ALLOCADAPT: ADAPTIVE TRAINING BUDGET ALLOCATION FOR CONTINUAL LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent class-incremental learning (CIL) methods achieve strong performance by adapting pre-trained models with lightweight adapters and low-rank updates. However, their parameter efficiency does not necessarily translate into efficient training, as fixed schedules overlook differences in optimization demand across tasks and stages. We propose AllocAdapt, an adaptive update allocation framework that jointly addresses two complementary decisions: *how much to update* and *where to allocate updates*. HowMuch combines probe statistics with historical calibration to determine task- and stage-level training horizons under budget constraints. Within these horizons, Where allocates update opportunities through parameter-update routing or class sampling, according to each learner's optimization structure. Experiments on three recent parameter-efficient CIL methods and three image classification benchmarks show that AllocAdapt reduces training iterations by up to 80% while largely preserving average incremental accuracy relative to the original training procedures.

1 INTRODUCTION

Class-incremental learning (CIL) requires a model to recognize an expanding set of classes while retaining earlier knowledge (Rebuffi et al., 2017; Douillard et al., 2020). Pre-trained models (PTMs) provide a strong starting point, supporting lightweight adaptation through prompts (Wang et al., 2022a;b) and adapters or low-rank modules (Zhou et al., 2024; He et al., 2025). Adapter merging in MOS (Sun et al., 2025) and the integration of task-specific and universal adapters in TUNA (Wang et al., 2025) further illustrate the diversity of these pipelines. Their parameter efficiency, however, leaves open how much training is needed as new tasks arrive.

Parameter efficiency does not itself ensure optimization efficiency. A fixed task or stage schedule assigns training duration in advance, although loss trajectories, remaining improvement, and instability can vary across the task sequence. Uniformly shortening these schedules may remove useful updates from demanding tasks while leaving redundant training elsewhere. Prior work has established the importance of computational budgets and adaptive computation in continual learning (Prabhu et al., 2023; Zhang et al., 2024; Seo et al., 2025). We focus on allocating optimization updates within modular, parameter-efficient CIL pipelines.

This setting raises two coupled decisions: *How Much should we update?* and *Where should we allocate updates?* The former determines the training horizon of each task or stage; the latter distributes update opportunities across parameter branches, class samples, or classifier heads. These choices depend on the learner's optimization structure. Reducing training duration alone leaves the distribution of the remaining updates unchanged.

We propose **AllocAdapt** to address both decisions. *HowMuch* combines current probe statistics with historical calibration to estimate demand, then selects a feasible training horizon under budget constraints, including the updates already used for probing. *Where* uses the learner's available interfaces: loss-gradient routing in CL-LoRA, class-sample allocation during classifier alignment in MOS, and classifier-head update allocation during classifier alignment in TUNA. These instantiations cover different optimization structures while retaining the underlying backbones and adaptation modules.

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

We evaluate the three learners on CIFAR-100, ImageNet-R, and ImageNet-A. AllocAdapt reduces training iterations by 15.10%–80.04% relative to each learner's original training procedure. In seven of the nine comparisons, it removes more than half of the training iterations while keeping average incremental accuracy within 0.5 percentage points of the original learner. Ablations on CL-LoRA and MOS examine the interaction between horizon selection and update allocation at comparable iteration counts.

Our contributions are:

- We formulate adaptive update allocation in parameter-efficient CIL through coupled decisions on training duration and update distribution, accounting for differences across tasks, stages, and optimization structures.
- We develop **AllocAdapt**, which calibrates current probe statistics against task history to select feasible task- and stage-level training horizons under budget constraints. Within these horizons, it adapts loss-gradient routing, class sampling, or classifier-head selection to the learner's optimization structure, enabling update allocation across low-rank and adapter-based learners.
- Experiments across three learners and three benchmarks demonstrate up to 80% fewer training iterations with competitive average incremental accuracy. Component ablations examine the roles of both allocation decisions at comparable iteration counts.

2 Related Work

Parameter-efficient adaptation. Parameter-efficient adaptation reduces the number of trainable parameters needed to adapt a pre-trained model. Adapters insert lightweight trainable modules into a largely frozen backbone (Houlsby et al., 2019), while LoRA represents weight updates through low-rank matrices (Hu et al., 2022). AdaLoRA makes this capacity allocation adaptive by distributing a parameter budget across weight matrices according to importance estimates (Zhang et al., 2023b). These approaches control the size and distribution of adaptation capacity. Training effort additionally depends on how long optimization continues and how updates are distributed during training. Parameter efficiency therefore leaves room for a complementary form of allocation: adapting the training process while retaining the learner's parameterization.

