 ± 3.26	69.77 ± 2.73	6.97 ± 1.21
SD-LoRA [42]	87.16 ± 1.92	89.94 ± 0.69	1.69 ± 0.98	74.01 ± 0.78	79.62 ± 0.57	3.62 ± 0.59	72.12 ± 0.88	77.51 ± 0.62	6.81 ± 0.94
CL-LoRA [11]	87.31 ± 0.18	90.22 ± 0.16	1.96 ± 0.13	75.31 ± 0.76	80.76 ± 0.55	3.41 ± 0.42	74.28 ± 0.21	**81.23** ± 0.62	6.45 ± 0.62
PM-LoRA	**91.55** ± 0.17	96.36 ± 0.52	**1.21** ± 0.19	**78.17** ± 0.77	**83.56** ± 0.31	**2.43** ± 0.31	**76.37** ± 0.51	79.26 ± 0.44	**4.21** ± 0.43
PM-LoRA (TIES)	91.33 ± 0.09	**97.21** ± 0.17	**1.45** ± 0.21	**78.01** ± 0.56	**83.25** ± 0.39	**2.78** ± 0.27	**76.49** ± 0.47	**80.67** ± 0.59	**3.97** ± 0.39

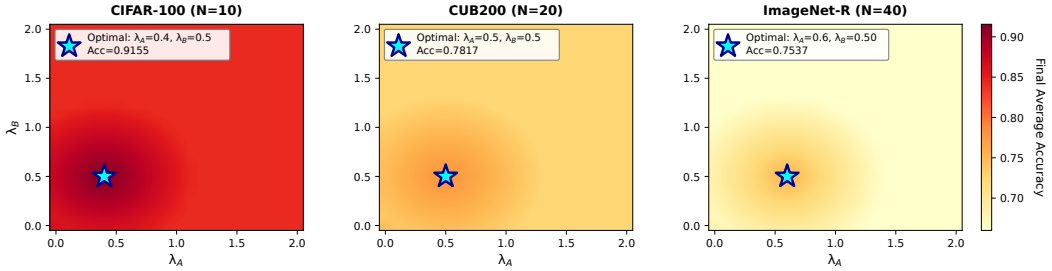


Figure 3: **Hyperparameter sensitivity analysis of regularization parameters λ_A and λ_B,** Heatmaps show final average accuracy for different (λ_A, λ_B) combinations across CIFAR-100 (N=10), CUB200 (N=20), and ImageNet-R (N=40). Blue stars mark optimal configurations, which notably fall off the diagonal, indicating λ_A and λ_B require independent tuning.

adapter is used to classify all previously seen classes without task identifiers or rehearsal memory. All experiments are implemented in PyTorch 2.1 and run on NVIDIA A100 GPUs (40GB VRAM). We use a Vision Transformer (ViT-B/16) backbone pre-trained on ImageNet-21K [5], following common practice in recent CL studies. Each task is trained for 20 epochs using the Adam optimizer with learning rate 1×10^{-4} and weight decay 0.1. Gradient clipping with maximum ℓ_2 norm of 1.0 is applied for stability. The training loss consists of cross-entropy over the unified classifier head plus the Spectral Compatibility Regularizer applied to all LoRA layers. Images are resized to 224×224 and normalized using ImageNet statistics (mean $(0.485, 0.456, 0.406)$, std $(0.229, 0.224, 0.225)$). Training augmentation includes random horizontal flipping (0.5 probability) and random resized crop to 224×224 with scale $[0.8, 1.0]$. No augmentation is applied during evaluation. Batch sizes are 128 for CIFAR-100 and ImageNet-R, and 64 for CUB-200. CIFAR-100 is divided into 10 tasks of 10 classes each. ImageNet-R and CUB-200 are split into 40 and 20 tasks respectively. Class orders are determined by a single random permutation with seed 42 and the same order is used for all methods.

As detailed in Table 1, we systematically compared PM-LoRA with representative methods from four CL paradigms based on regularization, prompt, adapter, and LoRA, respectively. All methods use the same backbone. See Appendix B for more implementation and method details.

Following standard evaluation protocols in CL, we report Final Average Accuracy (FA), the average accuracy over all seen tasks after the last task; Cumulative Average Accuracy (CA), the running average of this accuracy over the task sequence; and Average Forgetting (FM). Formal definitions are given in Appendix C.

Table 2: Comparison to LoRA-Merging methods for the same pre-trained model (**ViT-B/16-IN21K**) on other datasets with different amounts of tasks. Our PM-LoRA progressively merges each task-specific LoRA without storing them.

Method	Progressive Merging	Post-hoc Merging	VTAB ($N = 5$) FA(↑)	VTAB ($N = 5$) CA(↑)	CUB200 ($N = 20$) FA(↑)	CUB200 ($N = 20$) CA(↑)
Weight Average	×	✓	74.41 ± 0.65	80.25 ± 0.37	61.21 ± 2.07	66.47 ± 1.33
TA [15]	×	✓	76.51 ± 0.59	82.17 ± 0.69	68.81 ± 1.25	73.42 ± 1.47
Fisher [26]	×	✓	77.35 ± 0.87	83.17 ± 0.51	73.56 ± 3.17	78.45 ± 2.79
KnOTS [31]	×	✓	84.13 ± 0.23	90.78 ± 0.06	72.52 ± 1.77	78.24 ± 1.52
OPCM [33]	✓	✓	82.51 ± 0.29	87.65 ± 0.16	70.22 ± 1.36	76.52 ± 0.97
MagMax [25]	×	✓	84.81 ± 0.49	91.07 ± 0.31	71.22 ± 1.21	77.37 ± 1.19
ACMap [8]	✓	×	87.55 ± 0.58	92.16 ± 0.28	72.14 ± 1.06	77.44 ± 0.96
PM-LoRA (Post-hoc)	×	✓	**90.31** ± 0.09	**95.15** ± 0.03	**77.35** ± 0.63	**83.12** ± 0.47
PM-LoRA	✓	×	**90.15** ± 0.08	**95.53** ± 0.03	**78.17** ± 0.77	**83.56** ± 0.31

Table 3: Ablation study in PM-LoRA components, Spectral Compatibility Regularizer (SCR), Scaled Progressive Merging (SPM), and β-Scaling, on ImageNet-R across different N tasks. Gain is the improvement over the baseline. The full configuration achieves the highest FA.

Components			$N = 10$		$N = 20$		$N = 40$		$N = 50$	
SCR	SPM	β-Scaling	FA(↑)	Gain	FA(↑)	Gain	FA(↑)	Gain	FA(↑)	Gain
×	×	×	76.25	–	67.23	–	63.33	–	58.26	–
✓	×	×	78.82	+2.57	71.58	+4.35	68.34	+5.01	64.15	+5.89
×	✓	×	77.33	+1.08	69.71	+2.48	68.17	+4.84	63.23	+4.97
×	×	✓	77.68	+1.43	70.37	+3.14	67.45	+4.12	63.09	+4.83
✓	✓	×	85.55	+9.30	75.78	+8.55	72.08	+8.75	69.84	+11.58
✓	×	✓	82.23	+5.98	73.35	+6.12	70.46	+7.13	67.42	+9.16
×	✓	✓	83.73	+7.48	74.41	+7.18	71.45	+8.12	68.01	+9.75
✓	✓	✓	**87.37**	**+11.12**	**78.17**	**+10.94**	**76.37**	**+13.04**	**76.15**	**+17.89**

4.2 Performance on Benchmark Datasets.

In Table 1, PM-LoRA achieves 91.55% FA on Split CIFAR-100 ($N = 10$), the best among the compared LoRA-based methods and 0.12 percentage points below HiDe-Prompt (91.67%). On CUB200 ($N = 20$), PM-LoRA reaches 78.17% FA, outperforming all compared methods and exceeding the next-best LoRA-based method by 2.86 percentage points. On ImageNet-R ($N = 40$), PM-LoRA achieves 76.37% FA, exceeding SD-LoRA and CL-LoRA by 4.25 and 2.09 percentage points, respectively, while retaining a single merged adapter after each task. InfLoRA also supports task-count-independent inference without retaining all historical LoRA branches. PM-LoRA addresses the complementary problem of training each temporary update for consolidation into a persistent adapter with a fixed rank budget.

We also evaluate PM-LoRA-TIES, which applies TIES [43] trimming, sign election, and disjoint merging after magnitude scaling and before rank-constrained projection. It reaches 76.49% FA on ImageNet-R, compared with 76.37% for base PM-LoRA. The gains are not consistent across datasets. The details and more benchmark performance studies with larger backbone models can be found in Appendix C

4.3 Ablation study

Merging Method Comparisons. Table 2 compares PM-LoRA with the evaluated model-merging baselines and a post-hoc variant of PM-LoRA.

Our progressive merging method, PM-LoRA, achieves 90.15% FA and 95.53% CA on VTAB, significantly outperforming all other methods. We also found that even if PM-LoRA performs post-hoc merging, the performance is still better than that of other post-hoc merging methods. The post-hoc variant of PM-LoRA is close to the progressive version (90.31 vs. 90.15 FA on VTAB and 77.35 vs. 78.17 on CUB200), so the margin over the other merging methods comes mainly from training each update for compatibility with consolidation; progressive merging additionally removes the need to store task adapters. Additionally, we further compare

Table 4: Ablation of regularization and merging strategies.

Reg.	Merge	CIFAR-100 ($N = 10$) FA	CIFAR-100 ($N = 10$) CA	ImageNet-R ($N = 40$) FA	ImageNet-R ($N = 40$) CA
OPCM	SPM + β scaling	87.23	91.15	70.84	74.25
SCR	MagMax	85.24	88.28	69.15	73.44
SCR	Fisher	87.91	91.87	71.36	74.59
PM-LoRA		**91.55**	**96.36**	**76.37**	**79.26**

Table 5: Comparison of efficiency and scalability of LoRA-based CL methods on ImageNet-R ($N = 20$). We used the same rank for training all methods ($r = 16$). PM-LoRA achieves constant memory and low complexity under a fixed rank budget ($R_{\max} = 64$), requiring only one merged adapter for all tasks without storing task-specific LoRA adapters, further demonstrating PM-LoRA's scalability across long horizon CL.

Method	Adapter GFLOPs	Stored Features (M)	Learnable Parameters (M)	Inference LoRAs (M)	Scalability Complexity
InfLoRA	35.12	0.10	0.37	5.4	$\mathcal{O}(1)$
C-LoRA	20.57	0	0.55	1.7	$\mathcal{O}(1)$
SD-LoRA	35.12	0	0.37	5.4	$\mathcal{O}(T)$
CL-LoRA	37.03	0	0.37	5.9	$\mathcal{O}(T)$
PM-LoRA (Ours)	**7.31**	0	0.37	**1.5**	$\boldsymbol{\mathcal{O}(1)}$

PM-LoRA with OPCM [33], a post-hoc merging approach that merges all task-specific models by introducing models to the merging stage progressively. Our PM-LoRA, which continually integrates and preserves knowledge during training, still outperforms OPCM. The same trend holds on CUB200, where PM-LoRA again leads all other approaches, validating our progressive merging strategy during LoRA training.

Component Analysis. We next dissect the contribution of each individual component in PM-LoRA, as detailed in Table 3. We analyze the Spectral Compatibility Regularizer (SCR), Scaled Progressive Merging (SPM), and β-scaling. A baseline model with only naive merging shows poor performance, achieving just 76.25% FA on ImageNet-R ($N = 10$). While adding any single component provides a modest benefit, the true strength of our method is revealed in its synergy. The full PM-LoRA model, with all three components enabled, achieves 87.37% FA. This represents a massive gain of +11.12% over the baseline, which proves that all three components are critical. The SCR solves spectral interference, the SPM solves magnitude domination, and the β-scaling bridges the consistency between these two components. The gain becomes larger as the task horizon increases, suggesting that SCR and SPM are especially important after the merged adapter approaches its rank budget.

Moreover, PM-LoRA's advantage becomes increasingly pronounced as task sequences grow longer. On a 50-task ImageNet-R benchmark, the complete system delivers an impressive +17.89% improvement over naive merging, demonstrating its robustness and scalability under long-horizon CL. We further conduct additional cross-component ablations with OPCM [33], MagMax [25], and Fisher [26] in Table 4. PM-LoRA consistently achieves the best accuracy under $O(1)$ inference. All these results highlight the ability of PM-LoRA to maintain spectral diversity and balanced task importance even after dozens of sequential merges, which prior LoRA-based approaches fail to achieve due to accumulated interference and rank saturation (see Appendix C for more ablations).

To further analyze our merging method, we investigate the sensitivity of the SCR's hyperparameters, λ_A and λ_B. The heatmaps in Fig. 3 visualize the final average accuracy, and it is clear that the optimal performance is not at zero. For CIFAR-100 ($N = 10$), the peak accuracy of 91.55% is achieved at $\lambda_A = 0.4$ and $\lambda_B = 0.5$. This confirms our hypothesis that actively forcing orthogonality is crucial for a low-loss merge. Without our regularizer, our model results in significant spectral interference and demonstrably lower accuracy, empirically validating the necessity of the SCR.

Scalability Analysis. Finally, we demonstrate the exceptional and defining scalability advantage of PM-LoRA. As shown in Table 5, other LoRA-based CL methods like SD-LoRA and CL-LoRA are fundamentally accumulation methods. They must store every previously learned adapter and process all of them during the forward pass, causing their storage cost to grow linearly with the number of tasks, $\mathcal{O}(T)$. Even more critically, their inference cost also scales linearly, $\mathcal{O}(T)$, because they must compute the sum of all T adapters for every forward pass. InfLoRA also supports task-count-independent inference without retaining all historical LoRA branches. Our comparison therefore distinguishes scaling with task count from the computation and persistent state required by each method.

PM-LoRA follows a merge-then-compress framework that performs rank-constrained consolidation after each task. By projecting updates onto a bounded rank space, the method maintains constant storage and $\mathcal{O}(1)$ inference cost (Table 5). We select the rank budget $R_{\max} \in \{16, 32, 48, 64, 128\}$ based on validation performance, and use $R_{\max} = 64$ unless otherwise stated. The effective adapter

Table 6: **Component replacements and compression strategies.** ΔFA is relative to PM-LoRA. Dashes indicate unreported ranks.

Variant	FA ↑	ΔFA	Final rank
OPCM + proposed consolidation	70.84	−5.53	–
SCR + MagMax	69.15	−7.22	–
SCR + Fisher	71.36	−5.01	–
Unrestricted SCR + scaled accumulation	77.11	+0.74	640
Naive rank-64 truncation	63.33	−13.04	64
PM-LoRA	**76.37**	0.00	**64**

rank increases only when new spectral energy is introduced but is always capped by $R_{\max}$, ensuring constant parameter complexity regardless of the number of tasks (see Appendix C for more analysis).

We examine whether PM-LoRA benefits from coupling its training objective to bounded consolidation, how this coupling affects plasticity over long task sequences, and where its accuracy and efficiency advantages hold. FA denotes final average accuracy, and FM denotes forgetting; accuracy and forgetting scores are reported as percentages.

Coupling regularization to the persistent state. Our contribution lies in what we regularize and how this objective is coupled to consolidation. O-LoRA [37] retains task-dependent orthogonal updates, CODA-Prompt [30] composes prompt components, and InfLoRA [22] protects prior knowledge through task-dependent gradient or feature subspaces and auxiliary statistics. We instead regularize a temporary update against the current compressed adapter, into which that update is immediately consolidated. The persistent adapter remains bounded by a fixed rank budget, set to 64 in the experiments below, independently of the number of tasks. SCR therefore encourages compatibility with the state that will persist after consolidation, rather than allocating an expanding collection of task-specific subspaces.

Table 6 evaluates this coupling by replacing either component. OPCM with our consolidation reaches 70.84 FA, while SCR with MagMax or Fisher merging reaches 69.15 and 71.36, respectively, compared with 76.37 for PM-LoRA. These replacements fall short by 5.01–7.22 points, supporting the importance of the combined design.

Compatibility before fixed-rank compression. We use standard truncated SVD, and the contribution is the coupling of task optimization to subsequent compression. Unrestricted SCR with scaled accumulation reaches 77.11 FA at rank 640. PM-LoRA reaches 76.37 at rank 64, a gap of only 0.74 points at one tenth of the rank, retaining 99.0 % of the unrestricted accuracy. Naive rank-64 truncation reaches only 63.33 FA. The 13.04-point advantage over naive truncation shows that applying SVD alone does not reproduce our result and supports training for compatibility before consolidation. We provide more details in **Appendix**.

5 Conclusion

In this paper, we propose PM-LoRA, a progressive adapter merging framework for CL that maintains a single evolving LoRA adapter throughout training. By training task updates to be consolidation-compatible and integrating them through rank-constrained merging, PM-LoRA enables continual knowledge accumulation under a fixed adapter capacity without storing task-specific adapters or rehearsal data. Experiments on multiple class-incremental benchmarks demonstrate that PM-LoRA achieves strong performance while maintaining constant memory and low computational overhead. An interesting future direction is extending this consolidation framework to large-scale foundation models. In particular, adapting progressive adapter consolidation to vision–language models (VLMs) and large language models (LLMs) could enable scalable lifelong adaptation in settings with long task streams and strict memory constraints. We hope PM-LoRA provides a useful step toward scalable continual learning for future foundation models.

Acknowledgments and Disclosure of Funding

This study was supported in part by NIH grants R21EB034911 and the NVIDIA Academic Grant Program. The authors declare no competing interests.

References

[1] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *ECCV*, pages 139–154, 2018.

[2] Prashant Shivaram Bhat, Shakib Yazdani, Elahe Arani, and Bahram Zonooz. Parameter efficient continual learning with dynamic low-rank adaptation. *Transactions on Machine Learning Research*, 2026. URL `https://openreview.net/forum?id=ZqQATq0Geg`.

[3] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *ICLR*, 2018.

[4] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[6] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[7] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[8] Takuma Fukuda, Hiroshi Kera, and Kazuhiko Kawamoto. Adapter merging with centroid prototype mapping for scalable class-incremental learning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 4884–4893, 2025.

[9] Jiangpeng He. Gradient reweighting: Towards imbalanced class-incremental learning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16668–16677, 2024.

[10] Jiangpeng He, Runyu Mao, Zeman Shao, and Fengqing Zhu. Incremental learning in online scenario. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13926–13935, 2020.