Pre-trained models for class-incremental learning. Class-incremental learning (CIL) requires acquiring new classes while preserving earlier knowledge (Rebuffi et al., 2017; Douillard et al., 2020). Pre-trained models provide transferable representations, shifting attention toward how these representations should be adapted across tasks. Prompt-based methods address this problem through lightweight conditioning: L2P learns a prompt pool (Wang et al., 2022b), DualPrompt separates task-invariant and task-specific prompts (Wang et al., 2022a), and CODA-Prompt learns input-conditioned combinations of prompt components (Smith et al., 2023). Beyond representation adaptation, SLCA highlights the role of classifier alignment, combining slow representation learning with a subsequent alignment stage based on class-distribution modeling (Zhang et al., 2023a). Modular adaptation provides another way to preserve and share knowledge: EASE ensembles expanding task-specific subspaces (Zhou et al., 2024); MOS combines adapter merging with self-refined retrieval (Sun et al., 2025); and TUNA integrates task-specific and universal adapters (Wang et al., 2025). Within low-rank adaptation, CL-LoRA combines shared and task-specific modules with knowledge distillation and gradient reassignment (He et al., 2025). Together, these methods establish complementary structures for continual adaptation, including shared and task-specific parameters and separate representation-learning and classifier-alignment stages. These structures also provide concrete interfaces for controlling training duration and allocating updates within an existing learner.

Computationally efficient continual learning. Computationally budgeted evaluation shows that resource constraints can change the relative performance of continual learning methods (Prabhu et al., 2023). Several approaches therefore incorporate computation into the learning process. DietCL allocates a constrained budget between supervised and self-supervised learning in sparsely labeled streams (Zhang et al., 2024). aL-SAR combines adaptive layer freezing with frequency-based replay sampling under FLOP and total-memory budgets (Seo et al., 2025). C-Flat Turbo

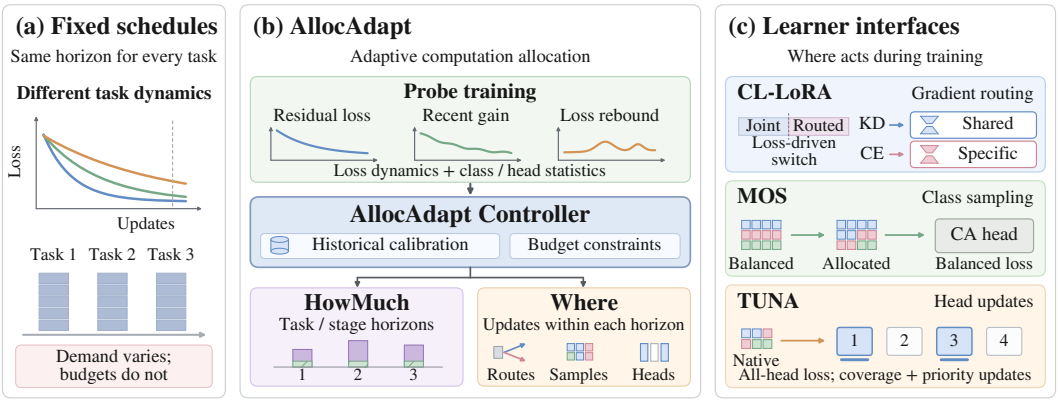

Figure 1: Overview of AllocAdapt. (a) Fixed schedules overlook differences in task optimization dynamics. (b) Probe statistics and historical calibration inform the controller: HowMuch selects task- and stage-level horizons under budget constraints, while Where allocates updates within these horizons. Hatched green segments denote probe updates included in each horizon. (c) CL-LoRA switches from joint training to CE-to-specific and KD-to-shared gradient routing; MOS redistributes a fixed sample count during classifier alignment while retaining a balanced objective; TUNA allocates classifier-head updates through coverage and priority. Highlighted TUNA heads receive updates; all heads participate in the forward computation and loss. Curves, horizons, sample counts, and head selections are schematic.

reduces repeated sharpness and flatness gradient computations through gradient reuse and adaptive scheduling (Li et al., 2026). These approaches address complementary allocation decisions, including learning objectives, trainable layers, replay samples, and optimizer computations. AllocAdapt studies how to connect task- and stage-level training duration with update allocation in modular, pre-trained CIL learners. Its HowMuch component uses current probes and historical calibration to select training horizons under budget constraints, while Where allocates updates within those horizons through the learner's available training interfaces. The resulting controls include loss-gradient routing, class sampling during classifier alignment, and classifier-head update selection. This formulation ties training length to allocation within training while retaining the base learner's adaptation architecture.

3 Method

3.1 Problem Setup and Overview

We consider class-incremental learning over tasks $\{\mathcal{D}_t\}_{t=1}^{T}$ with disjoint class sets $\mathcal{C}_t$. After task t, the learner predicts over all seen classes $\mathcal{C}_{1:t} = \bigcup_{j \le t} \mathcal{C}_j$ without task identity at inference. We focus on learners that adapt a frozen pre-trained backbone through lightweight modules and classifiers.

AllocAdapt separates two decisions: *HowMuch* determines the training horizon of each task stage, and *Where* allocates training operations within that horizon. A stage s exposes an allocation interface: loss-gradient paths in CL-LoRA, class-sampling quotas in MOS, or classifier-head updates in TUNA. The controller first observes optimization during a training probe, calibrates demand using earlier-task statistics, and selects a feasible horizon $B_{t,s}$. The learner-specific policy then allocates the remaining training opportunities. Probe updates count toward $B_{t,s}$, and the learner retains its architecture, stage order, and inference procedure.

Figure 1 summarizes the allocation procedure and its learner-specific training interfaces.

3.2 HowMuch: Demand-Guided Training Horizons

Estimating stage demand. A fixed schedule allocates the same training length regardless of optimization progress. For the Adapter and classifier-alignment (CA) stages of MOS and TUNA, let

$\ell_1, \ldots, \ell_P$ be probe epoch losses and $g_i = (\ell_{i-1} - \ell_i)/(|\ell_{i-1}| + \epsilon)$. We collect

$$R = \frac{|\ell_P|}{|\ell_1| + \epsilon}, \qquad G = \mathrm{median}_{i \in \mathcal{I}}[g_i]_+, \qquad I = \mathrm{median}_{i \in \mathcal{I}}[-g_i]_+, \tag{1}$$

where $\mathcal{I}$ contains the final probe observations and $[x]_+ = \max(x, 0)$. These statistics capture residual loss, continued improvement, and recent loss increases. They provide complementary indicators of remaining optimization demand: a small endpoint loss alone cannot distinguish a converging stage from one that is still improving or fluctuating.

Historical calibration and horizon selection. We normalize each statistic against its earlier-task history within the same stage:

$$\begin{aligned} \mathcal{N}_s(x) &= \mathrm{clip}_{[-z_{\max}, z_{\max}]}\left(\frac{x - \mu_{s,x}}{\max(\sigma_{s,x}, \sigma_{\min})}\right), \\ d_{t,s} &= w_R \mathcal{N}_s(R) + w_G \mathcal{N}_s(G) + w_I \mathcal{N}_s(I). \end{aligned} \tag{2}$$

The nonnegative weights sum to one. Separate histories accommodate different Adapter and CA loss scales; insufficient history gives zero normalized demand. Given a reference horizon E_s^0 and admissible bounds, the desired epoch count is

$$E_{t,s}^{\mathrm{des}} = \mathrm{clip}_{[E_s^{\min}, E_s^{\max}]}\left(E_s^0 + \mathbf{1}\{|d_{t,s}| \geq \delta_s\}\,\mathrm{round}(\beta_s d_{t,s})\right). \tag{3}$$

The reference and bounds specify the operating budget; observed demand determines task-specific deviations. The deadband δ_s suppresses small adjustments. Enabled stage-specific budget balances allow later stages to spend updates saved by earlier tasks. The executable horizon respects the original schedule and includes the completed probe.

CL-LoRA uses the same probe-and-calibrate principle with classification (CE) and distillation (KD) loss dynamics. It combines recent CE/KD improvement and KD-loss rebound into a task-demand score, then maps its historical normalization to a bounded retained-budget ratio. The exact mapping and allowance rules are given in Appendix A.

3.3 WHERE: ALLOCATION THROUGH LEARNER-SPECIFIC INTERFACES

The useful allocation unit depends on the optimization structure. We choose operations already exposed by each learner so that Where can redistribute training opportunities without introducing a new inference module. This yields three complementary instantiations.

CL-LoRA: loss-based branch routing. CL-LoRA (He et al., 2025) exposes shared and task-specific low-rank branches. Early joint training allows both branches to adapt; later, directing classification toward the task-specific branch and distillation toward the shared branch separates their update roles. We determine this transition from the evolving losses. Let $\bar{g}_{t,k}^{\mathrm{CE}}$ and $\bar{g}_{t,k}^{\mathrm{KD}}$ be recent median relative improvements of smoothed window losses, and let $\bar{r}_{t,k}^{\mathrm{KD}}$ measure rebound above the best preceding smoothed KD loss. The routing score is

$$u_{t,k} = a_{\mathrm{CE}}[\bar{g}_{t,k}^{\mathrm{CE}}]_+ + a_{\mathrm{KD}}[\bar{g}_{t,k}^{\mathrm{KD}}]_+ - a_{\mathrm{reb}}\bar{r}_{t,k}^{\mathrm{KD}}. \tag{4}$$

Continued improvement favors extending joint training, while rebound favors earlier separation. A robust normalization against earlier-task scores adjusts a switch target within a budget-conditioned decision interval. After entering this interval, the controller re-evaluates the target at each check and switches when current progress reaches it. The route is then retained: CE gradients are masked on the shared branch and KD gradients on the task-specific branch. This controls loss-gradient contributions while retaining the native optimizer dynamics. Appendix A.2 specifies the transition rule.

MOS: class-sample allocation. MOS (Sun et al., 2025) performs CA using synthetic features drawn from estimated class-conditional distributions. This exposes a direct allocation unit: the number of features sampled from each class. Uniform counts need not reflect differences in class loss or gradient variability. Its linear CA head makes class statistics accessible from losses, logits,

and synthetic features. We therefore redistribute a fixed sample budget across classes, using these statistics to guide sampling while retaining joint optimization of the classifier.

Let P_c be the synthetic-feature distribution for class c, and $q_c = 1/C$ for C seen classes. The target objective is

$$\mathcal{L}_{\mathrm{bal}}(\phi) = \sum_{c=1}^{C} q_c \, \mathbb{E}_{\tilde{x}\sim P_c}\big[\ell(h_\phi(\tilde{x}), c)\big]. \tag{5}$$

Where changes the class proposal or integer quotas while preserving the total number of synthetic features per epoch. Importance correction or stratified class means account for the changed allocation, maintaining the balanced target at a fixed classifier. This allows sampling effort to follow class-level demand without assigning larger target loss weights to frequently sampled classes. Appendix A.3 details the probe-fixed sampling formulation and its importance correction, while Appendix B.2 documents the configurations used in the component comparisons.