[11] Jiangpeng He, Zhihao Duan, and Fengqing Zhu. Cl-lora: Continual low-rank adaptation for rehearsal-free class-incremental learning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 30534–30544, 2025.

[12] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, Dawn Song, Jacob Steinhardt, and Justin Gilmer. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[13] 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 *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

[14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1 (2):3, 2022.

[15] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

[16] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[18] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

[19] Mei Li, Yuxiang Lu, Qinyan Dai, Suizhi Huang, Yue Ding, and Hongtao Lu. Became: Bayesian continual learning with adaptive model merging. *arXiv preprint arXiv:2504.02666*, 2025.

[20] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[21] Zhizhong Li and Derek Hoiem. Learning without forgetting. *TPAMI*, 40(12):2935–2947, 2017.

[22] Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. In *CVPR*, pages 23638–23647, 2024.

[23] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Neurips*, 30, 2017.

[24] Xin Luo, Yunni Xia, and Qingsheng Zhu. Incremental collaborative filtering recommender based on regularized matrix factorization. *Knowledge-Based Systems*, 27:271–280, 2012.

[25] Daniel Marczak, Bartłomiej Twardowski, Tomasz Trzciński, and Sebastian Cygert. Magmax: Leveraging model merging for seamless continual learning. In *ECCV*, pages 379–395. Springer, 2024.

[26] Michael S Matena and Colin A Raffel. Merging models with fisher-weighted averaging. *Advances in Neural Information Processing Systems*, 35:17703–17716, 2022.

[27] Mark D. McDonnell, Dong Gong, Amin Parvaneh, Ehsan Abbasnejad, and Anton van den Hengel. RanPAC: Random projections and pre-trained models for continual learning. In *Advances in Neural Information Processing Systems*, 2023.

[28] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[29] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *CVPR*, pages 2001–2010, 2017.

[30] James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *CVPR*, pages 11909–11919, 2023.

[31] George Stoica, Pratik Ramesh, Boglarka Ecsedi, Leshem Choshen, and Judy Hoffman. Model merging with svd to tie the knots. *arXiv preprint arXiv:2410.19735*, 2024.

[32] Hai-Long Sun, Da-Wei Zhou, Hanbin Zhao, Le Gan, De-Chuan Zhan, and Han-Jia Ye. Mos: Model surgery for pre-trained model-based class-incremental learning. In *AAAI*, volume 39, pages 20699–20707, 2025.

[33] Anke Tang, Enneng Yang, Li Shen, Yong Luo, Han Hu, Bo Du, and Dacheng Tao. Merging models on the fly without retraining: A sequential approach to scalable continual model merging. *arXiv preprint arXiv:2501.09522*, 2025.

[34] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[35] Huiyi Wang, Haodong Lu, Lina Yao, and Dong Gong. Self-expansion of pre-trained models with mixture of adapters for continual learning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 10087–10098, 2025.

[36] Liyuan Wang, Jingyi Xie, Xingxing Zhang, Mingyi Huang, Hang Su, and Jun Zhu. Hierarchical decomposition of prompt-based continual learning: Rethinking obscured sub-optimality. *Neurips*, 36, 2024.

[37] Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

[38] Yabin Wang, Zhiwu Huang, and Xiaopeng Hong. S-prompts learning with pre-trained transformers: An occam's razor for domain incremental learning. *Advances in Neural Information Processing Systems*, 35:5682–5695, 2022.

[39] Zhen Wang, Liu Liu, Yiqun Duan, Yajing Kong, and Dacheng Tao. Continual learning with lifelong vision transformer. In *CVPR*, pages 171–181, 2022.

[40] Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, et al. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *ECCV*, pages 631–648. Springer, 2022.

[41] Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *CVPR*, pages 139–149, 2022.

[42] Yichen Wu, Hongming Piao, Long-Kai Huang, Renzhen Wang, Wanhua Li, Hanspeter Pfister, Deyu Meng, Kede Ma, and Ying Wei. Sd-lora: Scalable decoupled low-rank adaptation for class incremental learning. *ICLR*, 2025.

[43] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36:7093–7115, 2023.

[44] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *CVPR*, pages 3014–3023, 2021.

[45] Xiaobing Yu, Jin Yang, Xiao Wu, Peijie Qiu, and Xiaofeng Liu. Fm-lora: Factorized low-rank meta-prompting for continual learning. In *CVPR Workshop*, pages 6409–6418, 2025.

[46] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, pages 3987–3995. PMLR, 2017.

[47] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, Lucas Beyer, Olivier Bachem, Michael Tschannen, Marcin Michalski, Olivier Bousquet, Sylvain Gelly, and Neil Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[48] Xiaoyan Zhang and Jiangpeng He. One adapter for all: Towards unified representation in step-imbalanced class-incremental learning. *arXiv preprint arXiv:2603.10237*, 2026.

[49] Xin Zhang, Liang Bai, Xian Yang, and Jiye Liang. C-lora: Continual low-rank adaptation for pre-trained models. *arXiv preprint arXiv:2502.17920*, 2025.

[50] Da-Wei Zhou, Hai-Long Sun, Han-Jia Ye, and De-Chuan Zhan. Expandable subspace ensemble for pre-trained model-based class-incremental learning. In *CVPR*, pages 23554–23564, 2024.

[51] Da-Wei Zhou, Zi-Wen Cai, Han-Jia Ye, De-Chuan Zhan, and Ziwei Liu. Revisiting class-incremental learning with pre-trained models: Generalizability and adaptivity are all you need. *International Journal of Computer Vision*, 133(3):1012–1032, 2025.

A Theoretical Analysis

This section provides a theoretical analysis of PM-LoRA.

A.1 Justification of the Spectral Compatibility Regularizer

We first formalize spectral interference between the accumulated adapter ΔW_m and the new adapter ΔW_t.

Definition 1 (Spectral interference)**.** *The Frobenius inner product between ΔW_m and ΔW_t is*

$$\langle \Delta W_m, \Delta W_t \rangle_F = \mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big). \tag{12}$$

The spectral interference between the two adapters is defined as

$$\mathcal{I}(\Delta W_m, \Delta W_t) = |\langle \Delta W_m, \Delta W_t \rangle_F| = \big|\mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big)\big|. \tag{13}$$

A value of $\mathcal{I}(\Delta W_m, \Delta W_t)$ close to zero indicates that the adapters are nearly orthogonal and cause minimal interference when combined.

Using the low-rank factorizations in Method section, the interference term becomes

$$\mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big) = \mathrm{tr}\big((B_m A_m)^\top (B_t A_t)\big). \tag{14}$$

By expanding this expression we obtain

$$\mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big) = \mathrm{tr}\big(A_m^\top B_m^\top B_t A_t\big). \tag{15}$$

Define the factor cross-correlation matrices

$$X = B_m^\top B_t \in \mathbb{R}^{R_{\max} \times r_t}, Y = A_t A_m^\top \in \mathbb{R}^{r_t \times R_{\max}}. \tag{16}$$

Using cyclicity of trace we have

$$\mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big) = \mathrm{tr}(XY). \tag{17}$$

Proposition 2 (Interference upper bound in factor space.)**.** *For ΔW_m and ΔW_t defined in Method section, the interference satisfies*

$$\big|\mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big)\big| \le \big\|B_m^\top B_t\big\|_F \big\|A_t A_m^\top\big\|_F. \tag{18}$$

Proof. From (16) and (17), the Frobenius inner product representation yields

$$\mathrm{tr}(XY) = \big\langle X, Y^\top \big\rangle_F. \tag{19}$$

The Cauchy Schwarz inequality for the Frobenius inner product gives

$$|\mathrm{tr}(XY)| \le \|X\|_F \, \|Y^\top\|_F. \tag{20}$$

which implies

$$\big|\mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big)\big| \le \|B_m^\top B_t\|_F \, \|A_t A_m^\top\|_F. \tag{21}$$

This is exactly (18). □

The Spectral Compatibility Regularizer used in Stage 1 is

$$R_{\text{spec}} = \lambda_A \|A_t A_m^\top\|_F^2 + \lambda_B \|B_m^\top B_t\|_F^2, \lambda_A > 0, \lambda_B > 0. \tag{22}$$

which penalizes correlations between the input and output factor subspaces of the two adapters.

Lemma 1 (Zero regularizer implies zero interference.)**.** *If $R_{spec} = 0$, then $\mathcal{I}(\Delta W_m, \Delta W_t) = 0$.*

Proof. From (22), $R_{\text{spec}} = 0$ and $\lambda_A, \lambda_B > 0$ imply

$$A_t A_m^\top = 0, \qquad B_m^\top B_t = 0. \tag{23}$$

Substituting (23) into (18) yields

$$\big|\mathrm{tr}\big(\Delta W_m^\top \Delta W_t\big)\big| \le \|B_m^\top B_t\|_F \, \|A_t A_m^\top\|_F = 0. \tag{24}$$

and therefore

$$\mathcal{I}(\Delta W_m, \Delta W_t) = 0. \tag{25}$$

□

Even away from the ideal case, R_{spec} directly controls the magnitude of interference.

Corollary 1 (Regularization controls interference.)**.** *There exists a constant $C > 0$ depending only on λ_A and λ_B such that*

$$\left|\operatorname{tr}\left(\Delta W_m^\top \Delta W_t\right)\right| \leq C\, R_{spec}. \tag{26}$$

Proof. Define

$$u = \sqrt{\lambda_A}\|A_t A_m^\top\|_F, \qquad v = \sqrt{\lambda_B}\|B_m^\top B_t\|_F. \tag{27}$$

Then

$$R_{\text{spec}} = u^2 + v^2. \tag{28}$$

Using (18) and (27) we obtain

$$\left|\operatorname{tr}\left(\Delta W_m^\top \Delta W_t\right)\right| \leq \frac{uv}{\sqrt{\lambda_A \lambda_B}}. \tag{29}$$

The arithmetic-geometric inequality gives

$$2uv \leq u^2 + v^2 = R_{\text{spec}}, \qquad uv \leq \frac{1}{2} R_{\text{spec}}. \tag{30}$$

Combining (29) and (30) yields

$$\left|\operatorname{tr}\left(\Delta W_m^\top \Delta W_t\right)\right| \leq \frac{1}{2\sqrt{\lambda_A \lambda_B}}\, R_{\text{spec}}. \tag{31}$$

Setting

$$C = \left(2\sqrt{\lambda_A \lambda_B}\right)^{-1}. \tag{32}$$

in (31) gives (26). □

Because the transformation

$$(B_m, A_m) \mapsto (B_m S, S^{-1} A_m), (B_t, A_t) \mapsto (B_t T, T^{-1} A_t). \tag{33}$$

for invertible S and T leaves $A_t A_m^\top$ and $B_m^\top B_t$ unchanged, the regularizer in (22) depends only on the subspaces spanned by the factors. Minimizing R_{spec} therefore encourages the input and output subspaces of ΔW_t to be nearly orthogonal to those of ΔW_m, which is precisely what is needed to reduce interference.

A.2 Analysis of catastrophic forgetting

We now relate interference and merging to the change in the function implemented on previous tasks. Before merging, the linearized mapping at this layer is

$$f_m(x) = (W_0 + \Delta W_m)x. \tag{34}$$

After merging, the mapping becomes

$$f_{\text{new}}(x) = (W_0 + \Delta W_{\text{new}})x. \tag{35}$$

Since W_0 is fixed, the change in the output is entirely determined by the change in the adapter.

Definition 2 (Forgetting error at one layer)**.** *For an input x from a previous task, the forgetting error at this layer after merging task t is*

$$\mathcal{E}_{forget}(x) = \left\|(\Delta W_{new} - \Delta W_m)x\right\|_2. \tag{36}$$

In Stage 2 of PM-LoRA, the two adapters are first scaled and summed. The Frobenius norms of the adapters are

$$\alpha_m = \|\Delta W_m\|_F, \qquad \alpha_t = \|\Delta W_t\|_F, \tag{37}$$

and the scaling factor is defined as

$$\beta = \frac{\|\Delta W_m\|_F}{\|\Delta W_t\|_F} = \frac{\alpha_m}{\alpha_t}. \tag{38}$$

The scaled sum of the two adapters is

$$W_{\text{sum}} = \Delta W_m + \beta\, \Delta W_t. \tag{39}$$

The merged adapter is obtained by projecting this sum back to a rank $r_m \leq R_{\max}$ through truncated singular value decomposition,

$$\Delta W_{\text{new}} = \mathcal{T}_{r_m}(W_{\text{sum}}). \tag{40}$$

where $\mathcal{T}_{r_m}$ denotes the best rank r_m approximation in Frobenius norm. The truncation residual is

$$E_{\text{trunc}} = W_{\text{sum}} - \Delta W_{\text{new}} = \Delta W_m + \beta\, \Delta W_t - \Delta W_{\text{new}}. \tag{41}$$

Substituting (39) and (41) into (36) gives

$$\mathcal{E}_{\text{forget}}(x) = \left\| (\beta\, \Delta W_t - E_{\text{trunc}})x \right\|_2. \tag{42}$$

Using the triangle inequality we obtain

$$\mathcal{E}_{\text{forget}}(x) \leq \beta\, \|\Delta W_t x\|_2 + \|E_{\text{trunc}} x\|_2. \tag{43}$$

The first term is functional interference of the new adapter evaluated on old task features. The second term is the compression error induced by the rank constraint.

Proposition 3 (Ideal orthogonality eliminates functional interference.)**.** *Let $\Delta W_t = B_t A_t$ and $\Delta W_m = B_m A_m$ with $A_m \in \mathbb{R}^{R_{\max} \times d_{in}}$ and $A_t \in \mathbb{R}^{r_t \times d_{in}}$. Consider any input $x \in \mathbb{R}^{d_{in}}$ for which there exists a vector z such that*

$$x = A_m^\top z. \tag{44}$$

Suppose that the factor cross term satisfies

$$A_t A_m^\top = 0. \tag{45}$$

Then, for every such x, the new adapter vanishes on x:

$$\Delta W_t x = 0, \tag{46}$$

and the forgetting error obeys

$$\mathcal{E}_{\textit{forget}}(x) \leq \|E_{\textit{trunc}} x\|_2. \tag{47}$$

Proof. From (44) and the definition $\Delta W_t = B_t A_t$ we have

$$\Delta W_t x = B_t A_t A_m^\top z. \tag{48}$$

Using (45) gives

$$\Delta W_t x = B_t \cdot 0 = 0, \tag{49}$$

which proves (46). Substituting (46) into the decomposition

$$\mathcal{E}_{\text{forget}}(x) \leq \beta\, \|\Delta W_t x\|_2 + \|E_{\text{trunc}} x\|_2, \tag{50}$$

which is (43), yields (47). □

The next result provides a quantitative bound on functional interference without requiring exact orthogonality, again as a purely linear algebraic statement.

Proposition 4 (Functional interference bound in terms of R_{spec})**.** *Let $\Delta W_t = B_t A_t$ and $\Delta W_m = B_m A_m$ as above. Assume that A_m has full row rank and that its smallest singular value satisfies*

$$\sigma_{\min}(A_m) > 0. \tag{51}$$

Consider any input $x \in \mathbb{R}^{d_{in}}$ that can be written as

$$x = A_m^\top z. \tag{52}$$

for some z. Then there exist nonnegative constants C_1, depending only on B_t and A_m, and C_2, depending on B_t, A_m, and λ_A, such that

$$\|\Delta W_t x\|_2 \leq C_1 \|A_t A_m^\top\|_F\, \|x\|_2 \leq C_2 \sqrt{R_{\textit{spec}}}\, \|x\|_2. \tag{53}$$

where $R_{\textit{spec}}$ is defined in (22)*.*

Proof. Using (52) and $\Delta W_t = B_t A_t$, we have

$$\Delta W_t x = B_t A_t A_m^\top z. \tag{54}$$

By submultiplicativity of operator norms,

$$\|\Delta W_t x\|_2 \le \|B_t\|_2 \, \|A_t A_m^\top\|_2 \, \|z\|_2 \le \|B_t\|_2 \, \|A_t A_m^\top\|_F \, \|z\|_2. \tag{55}$$

Because A_m has full row rank and (51) holds, the mapping $z \mapsto A_m^\top z$ is injective on its domain and satisfies

$$\|z\|_2 \le \frac{1}{\sigma_{\min}(A_m)} \, \|A_m^\top z\|_2 = \frac{1}{\sigma_{\min}(A_m)} \, \|x\|_2. \tag{56}$$

Substituting (56) into (55) yields

$$\|\Delta W_t x\|_2 \le \frac{\|B_t\|_2}{\sigma_{\min}(A_m)} \, \|A_t A_m^\top\|_F \, \|x\|_2. \tag{57}$$

Define

$$C_1 = \frac{\|B_t\|_2}{\sigma_{\min}(A_m)}. \tag{58}$$

which gives the first inequality in (53). For the second inequality, recall the definition from the Method section:

$$R_{\text{spec}} = \lambda_A \, \|A_t A_m^\top\|_F^2 + \lambda_B \, \|B_m^\top B_t\|_F^2. \tag{59}$$

so in particular

$$\|A_t A_m^\top\|_F \le \sqrt{\frac{R_{\text{spec}}}{\lambda_A}}. \tag{60}$$

Combining (57) and (60) and setting

$$C_2 = \frac{C_1}{\sqrt{\lambda_A}}. \tag{61}$$

proves (53). □

The following theorem is then a direct consequence of the decomposition of the forgetting error and the previous proposition.