TUNA: classifier-head update allocation. TUNA (Wang et al., 2025) combines task-specific and universal adapters. In our implementation, its expandable classifier additionally exposes parameter blocks corresponding to the classes introduced by successive tasks. These blocks provide a natural unit for allocating updates: their optimization demands can differ even under balanced sampling. We use this interface to adjust update frequency while preserving native synthetic sampling and cross-entropy over all seen classes. Routing acts on classifier parameters during CA; adapter selection remains part of the base learner's inference procedure.

For classifier block θ_h with d_h parameters, demand is estimated from ordinary CA backward passes:

$$s_h = \mathrm{EMA}\left[\mathrm{Avg}_{b\in\mathcal{B}_h} \frac{\|\nabla_{\theta_h}\mathcal{L}_b\|_2^2}{d_h}\right]. \tag{6}$$

Here $\mathcal{B}_h$ contains eligible probe or coverage observations. Differentiating through the actual classifier accounts for its normalization and scaling. After the full-head probe, each routed step updates K of the H blocks. Rotating coverage steps alternate with priority steps, whose soft target inclusion rates satisfy

$$\pi_h = \min(1, \lambda\tilde{s}_h), \qquad \sum_{h=1}^{H}\pi_h = K, \tag{7}$$

where $\tilde{s}_h$ is a numerically stabilized score. A deterministic quota scheduler converts these rates into selections. Coverage maintains update opportunities across heads; soft quotas distribute priority opportunities smoothly across their relative demands. Scores are refreshed from coverage observations for subsequent epochs. Unselected blocks remain in the full-class forward computation, while their parameters and optimizer state are unchanged for that step. Thus, MOS allocates samples across classes, whereas TUNA allocates updates across classifier blocks.

3.4 Execution and Integration

Algorithm 1 summarizes the allocation procedure. HowMuch follows the learner-specific mappings in Section 3.2, while Where follows the decision and refresh schedules in Section 3.3.

For each executed stage, AllocAdapt trains the probe, determines the horizon, and applies Where at the prescribed decision points until that horizon is reached. A stage ending at its probe has no subsequent routed updates. Demand histories and budget balances carry information forward to later tasks; the underlying learner performs feature-statistics estimation, adapter operations, and prediction in its prescribed order.

HowMuch controls training length, while Where is designed to improve the use of the available updates. We count all executed training iterations, including the probe, using the same convention for each learner and its original counterpart. Sampling allocation preserves the per-epoch sample budget, and head routing changes parameter-update exposure within CA steps. The accounting conventions and implementation details are provided in Appendix A.

Algorithm 1 AllocAdapt: Adaptive Update Allocation

Input: Tasks $\{\mathcal{D}_t\}_{t=1}^{T}$, learner $\mathcal{M}$, allocation settings
Output: Updated learner $\mathcal{M}$

Initialize demand histories $\mathcal{H}$, budget state $\mathcal{K}$, and Where state
for $t = 1, \ldots, T$ **do**
  Prepare task t using the native procedure
  for each executed stage s in native order **do**
    if stage s is exempt from allocation **then**
      Execute its original training schedule
    else
      Train the probe; retain statistics and executed count u
      Calibrate demand against earlier-task stage history $\mathcal{H}_{<t,s}$
      Map calibrated demand to $B_{t,s}^{\mathrm{des}}$ as in Section 3.2
      $A_{t,s} \leftarrow$ available allowance under the learner's budget rule
      $B_{t,s} \leftarrow \max\{u, \min(B_{t,s}^{\mathrm{des}}, B_{t,s}^{\mathrm{orig}}, A_{t,s})\}$
      Append the probe demand summary to $\mathcal{H}$
      while $u < B_{t,s}$ **do** ▷ apply the matching update
        CL-LoRA: budget-conditioned switching; joint → CE-to-specific/KD-to-shared updates
        MOS CA: allocate a fixed sample count; update with balanced-objective weights q_c/r_c
        TUNA CA: alternate coverage/priority heads; use full-class loss and update selected heads only
        Otherwise: execute one native iteration
        Maintain Where state at its prescribed decision/refresh boundaries
        $u \leftarrow u + 1$
      end while
      Settle $\mathcal{K}$ using actual usage u
    end if
    Complete the native operations associated with stage s
  end for
end for

4 EXPERIMENTS

We evaluate accuracy and iteration savings, component contributions, and realized allocation behavior.

4.1 EXPERIMENTAL SETUP

Benchmarks and comparisons. We evaluate CL-LoRA (He et al., 2025), MOS (Sun et al., 2025), and TUNA (Wang et al., 2025) on CIFAR-100 (Krizhevsky, 2009), ImageNet-R (Hendrycks et al., 2021a), and ImageNet-A (Hendrycks et al., 2021b), using pre-trained ViT-B/16 and 224×224 CIFAR-100 inputs. The learners cover complementary allocation interfaces. Task partitions may differ across learners; each Original/AllocAdapt pair shares its partition, class order, backbone, and base architecture. Original denotes our implementation's full reference schedule. Protocols and reference configurations are given in Appendix B.1.