Theorem 1 (Forgetting bound for a single merge)**.** *Assume the conditions of Proposition 4 hold and let x satisfy* (52)*. Then there exist constants $C_2 > 0$ and $C_3 > 0$ such that*

$$\mathcal{E}_{\textit{forget}}(x) \le \left(C_2 \beta \sqrt{R_{\textit{spec}}} + C_3 \|E_{\textit{trunc}}\|_F \right) \|x\|_2. \tag{62}$$

Proof. Starting from the decomposition in (43),

$$\mathcal{E}_{\text{forget}}(x) \le \beta \, \|\Delta W_t x\|_2 + \|E_{\text{trunc}} x\|_2, \tag{63}$$

apply (53) to the first term to obtain

$$\beta \, \|\Delta W_t x\|_2 \le \beta C_2 \sqrt{R_{\text{spec}}} \, \|x\|_2. \tag{64}$$

For the second term, use the operator norm inequality

$$\|E_{\text{trunc}} x\|_2 \le \|E_{\text{trunc}}\|_2 \, \|x\|_2 \le \|E_{\text{trunc}}\|_F \, \|x\|_2. \tag{65}$$

Combining (64) and (65) inside (63) gives

$$\mathcal{E}_{\text{forget}}(x) \le \left(C_2 \beta \sqrt{R_{\text{spec}}} + \|E_{\text{trunc}}\|_F \right) \|x\|_2. \tag{66}$$

Setting

$$C_3 = 1 \tag{67}$$

yields (62). □

Finally, by the Eckart–Young–Mirsky theorem, the truncation operator $\mathcal{T}_{r_m}$ used in

$$\Delta W_{\text{new}} = \mathcal{T}_{r_m}(W_{\text{sum}}) \tag{68}$$

gives a matrix that minimizes

$$\|E_{\text{trunc}}\|_F^2 = \|W_{\text{sum}} - \Delta W_{\text{new}}\|_F^2 \tag{69}$$

among all approximations of W_{sum} with rank at most r_m; this minimizer is unique when $\sigma_{r_m}(W_{\text{sum}}) > \sigma_{r_m+1}(W_{\text{sum}})$. Thus, for any fixed rank budget r_m, PM-LoRA chooses the merged adapter that minimizes the compression term in (62), while the regularizer R_{spec} controls the interference term through (53) and (26).

A.3 Rank compression as spectral regularization

The rank constraint in PM-LoRA acts as a spectral regularizer that denoises task-specific updates and stabilizes continual learning. For the current task t, decompose the new adapter into a signal component and a noise component:

$$\Delta W_t = S_t + N_t, \tag{70}$$

where S_t captures shared, generalizable directions and N_t represents high variance, task-specific fluctuations. In high-dimensional settings it is natural to assume that the dominant singular values of ΔW_t correspond to S_t while N_t is concentrated in the trailing singular directions.

The scaled sum before truncation is

$$W_{\text{sum}} = \Delta W_m + \beta(S_t + N_t). \tag{71}$$

The merged adapter is

$$\Delta W_{\text{new}} = \mathcal{T}_{r_m}\big(\Delta W_m + \beta(S_t + N_t)\big). \tag{72}$$

and under the assumption that singular values associated with S_t dominate those of N_t, the effect of $\mathcal{T}_{r_m}$ is approximately

$$\Delta W_{\text{new}} \approx \mathcal{T}_{r_m}\big(\Delta W_m + \beta S_t\big). \tag{73}$$

Consequently, the effective update relative to ΔW_m satisfies

$$\Delta W_{\text{new}} - \Delta W_m \approx \beta S_t. \tag{74}$$

that is, the merged adapter mainly incorporates the shared signal and suppresses a large fraction of the noise.

If adapters were accumulated additively without compression, the adapter after T tasks would be

$$\Delta W_{\text{add}}^{(T)} = \sum_{t=1}^{T} \beta_t \Delta W_t = \sum_{t=1}^{T} \beta_t S_t + \sum_{t=1}^{T} \beta_t N_t. \tag{75}$$

and the term

$$\sum_{t=1}^{T} \beta_t N_t. \tag{76}$$

would typically grow with T, leading to drift and degradation of performance. In contrast, PM-LoRA maintains a recursively merged adapter

$$\Delta W_{\text{merge}}^{(t+1)} = \mathcal{T}_{r_m^{(t+1)}}\big(\Delta W_{\text{merge}}^{(t)} + \beta_{t+1}\Delta W_{t+1}\big), \Delta W_{\text{merge}}^{(0)} = 0. \tag{77}$$

where $r_m^{(t+1)} \leq R_{\max}$ at each step. Each merge suppresses a large portion of the task-specific noise introduced at that step while preserving the dominant shared structure.

From a capacity perspective, the fixed budget $R_{\max}$ imposes an explicit limit on the dimensionality of the adapter subspace, and the regularizer R_{spec} in (22) governs how this limited capacity is allocated across tasks. The analysis above shows that this design simultaneously limits interference through subspace orthogonality, minimizes the forgetting bound in (62) for a given rank through optimal truncation, and suppresses task-specific noise that would otherwise accumulate. This explains why PM-LoRA can retain strong performance in class incremental learning while storing and using only a single merged adapter.

We further demonstrate clear comparisons between LoRA-based CL methods, Mode Merging-based CL methods, and our proposed PM-LoRA in Fig 4 to further demonstrate our advantages compared to those 2 kinds of methods. We also illustrate how our PM-LoRA works under CL scenario in Algorithm 1.

B Implementation Details

B.1 Experimental Setup and Architecture

We implement PM-LoRA in PyTorch 2.1 and run all experiments on NVIDIA A100 GPUs with 40GB of VRAM. The backbone is a Vision Transformer (ViT-B/16) pre-trained on ImageNet-21K, following common practice in recent continual learning work.

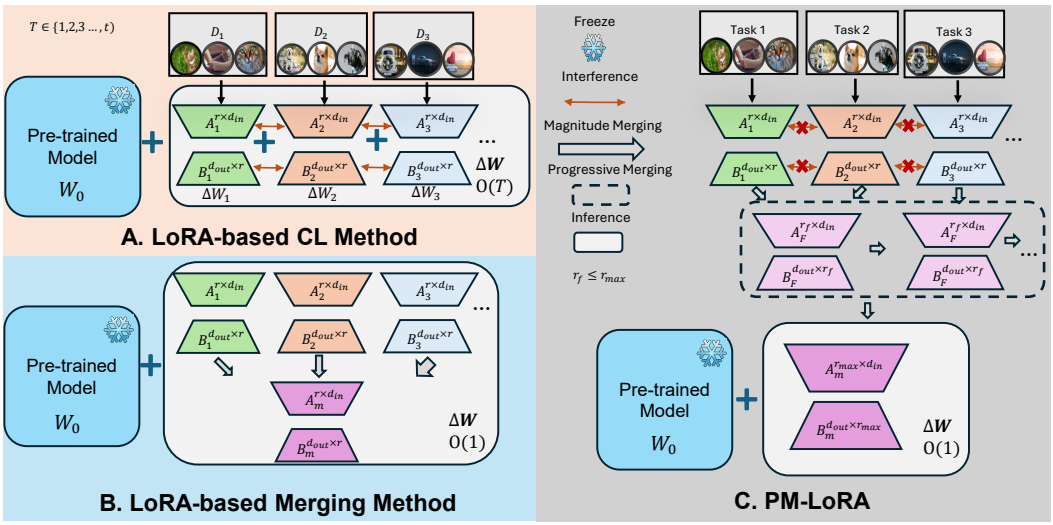


Figure 4: **Comparison of LoRA-based continual learning strategies.** (A) Traditional CL methods allocate a new adapter per task, causing linear growth in parameters. (B) LoRA merging methods combine task adapters post hoc but require storing all adapters during training. (C) PM-LoRA performs progressive merge-then-compress consolidation, maintaining a single bounded adapter with constant memory and $\mathcal{O}(1)$ inference cost.

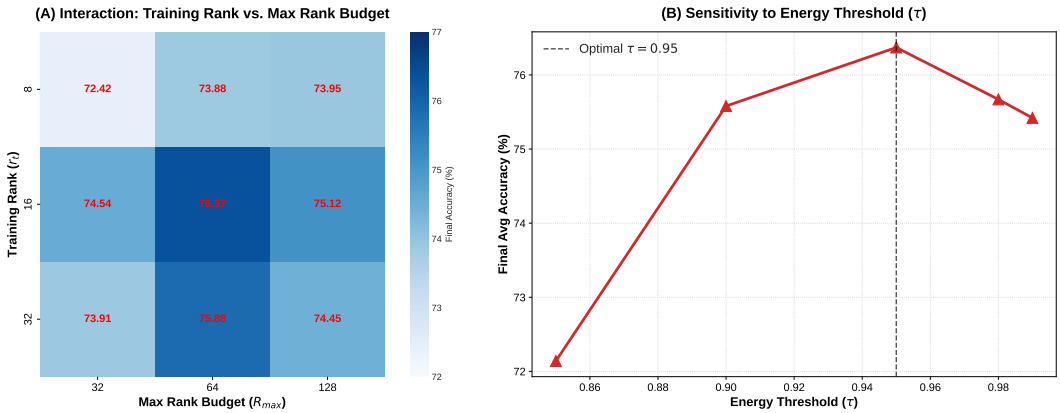


Figure 5: **Sensitivity of PM-LoRA to rank budget and energy threshold.** (A) Interaction between the training rank r_t and the maximum merged rank budget $R_{\max}$ on Split ImageNet-R with $N = 40$ tasks (ViT–B/16 backbone). Each cell reports the final average accuracy (FA, %) after the last task. Performance is stable across a broad range of settings and peaks at $r_t = 16$, $R_{\max} = 64$ (76.37%), indicating that a moderate, fixed budget is sufficient and that larger $R_{\max}$ does not provide additional gains. (B) Sensitivity of FA to the spectral energy threshold τ used by the adaptive rank rule in Stage 2. Accuracy increases sharply when moving from aggressive compression to $\tau = 0.90$, reaches its maximum around $\tau = 0.95$, and degrades only mildly for larger thresholds. This shows that PM-LoRA is robust to the choice of τ and justifies the default setting $\tau = 0.95$.