Metrics. Let a_t denote all-seen-class accuracy after task t, evaluated without task identities. We report Avg $\bar{A} = T^{-1}\sum_{t=1}^{T} a_t$, Final a_T, and Save (%), defined as $S = 100(1 - N^{\mathrm{AllocAdapt}}/N^{\mathrm{Original}})$. Here N sums native training iterations across tasks and stages, including initial-task and probe training. Stage counts are unweighted; S measures iteration savings rather than wall-clock or FLOP reduction. For both the main results and ablation studies, we report the mean of each metric over independent runs with different random seeds. Iteration savings are computed relative to the matched Original run before averaging.

4.2 MAIN RESULTS

Substantial savings with closely preserved accuracy. Table 1 shows iteration savings of 15.10–80.04%. MOS saves 50.41–76.43% with average-accuracy differences of at most 0.263 percentage points, and TUNA saves 58.80–80.04% with differences of at most 0.455 points. For example, MOS

Table 1: Accuracy (%) and training-iteration savings. Save (%) is relative to the corresponding Original learner.

	CIFAR-100			ImageNet-R			ImageNet-A		
Method	Avg ↑	Final ↑	Save (%) ↑	Avg ↑	Final ↑	Save (%) ↑	Avg ↑	Final ↑	Save (%) ↑
CL-LoRA	90.400	86.03	–	81.444	73.57	–	70.021	58.85	–
+AllocAdapt	90.160	84.56	57.00	81.821	74.63	20.36	69.310	58.39	15.10
MOS	93.115	89.12	–	84.295	78.90	–	67.180	56.88	–
+AllocAdapt	92.852	88.88	63.96	84.129	78.63	50.41	67.109	57.21	76.43
TUNA	94.622	91.32	–	85.473	79.88	–	71.190	61.22	–
+AllocAdapt	94.488	91.22	58.80	85.818	80.67	67.35	70.735	62.01	80.04

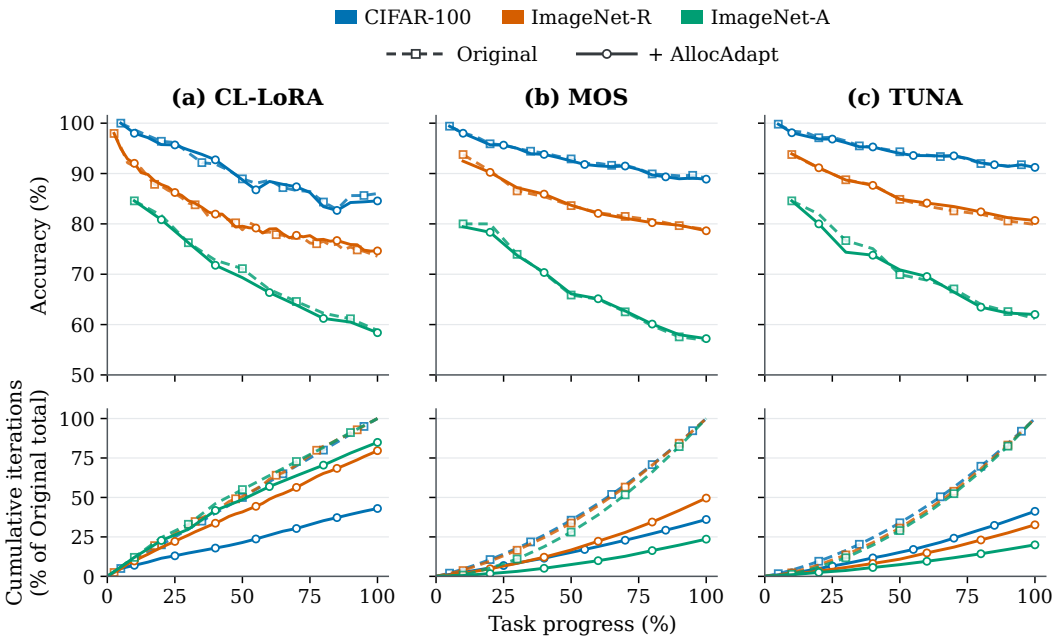

Figure 2: Task-wise accuracy (top) and cumulative iterations relative to each Original's complete training count (bottom). Colors denote datasets; dashed/solid lines denote Original/AllocAdapt. Task progress is $100t/T$; curves connect observed task evaluations.

retains 92.852% average accuracy on CIFAR-100 against 93.115% for Original while eliminating nearly two thirds of the iterations. Thus, large reductions can preserve competitive average accuracy across different adaptation structures.

Accuracy–iteration trade-offs. On ImageNet-R, CL-LoRA improves final accuracy by 1.06 percentage points while saving 20.36% of iterations; TUNA improves it by 0.79 points while saving 67.35%. On ImageNet-A, TUNA achieves the largest iteration reduction, 80.04%, with a 0.79-point gain in final accuracy and a 0.455-point decrease in average accuracy. These paired results show that substantial iteration reductions can preserve or improve accuracy, with the balance varying across learners and datasets.

Savings throughout incremental learning. In Figure 2, MOS's paired accuracy curves remain close as iteration savings accumulate. TUNA behaves similarly on CIFAR-100 and improves later ImageNet-R evaluations despite fewer iterations. The lower curves reach $100-S$ for AllocAdapt and 100 for Original, linking the process directly to Table 1. Substantial cumulative savings therefore need not produce a progressively widening accuracy gap.

Table 2: Component ablations at comparable iteration budgets. H/W denote HowMuch/Where. Accuracy and savings are in percent; bold marks the highest accuracy within each learner–dataset group.

			CIFAR-100			ImageNet-R		
Variant	H	W	Avg ↑	Final ↑	Save (%) ↑	Avg ↑	Final ↑	Save (%) ↑
CL-LoRA								
Static	–	–	89.683	83.42	57.00	81.559	73.38	19.51
HowMuch-only	✓	–	89.736	83.56	57.26	81.735	74.48	19.83
Where-only	–	✓	89.679	83.47	57.00	81.556	73.45	19.51
Full	✓	✓	**90.160**	**84.56**	57.00	**81.821**	**74.63**	20.36
MOS								
Static	–	–	92.742	88.65	63.86	83.776	78.54	50.00
HowMuch-only	✓	–	92.845	88.73	63.96	83.857	78.53	50.45
Where-only	–	✓	92.826	88.66	63.86	84.110	78.60	50.00
Full	✓	✓	**92.852**	**88.88**	63.96	**84.129**	**78.63**	50.41

4.3 Ablation Studies

We evaluate CL-LoRA and MOS on CIFAR-100 and ImageNet-R. Static uses fixed shortened horizons and fixed/native allocation; HowMuch-only adapts horizons; Where-only adapts allocation within fixed horizons; Full enables both. Budgets are comparable within each group. The corresponding training configurations are documented in Appendix B.2.

Adaptive allocation improves on shortened training. Full achieves the highest mean Avg and Final accuracy in all four learner–dataset comparisons in Table 2. Relative to Static, CL-LoRA improves Avg/Final accuracy by 0.48/1.14 percentage points on CIFAR-100 and 0.26/1.25 points on ImageNet-R. MOS improves the corresponding metrics by 0.11/0.23 and 0.35/0.09 points. At comparable iteration budgets, these results support adaptive allocation over fixed shortened schedules.

Component contributions. Full achieves higher final accuracy than either single-component variant in all four comparisons. For CL-LoRA, adding Where to HowMuch-only improves final accuracy by 1.00 and 0.15 percentage points on CIFAR-100 and ImageNet-R, respectively. Adding HowMuch to Where-only yields gains of 1.09 and 1.18 points. For MOS, enabling Where on top of HowMuch improves final accuracy by 0.15 and 0.10 points, respectively, with an additional 0.27-point gain in average accuracy on ImageNet-R. These comparisons support combining horizon selection with update allocation, although the gain magnitudes vary across learners and datasets.

4.4 Allocation Analysis

HowMuch varies the executed horizon. Figure 3 shows the executed HowMuch horizons on ImageNet-R. CL-LoRA executes 70.5–89.7% of each incremental task's original iterations; MOS and TUNA execute 14–16 and 7–9 CA epochs, respectively. These task-dependent horizons reflect calibrated demand subject to budget constraints. Appendix B.3 presents the complete score–horizon trajectories, including historical calibration and the configured reference budgets.

Where adapts distinct update interfaces. Figure 4 shows task-dependent routing boundaries in CL-LoRA, class allocation in MOS, and head updates in TUNA. MOS's sample ratios span 0.69–1.84 while retaining the sample total within each task. TUNA's head update frequencies span 30.2–80.0%, combining coverage with differentiated priorities. These patterns visualize learner-specific allocation within the assigned horizons.

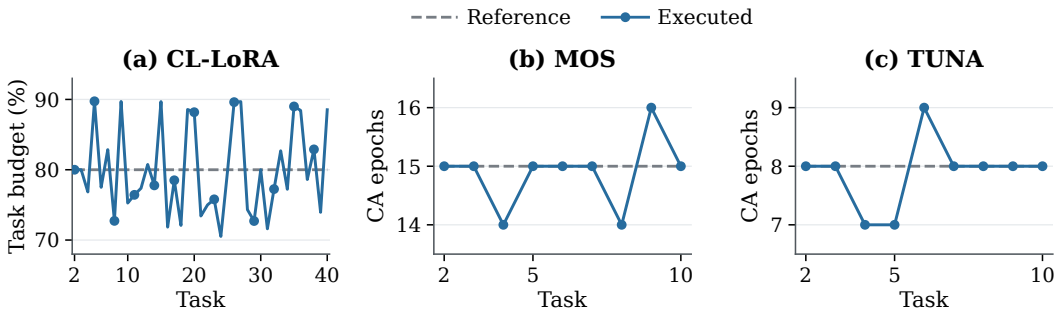

Figure 3: HowMuch on ImageNet-R. Configured reference and executed horizons, including probes. CL-LoRA uses a percentage of each task's original iterations; MOS/TUNA use CA epochs. The initial task is omitted.