Classifier head and CL protocol. We adopt a single unified classifier head for all tasks. At the first task, the linear head is initialized for the corresponding classes. When a new task arrives, we expand the head by appending new rows for the new classes, initialized with Kaiming Uniform initialization, while keeping the existing rows trainable. At inference time, we perform an $\arg\max$ over the logits of all classes seen so far, without access to task identifiers, corresponding to the standard rehearsal-free class-incremental learning protocol.

LoRA configuration. All pre-trained backbone weights are frozen. We insert LoRA adapters only in the `Query` and `Value` projection matrices of every MHSA block. For every new task we use a training rank $r = 16$, which we found to provide a good trade-off between capacity and efficiency.

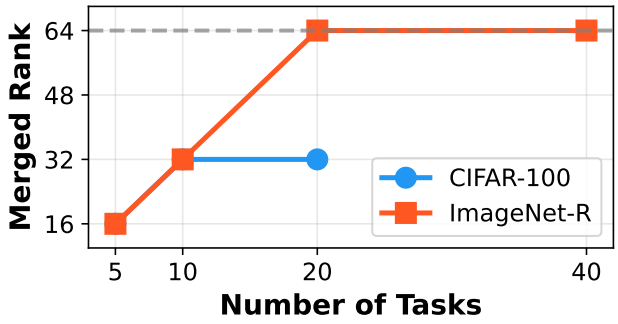


Figure 6: **Ablation study for selecting the rank budget $R_{\max}$.** The merged adapter rank increases as new tasks introduce additional spectral information but remains bounded by the predefined budget. Performance saturates when the rank reaches 64, so we select $R_{\max} = 64$ as a good trade-off between model capacity and parameter efficiency.

Each adapter is parameterized by low-rank factors $A_i \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B_i \in \mathbb{R}^{d_{\text{out}} \times r}$. We initialize A_i with Kaiming Uniform initialization and B_i to zero so that the initial adapter update is zero and the model starts each task from the current accumulated model, including the merged adapter. We use the standard LoRA scaling with α/r (with $\alpha = 16$) on the adapter output.

B.2 Training Strategy and Data Handling

Each task is trained for 20 epochs. We use the Adam optimizer with a learning rate of 1×10^{-4} and weight decay of 0.1. To stabilize training, we apply gradient clipping with a maximum ℓ_2 norm of 1.0. The loss for PM-LoRA is the sum of the standard cross-entropy loss over the unified head and the Spectral Compatibility Regularizer applied to all LoRA layers.

For data preprocessing, all images are resized to 224×224 pixels and normalized using ImageNet statistics with mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$. We use a batch size of 128 for CIFAR-100 and ImageNet-R, and 64 for CUB-200. During training we apply random horizontal flipping with probability 0.5 and random cropping (or random resized crop) to 224×224 with a scale range of roughly $[0.8, 1.0]$. No data augmentation is used during evaluation.

Task splits. For Split CIFAR-100 we partition the 100 classes into 10 disjoint tasks of 10 classes each. For ImageNet-R and CUB-200 we create 40 and 20 tasks respectively. In all cases the class order is determined by a single random permutation with seed 42 and the same task order is used for all methods.

B.3 PM-LoRA Hyperparameters

Unless otherwise stated, we fix the following PM-LoRA hyperparameters.

Spectral Compatibility Regularizer (SCR). The regularization strengths on the input and output factor subspaces are $(\lambda_A, \lambda_B) = (0.4, 0.5)$ for Split CIFAR-100 (10 tasks), $(0.6, 0.5)$ for ImageNet-R (40 tasks), and $(0.5, 0.5)$ for CUB-200 (20 tasks). These values are selected via a small grid search on a held-out validation split for each dataset.

Merge budget and energy threshold. To keep memory usage constant, we cap the rank of the merged adapter at $R_{\max} = 64$ for all datasets. After finishing training on task t, we apply Scaled Progressive Merging once: we scale the new adapter by $\beta = \|\Delta W_m\|_F / \|\Delta W_t\|_F$, add it to the current merged adapter, and compute an SVD. The new merged adapter keeps the smallest rank whose leading singular values retain at least $\tau = 0.95$ of the total spectral energy, subject to the cap $R_{\max}$. The temporary task-specific adapter ΔW_t is discarded after merging, so only a single merged adapter is stored across tasks.

Scaling factor at initialization. For the first task, the merged adapter is zero, so we set $\beta_1 = 1$. Because B_t is initialized to zero, $\|\Delta W_t\|_F = 0$ at the start of each task; we therefore compute $\beta_t = \|\Delta W_m\|_F / (\|\Delta W_t\|_F + \epsilon)$ with $\epsilon = 10^{-8}$.

PTM	Method	CIFAR-100 ($N = 10$)			Imagenet-R ($N = 40$)		
		FA (↑)	**CA**(↑)	**FM**(↓)	**FA** (↑)	**CA**(↑)	**FM**(↓)
ViT-L/14 [5]	HiDe-Prompt	**91.44** ± 0.11	**96.31** ± 0.02	**1.67** ± 0.29	69.39 ± 0.73	74.70 ± 0.69	8.16 ± 0.47
	InfLoRA	83.28 ± 0.49	87.85 ± 0.49	3.32 ± 0.94	65.98 ± 0.71	69.39 ± 0.67	8.32 ± 0.69
	SD-LoRA	88.57 ± 0.64	91.81 ± 0.30	2.07 ± 0.13	73.15 ± 0.39	76.73 ± 0.25	7.86 ± 0.53
	CL-LoRA	88.39 ± 0.17	91.67 ± 0.27	1.97 ± 0.13	**75.14** ± 0.72	**79.90** ± 0.53	**7.37** ± 0.56
	PM-LoRA	**91.04** ± 0.31	**95.33** ± 0.31	**1.91** ± 0.25	**77.15** ± 0.49	**82.76** ± 0.55	**5.35** ± 0.68
DINO-1K [28]	HiDe-Prompt	**90.51** ± 0.11	**95.25** ± 0.03	**1.26** ± 0.20	67.36 ± 0.74	71.39 ± 0.46	8.45 ± 0.79
	InfLoRA	81.54 ± 0.56	85.33 ± 0.52	5.32 ± 0.42	63.58 ± 0.79	66.45 ± 0.66	9.32 ± 1.69
	SD-LoRA	85.29 ± 0.71	88.81 ± 0.47	6.10 ± 0.38	70.25 ± 0.51	73.73 ± 0.39	7.86 ± 1.07
	CL-LoRA	86.39 ± 0.32	91.60 ± 0.14	4.79 ± 0.137	**72.41** ± 0.69	**75.93** ± 0.27	**7.73** ± 0.88
	PM-LoRA	**90.43** ± 0.33	**94.65** ± 0.62	**1.65** ± 0.25	**75.77** ± 0.44	**80.76** ± 0.49	**6.03** ± 0.77

Table 7: Overall performance comparison on CIFAR-100 ($N = 10$) and ImageNet-R ($N = 40$). We present Final Average Accuracy (FA), Cumulative Average Accuracy (CA), and Average Forgetting Measure (FM) of all methods under different larger pre-trained models. The best and second-best results for each dataset and task are highlighted in **red** and **blue**, respectively.

B.4 Baselines and Reproducibility

All LoRA-based baselines (such as SD-LoRA and CL-LoRA) are re-implemented using the same ViT-B/16 backbone, LoRA placement (query and value only), training rank $r = 16$, optimizer, number of epochs, data preprocessing, and task splits as PM-LoRA. Non-LoRA baselines follow their original configurations wherever possible, with our data pipeline and task splits applied for fairness. The class order is generated with seed 42 and is shared by all methods. Reported values are the mean ± standard deviation over K independent runs with seeds $\{\dots\}$, which change parameter initialization and data shuffling.

C More Experiments and Ablation

Evaluation Metrics. We evaluate all methods using three standard metrics: final average accuracy (FA), cumulative average accuracy (CA), and average forgetting (FM). Let $S_{i,t}$ denote the test accuracy on the i-th task after training up to task t. The average accuracy after task t is

$$A_t = \frac{1}{t}\sum_{i=1}^{t} S_{i,t}. \tag{78}$$

After learning all T tasks, we define

$$\mathrm{FA} = A_T, \qquad \mathrm{CA} = \frac{1}{T}\sum_{t=1}^{T} A_t, \tag{79}$$

and the average forgetting measure

$$\mathrm{FM} = \frac{1}{T-1}\sum_{i=1}^{T-1}\left(\max_{t\in\{1,\dots,T-1\}} S_{i,t} - S_{i,T}\right). \tag{80}$$

FA and CA are our primary metrics, since they jointly capture both plasticity and retention over the task sequence, while FM serves as a complementary indicator that summarizes the amount of performance degradation on earlier tasks by the end of training. For Scalability Analysis, we compare the inference computations in terms of GFLOPs.

Performance on different pretrained models. Across both pre-trained backbones, PM-LoRA delivers consistently strong performance. Under ViT-L/14 [5], it matches or surpasses all LoRA-based baselines on CIFAR-100 and ImageNet-R in terms of both final and cumulative accuracy, while also achieving lower forgetting. Importantly, when switching to the DINO-1K [28] pre-training, PM-LoRA remains the best or near-best among LoRA methods on all three metrics and both datasets, indicating that its gains are not tied to a particular pretrained model but transfer robustly across different PTMs.