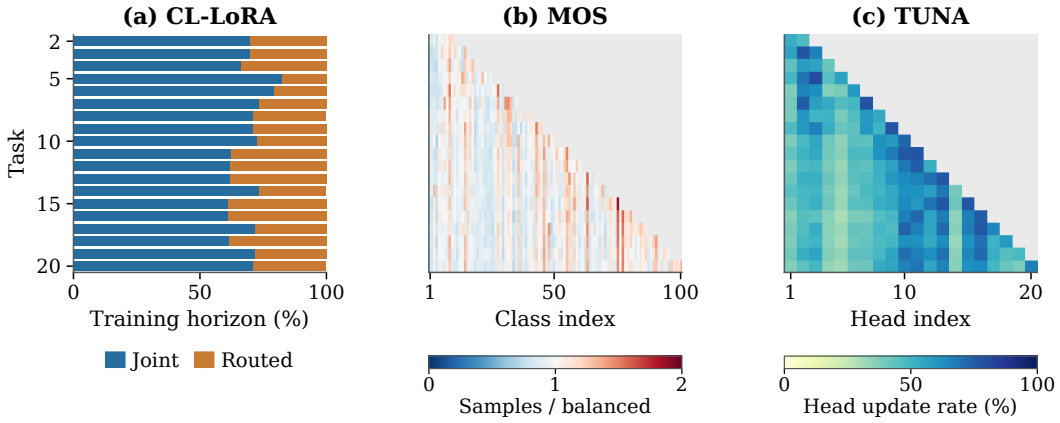

Figure 4: Where on CIFAR-100. (a) Joint/routed fractions. (b) Class samples relative to balanced allocation. (c) Head update frequencies. Rows denote tasks; gray denotes unseen classes/heads. Initial-task and MOS/TUNA probe-only training are excluded. Heatmap scales differ.

5 Conclusion

We presented AllocAdapt, a framework for adaptive training allocation in parameter-efficient class-incremental learning. By combining current optimization signals with historical calibration, its HowMuch component adjusts task- and stage-level training horizons, while Where allocates updates through the interfaces available in each learner. This separation supports loss-gradient routing, class sampling, and classifier-head updates within a common framework. Experiments with three base learners on three benchmarks show that AllocAdapt reduces training iterations by 15.10–80.04% while largely preserving average accuracy. At comparable iteration budgets, component comparisons on CL-LoRA and MOS further support the benefit of adaptive allocation over fixed shortened schedules. These results highlight training allocation as a practical complement to parameter-efficient adaptation. Future work will extend the framework to additional learning architectures and incorporate differences in update cost to connect iteration savings more directly to computational efficiency.

AI USE STATEMENT

In this work, we used generative AI tools solely to aid and polish manuscript writing and assist with translation. We did not use generative AI tools for any other purpose described in the ICLR 2027 AI Policy for Authors. All AI-assisted text was manually reviewed and verified by the authors for factual accuracy and originality. We take responsibility for the final content of this work, including text produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the experimental code, configuration files, and instructions for training and evaluation in the supplementary material. Appendix A describes the allocation mechanisms and implementation details. Appendix B documents the experimental protocols, reference schedules, ablation configurations, and training-iteration accounting used for the reported results. Together, these materials support reproduction of our experiments.

REFERENCES

Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. PODNet: Pooled outputs distillation for small-tasks incremental learning. In *European Conference on Computer Vision*, 2020. URL https://arxiv.org/abs/2004.13513.

Jiangpeng He, Zhihao Duan, and Fengqing Zhu. CL-LoRA: Continual low-rank adaptation for rehearsal-free class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://arxiv.org/abs/2505.24816.

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*, 2021a. URL https://arxiv.org/abs/2006.16241.

Dan Hendrycks, Kevin Zhao, Steven Basart, Jacob Steinhardt, and Dawn Song. Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021b. URL https://arxiv.org/abs/1907.07174.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

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

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://cave.cs.toronto.edu/kriz/learning-features-2009-TR.pdf.

Wei Li, Hangjie Yuan, Zixiang Zhao, Borui Kang, Ziwei Liu, and Tao Feng. A faster path to continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026. URL https://arxiv.org/abs/2604.11064.

Ameya Prabhu, Hasan Abed Al Kader Hammoud, Puneet K. Dokania, Philip H. S. Torr, Ser-Nam Lim, Bernard Ghanem, and Adel Bibi. Computationally budgeted continual learning: What does matter? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. URL https://arxiv.org/abs/2303.11165.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. URL https://arxiv.org/abs/1611.07725.

Minhyuk Seo, Hyunseo Koh, and Jonghyun Choi. Budgeted online continual learning by adaptive layer freezing and frequency-based sampling. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.15143.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. URL https://arxiv.org/abs/2211.13218.

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 *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025. URL https://arxiv.org/abs/2412.09441.

Yan Wang, Da-Wei Zhou, and Han-Jia Ye. Integrating task-specific and universal adapters for pre-trained model-based class-incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025. URL https://arxiv.org/abs/2508.08165.

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. DualPrompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, 2022a. URL https://arxiv.org/abs/2204.04799.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022b. URL https://arxiv.org/abs/2112.08654.

Gengwei Zhang, Liyuan Wang, Guoliang Kang, Ling Chen, and Yunchao Wei. SLCA: Slow learner with classifier alignment for continual learning on a pre-trained model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023a. URL https://arxiv.org/abs/2303.05118.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=lq62uWRJjiY.