We examine whether PM-LoRA benefits from coupling its training objective to bounded consolidation, how this coupling affects plasticity over long task sequences, and where its accuracy and efficiency advantages hold. FA denotes final average accuracy, and FM denotes forgetting; accuracy and forgetting scores are reported as percentages.

Table 8: **Plasticity and long-horizon behavior.** Panel (a) compares compatibility constraints. Panel (b) reports differences in 100-task FA, defined as PM-LoRA minus each baseline. The panels correspond to separate experimental settings.

(a) Compatibility constraints

Method	Current-task acc. ↑	FA ↑
No compatibility regularizer	**83.5**	58.4
Hard orthogonality	78.4	65.7
Soft SCR	82.8	**69.1**

(b) ImageNet-R: 100 two-class tasks

Baseline	Persistent state	PM-LoRA FA gain
C-LoRA	Fixed adapter state	$+12.0$
InfLoRA	Fixed adapter + auxiliary statistics	$+7.2$
SD-LoRA	Task-growing state	$+2.3$

Table 9: **Base PM-LoRA and PM-LoRA-TIES.** ΔFA is TIES minus base PM-LoRA. Bold marks the better variant within each dataset.

Dataset	Base FA	TIES FA	ΔFA
CIFAR-100	**91.55**	91.33	-0.22
CUB200	**78.17**	78.01	-0.16
ImageNet-R	76.37	**76.49**	$+0.12$

Plasticity after rank saturation. We regularize against one rank-bounded compressed adapter rather than every historical task subspace, and SCR imposes a soft penalty rather than a hard null-space constraint. Table 8(a) shows that soft SCR reduces current-task accuracy by only 0.7 points relative to no compatibility regularizer, while improving final average accuracy by 10.7 points. Hard orthogonality produces a larger plasticity loss, reducing current-task accuracy from 83.5 to 78.4.

We further evaluate ImageNet-R over 100 two-class tasks. As shown in Table 8(b), PM-LoRA exceeds C-LoRA, InfLoRA, and SD-LoRA by 12.0, 7.2, and 2.3 points, respectively. The merged rank reaches 64 at task 20, after which the model learns 80 additional tasks with 87.2% mean current-task accuracy. Accuracy on the first 20 tasks decreases by 3.6 points over this interval, while retained-subspace turnover averages 14.1% per merge. Thus, saturation bounds adapter dimensionality while allowing the retained basis to evolve; we observe no accuracy cliff over the evaluated horizon.

Scope of the accuracy claim and the TIES variant. We restrict our claim to the compared methods and evaluated benchmarks: the PM-LoRA family achieves the best FA among the compared LoRA-based approaches on CIFAR-100, CUB200, and ImageNet-R. Base PM-LoRA is best overall on CUB200, and PM-LoRA-TIES is best overall on ImageNet-R. On CIFAR-100, HiDe-Prompt is 0.12 points higher than base PM-LoRA, with 91.67 versus 91.55 FA. We therefore do not claim superiority over every prompt- and adapter-based method on every dataset.

PM-LoRA-TIES applies trimming, sign election, and disjoint merging after deterministic magnitude scaling and before the same rank-bounded projection. SCR, the fixed rank budget, and disposal of the temporary adapter remain unchanged. Table 9 shows small, mixed differences from base PM-LoRA, ranging from -0.22 to $+0.12$ points. TIES is consequently an optional merging variant rather than the source of the principal gains; our theoretical analysis and principal ablations concern the base method.

Efficiency under matched training conditions. We compare the LoRA methods using the same backbone, query/value placement, training rank, optimizer, epochs, data pipeline, and task order. InfLoRA does not retain all historical LoRA branches; our efficiency comparison concerns measured constant factors, rather than an inference cost that grows with the number of tasks. GFLOPs in Table 10 count adapter-side computation, with the frozen-backbone cost shared across methods.

Table 10: **Efficiency and comparison with RanPAC.** GFLOPs denote adapter-side computation, adaptation parameters are those used at inference, and stored features are reported separately. Parameter and feature counts are in millions. Dashes denote unreported metrics. RanPAC retains class statistics; their size is not reported here.

Method	FA ↑	FM ↓	Adapter GFLOPs ↓	Adaptation params. (M)	Stored features (M)
InfLoRA	–	–	35.12	5.4	0.10
C-LoRA	–	–	20.57	1.7	0
RanPAC	75.60	4.80	–	–	–
PM-LoRA	**76.37**	**4.21**	**7.31**	**1.5**	**0**

PM-LoRA uses 4.8× fewer adapter-side GFLOPs than InfLoRA and 2.8× fewer than C-LoRA, while using fewer inference adaptation parameters and storing no features.

Comparison with frozen-feature adaptation. RanPAC [27] reaches 75.60 FA with 4.80 FM, compared with 76.37 FA and 4.21 FM for PM-LoRA (Table 10). We thus improve FA by 0.77 points and reduce forgetting by 0.59 points in this comparison. RanPAC operates on frozen features and retains class statistics, whereas PM-LoRA adapts the representation through one persistent 1.5M-parameter adapter. This result complements the LoRA comparisons without treating these different forms of persistent state as equivalent.

C.1 Method Comparison Summary

We compare PM-LoRA against representative methods from four major CL paradigms. The baseline **Fine-tune** approach directly updates all parameters without any forgetting prevention mechanism, serving as a lower bound for performance.

Regularization-based methods **EWC** [16] applies a Fisher-information-weighted quadratic penalty to constrain updates to parameters deemed critical by previous tasks. **SI** [46] continuously accumulates a parameter importance score based on the magnitude and sensitivity of past loss changes and penalizes deviations proportionally.

Prompt-based methods **L2P** [41] maintains a dynamically learned prompt pool and retrieves instance-level prompts via similarity to frozen backbone features. **DualPrompt** [40] introduces complementary "general" and "expert" prompt streams to decouple task-invariant and task-specific instructions. **S-Prompt** [38] learns domain-specific prompts and selects them using nearest domain centers in a frozen feature space. **CODA-Prompt** [30] decomposes learned prompts into orthogonal attention components and composes them dynamically for each input. **HiDe-Prompt** [36] decomposes prediction into within-task prediction, task-identity inference, and task-adaptive prediction.

Adapter-based methods **Model Surgery (MOS)** [32] is trained on different tasks using adapters and merges adapted and pre-trained feature spaces for stable downstream task performance. **SEMA** [35] monitors for layer-wise distribution shifts and selectively self-expands or reuses adapter modules via a learned routing mechanism. **EASE** [50] trains a distinct adapter per task and synthesizes features for old-class prototypes using semantic projection without storing exemplars.

LoRA-based methods **InfLoRA** [22] injects low-rank matrices constrained to lie in subspaces orthogonal to all previous gradients, eliminating interference. **SD-LoRA** [42] separates the learning of LoRA magnitude and direction with task-incremental branches whose directions are frozen post-training. **CL-LoRA** [11] maintains a shared adapter to capture cross-task information and frozen task-specific branches, using distillation and orthogonality constraints to preserve old representations..

Model Merging methods **TA** [15] modifies a pre-trained model by adding a task vector derived from the element-wise difference between the fine-tuned weights and the pre-trained initialization to inject specific task capabilities. **Fisher merging** [26] combines model parameters using a weighted average, where the influence of each weight is determined by its Fisher Information value to preserve parameters that are statistically critical for their respective tasks. **KnOTS** [31] utilizes Singular

Value Decomposition to align the task-specific updates from different models into a shared subspace to resolve interference and parameter misalignment before the merging process. **OPCM** [33] sequentially integrates new tasks by projecting their parameter updates onto a subspace orthogonal to the existing merged model to reduce interference during merging. **MagMax** [25] constructs a unified model by comparing the parameter changes across all task vectors and selecting the single weight with the maximum absolute magnitude for each position to retain the most significant features.

Sensitivity to rank budget and spectral threshold. We next study how the performance of PM-LoRA depends on the rank budget used during training and merging. Fig 5(A) reports the final average accuracy on Split ImageNet-R ($N = 40$ tasks) for training ranks $r_t \in \{8, 16, 32\}$ and merged rank budgets $R_{\max} \in \{32, 64, 128\}$. Across all nine configurations the FA remains in a narrow band between 72.42% and 76.37%, indicating that the method is not overly sensitive to the exact rank choice. The best trade-off is achieved at $r_t = 16$ and $R_{\max} = 64$ (76.37%), while both smaller and larger budgets lead to only modest changes. In particular, increasing $R_{\max}$ from 64 to 128 lowers FA at $r_t = 16$ and $r_t = 32$, but raises it from 73.88% to 73.95% at $r_t = 8$. The highest FA in this sweep is achieved at $r_t = 16$ and $R_{\max} = 64$.

Fig 5(B) analyzes the spectral energy threshold τ used when selecting the effective merged rank r_m. When τ is too small ($\tau = 0.85$), the truncation is overly aggressive and the FA drops to about 72%. As τ increases, accuracy improves and peaks around $\tau = 0.95$, where the model retains enough spectral energy to preserve past knowledge while still discarding low-energy, task-specific noise. For τ between 0.90 and 0.99 the FA stays above 75%, showing that PM-LoRA is robust to the exact value of τ and does not require fine-grained tuning; we therefore use $\tau = 0.95$ as the default in all other experiments.

$R_{\max}$ is selected from $\{16, 32, 48, 64, 128\}$ based on validation performance, and the effective merged rank increases only when new spectral energy is introduced while being hard-capped at $R_{\max}$, guaranteeing $O(1)$ parameter complexity. Unless otherwise stated, we use $R_{\max} = 64$ across all experiments. As shown in Fig. 6, the merged rank grows as tasks introduce new spectral energy until it reaches the cap, and it never exceeds $R_{\max}$; in the 100-task ImageNet-R setting it reaches 64 at task 20. Exploring a fully adaptive $R_{\max}$ that adjusts to task complexity is an interesting direction for future work.

Algorithm 1 PM-LoRA: Progressive Merging LoRA

Input: Base model θ, Sequence of tasks $\mathcal{T}_1, \ldots, \mathcal{T}_T$ with data $\{\mathcal{D}_1, \cdots, \mathcal{D}_T\}$.
Input: Rank budget $R_{\max}$, new task rank r_t, hyperparameters λ_A, λ_B.
Output: Final unified adapter $\Delta\mathbf{W}_{\text{final}}$ capable of all tasks $\mathcal{T}_{1\ldots T}$.
Initialize: Frozen adapter $\Delta\mathbf{W}_m = (\mathbf{B}_m, \mathbf{A}_m)$ is zero initialized.
for $t = 1$ **to** T **do** ▷ **Continual Learning Loop**
    // Stage 1: Task Adaptation
    Initialize the task-specific adapter $\Delta\mathbf{W}_t = (\mathbf{B}_t, \mathbf{A}_t)$ with rank r_t
    for each training step on $\mathcal{D}_t$ **do**
        # Compute scaling factor β
        $\alpha_m \leftarrow \|\mathbf{B}_m\mathbf{A}_m\|_F, \quad \alpha_t \leftarrow \|\mathbf{B}_t\mathbf{A}_t\|_F$
        $\beta \leftarrow 1$ **if** $t = 1$ **else** $\alpha_m/(\alpha_t + \epsilon)$
        # Train on scaled sum + Regularizer
        $\Delta\mathbf{W}_{\text{sum}} \leftarrow \mathbf{B}_m\mathbf{A}_m + \beta \cdot (\mathbf{B}_t\mathbf{A}_t)$
        $\ell_t \leftarrow \ell(f_{\mathbf{W}_0+\Delta\mathbf{W}_{\text{sum}}}(\mathbf{x}), \mathbf{y}) + \mathcal{R}_{\text{spec}}(\mathbf{A}_t, \mathbf{A}_m, \mathbf{B}_t, \mathbf{B}_m)$
        Back-propagate and update $\mathbf{B}_t, \mathbf{A}_t$ ▷ $\mathbf{B}_m, \mathbf{A}_m$ are frozen
    // Stage 2: Knowledge Merging
    if $t == 1$ **then**
        $(\mathbf{B}_m, \mathbf{A}_m) \leftarrow \text{Project}(\mathbf{B}_t, \mathbf{A}_t, R_{\max})$ ▷ First task just sets the adapter
    else
        # Merge new knowledge into frozen adapter
        $\alpha_m \leftarrow \|\mathbf{B}_m\mathbf{A}_m\|_F, \quad \alpha_t \leftarrow \|\mathbf{B}_t\mathbf{A}_t\|_F$
        $\beta \leftarrow \alpha_m/\alpha_t$
        # Create scaled factors for the sum $\Delta\mathbf{W}_m + \beta\Delta\mathbf{W}_t$
        $\mathbf{B}_{\text{cat}} \leftarrow [\mathbf{B}_m \mid \beta \cdot \mathbf{B}_t]$; $\mathbf{A}_{\text{cat}} \leftarrow [\mathbf{A}_m^\top \mid \mathbf{A}_t^\top]^\top$
        # Compress back to fixed budget $R_{\max}$ (solving domination)
        $(\mathbf{B}'_m, \mathbf{A}'_m) \leftarrow \text{TractableSVDProject}(\mathbf{B}_{\text{cat}}, \mathbf{A}_{\text{cat}}, R_{\max})$
return $\Delta\mathbf{W}_{\text{final}} = \mathbf{B}_m\mathbf{A}_m$ ▷ Single adapter for all tasks

D Limitations

PM-LoRA operates under a fixed adapter rank budget. This bounds the stored adapter state, but it does not guarantee that all task-specific information can be retained as the task sequence grows. SCR is a soft penalty rather than a hard orthogonality constraint. It reduces interference but does not eliminate the trade-off between preserving earlier knowledge and fitting new tasks, particularly when tasks require substantially different update directions. Our long-horizon experiments examine this trade-off over a finite sequence and do not establish retention over arbitrarily long streams.

PM-LoRA also relies on a frozen pretrained backbone. Its effectiveness therefore depends on whether the pretrained representations support the incoming tasks; adaptation may be limited when substantial changes to those representations are required. Finally, the fixed-capacity claim applies to the LoRA adapter state. The class-incremental classifier still expands as new classes are introduced.

E Broader Impacts

PM-LoRA reduces the memory and inference cost of adapting a pretrained model over a long sequence of tasks: it keeps one rank-bounded adapter instead of one adapter per task, which can lower storage and energy costs for long-lived or on-device systems. The method is a general-purpose training procedure for image classifiers and does not by itself create new capabilities for misuse. Continual adaptation nonetheless carries risks. Updates learned from new data streams can carry biases or labeling errors in those data into the shared adapter, and although PM-LoRA reduces forgetting, it does not eliminate it, so performance on earlier classes can degrade in ways that are not visible without re-evaluation. Our experiments use standard public vision benchmarks; before deployment in applications where errors affect people, a continually updated model should be re-evaluated on all previously supported classes, including for fairness and robustness.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 claim that PM-LoRA keeps a single LoRA adapter with a fixed rank budget throughout continual learning (Sections 3.1 and 3.2; Algorithm 1) and achieves strong class-incremental performance on CIFAR-100, CUB-200, ImageNet-R, and VTAB (Tables 1 and 2). Appendix C limits the accuracy claim to the compared methods and benchmarks (HiDe-Prompt is 0.12 points higher on CIFAR-100), and Appendix D notes that the fixed-capacity claim covers the adapter state, not the growing classifier.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix D discusses that the fixed rank budget bounds the stored state but cannot guarantee that all task information is retained, that SCR is a soft penalty that leaves a stability–plasticity trade-off when tasks need very different update directions, that the long-horizon experiments cover finite task sequences, that performance depends on a frozen pretrained backbone, and that the classifier still grows with new classes. Appendix C restricts the accuracy claim to the compared methods and benchmarks.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Section 3.1 states the interference bound, and Appendix A gives the assumptions and complete proofs for the interference bound, Lemma 1, Corollary 1, Propositions 3 and 4, and the single-merge forgetting bound (Theorem 1). The discussion of rank compression as spectral denoising at the end of Appendix A is a heuristic argument under stated assumptions, not a theorem.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sections 3.1 and 3.2 and Algorithm 1 specify the method. The experimental setup and Appendix B give the backbone, LoRA placement, rank, and initialization, the optimizer, learning rate, weight decay, gradient clipping, epochs, batch sizes, augmentation, task splits and class order, the SCR weights, the rank budget, the energy threshold, and how the scaling factor β is computed at initialization; Appendix C defines the evaluation metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is available at `https://github.com/VisualReasoner/pmlora`. All benchmarks and pretrained backbones are publicly available, and the data pipeline, task splits, and class order are specified in Appendix B.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setup and Appendix B give the data splits, class order, optimizer, learning rate, weight decay, gradient clipping, epochs, batch sizes, augmentation, LoRA placement and rank, and the PM-LoRA hyperparameters, together with how (λ_A, λ_B), $R_{\max}$, and τ were chosen (Appendix B and C).

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Tables 1, 2, and 7 report the mean $\pm$ one standard deviation over independent runs; Appendix B states the number of runs and the sources of variability between them.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The paper reports the software and hardware (PyTorch 2.1 on NVIDIA A100 GPUs with 40GB memory) and the adapter-side inference GFLOPs and parameter counts (Tables 5 and 10), but not the training time per run, the total compute, or the compute used for preliminary experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: We have reviewed the NeurIPS Code of Ethics. The work evaluates a training algorithm on standard public vision benchmarks with public pretrained backbones and involves no human subjects or private data.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: Appendix E discusses positive impacts (lower memory and inference cost for adapting pretrained models over long task streams) and negative impacts (updates learned from new data can carry biases or label errors into the shared adapter, and forgetting of earlier classes is reduced but not eliminated), and recommends re-evaluating a continually updated model on all supported classes before deployment.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The released code trains image classifiers on public benchmarks; it is not a generative or otherwise high-risk model, and no scraped data are released.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: The datasets, pretrained backbones, and baseline methods are credited by citation, but the paper does not state the license and terms of use of each asset.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released code (`https://github.com/VisualReasoner/pmlora`) is documented with installation and usage instructions, the configurations of Appendix B, and a license. No new data are collected.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper involves no crowdsourcing and no research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The work uses only existing public benchmarks and involves no human participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not part of the method, the experiments, or the theoretical analysis; any use for writing or editing assistance is outside the core methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2026/MainTrackHandbook`) for what should or should not be described.