Wenxuan Zhang, Youssef Mohamed, Bernard Ghanem, Philip H. S. Torr, Adel Bibi, and Mohamed Elhoseiny. Continual learning on a diet: Learning from sparsely labeled streams under constrained computation. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Xvfz8NHmCj.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://arxiv.org/abs/2403.12030.

A Allocation Details

A.1 Horizon Mapping and Budget Allowances

In Eq. (3), rounding is to the nearest integer, with half-integer ties away from zero. Effective epoch bounds include the probe and do not exceed the original stage horizon. Historical means and standard deviations use earlier tasks only. A new task cannot change the reference statistics used to normalize its own demand.

For $m_{t,s}$ iterations per epoch, let $B^{\mathrm{des}}_{t,s} = m_{t,s}E^{\mathrm{des}}_{t,s}$. With completed probe count $P^{\mathrm{iter}}_{t,s}$, original horizon $B^{\mathrm{orig}}_{t,s}$, and available allowance $A_{t,s}$, the execution constraint is

$$B_{t,s} = \max\left\{P^{\mathrm{iter}}_{t,s}, \min(B^{\mathrm{des}}_{t,s}, B^{\mathrm{orig}}_{t,s}, A_{t,s})\right\}. \tag{8}$$

For an enabled Adapter or CA balance $K_{t-1,s}$, the affordable epoch count and balance update are

$$E_{t,s} = \min\left(E^{\mathrm{des}}_{t,s}, E^0_s + \left\lfloor \frac{K_{t-1,s}}{m_{t,s}} \right\rfloor\right), \qquad K_{t,s} = \max\{0, K_{t-1,s} + m_{t,s}E^0_s - B^{\mathrm{used}}_{t,s}\}. \tag{9}$$

The balance starts at zero. Adapter and CA maintain separate histories and balances. A disabled balance leaves the bounded desired horizon unchanged. Reference horizons, bounds, and controller sensitivities are explicit hyperparameters; historical calibration adapts allocations within these settings.

CL-LoRA task demand. Let L^{CE}_k and L^{KD}_k be window-averaged probe losses, and g^{CE}_k, g^{KD}_k their relative improvements. The implementation divides the loss difference by $\max(|L_{k-1}|, 10^{-8})$. With the best preceding KD loss $L^{\mathrm{KD,min}}_{<k}$, define

$$r^{\mathrm{KD}}_k = \frac{[L^{\mathrm{KD}}_k - L^{\mathrm{KD,min}}_{<k}]_+}{|L^{\mathrm{KD,min}}_{<k}| + \epsilon}, \tag{10}$$

$$D_t = \mathrm{median}_{k\in\mathcal{I}_t}\left(b_{\mathrm{CE}}[g^{\mathrm{CE}}_k]_+ + b_{\mathrm{KD}}[g^{\mathrm{KD}}_k]_+ + b_{\mathrm{reb}}r^{\mathrm{KD}}_k\right), \tag{11}$$

$$\rho^{\mathrm{des}}_t = \mathrm{clip}_{[\rho_{\min},\rho_{\max}]}\left(\rho_0 + \Delta_\rho \tanh(\mathcal{N}_B(D_t))\right). \tag{12}$$

Here $\mathcal{I}_t$ contains the final eligible probe windows; $\mathcal{N}_B$ uses earlier-task means and standard deviations with a scale floor and clipping. The desired iteration count is the rounded product $\rho^{\mathrm{des}}_t B^{\mathrm{orig}}_t$, clipped to the integer ratio bounds, then reconciled with the allowance and completed probe. The first task uses the original full-training schedule. Loss rebound increases the budget-demand proxy, allowing more training to be requested; its role in branch routing is different, as described below.

CL-LoRA credit policy. Exclude the exempt first task and let $N_{<t}$ and $U_{<t}$ be cumulative original and executed iteration counts. With configured cumulative reference ratio ρ_g, define $Q^-_t = \lfloor \rho_g N_{<t} \rfloor$ and $Q^+_t = \lfloor \rho_g (N_{<t} + B^{\mathrm{orig}}_t) \rfloor$. Existing debt is $d_t = [U_{<t} - Q^-_t]_+$. When bounded future credit is enabled and task t is not the last task, its remaining credit is

$$F_t = [\mathrm{round}(\gamma B^{\mathrm{orig}}_t) - d_t]_+; \qquad A_t = \mathrm{clip}_{[B^{\min}_t, B^{\mathrm{orig}}_t]}(Q^+_t - U_{<t} + F_t). \tag{13}$$

Otherwise $F_t = 0$. Existing debt reduces subsequent credit, preventing the same allowance from being borrowed repeatedly. The minimum-budget safeguard and completed-probe requirements take precedence over the cumulative reference when the constraints conflict.

A.2 CL-LoRA: Budget-Conditioned Joint-to-Specific Routing

CL-LoRA routing uses the recent classification and distillation loss dynamics. Window CE/KD losses are exponentially smoothed before computing relative improvement and KD rebound. Recent improvements and rebound are separately aggregated by their window medians to obtain Eq. (4). The positive improvement terms measure the case for continuing joint training; the negative rebound term advances the transition toward separate CE/KD branch roles. A rebound can therefore request more overall training through HowMuch while favoring earlier branch separation through Where.

For the assigned task-budget ratio $b_t = B_t/B^{\mathrm{orig}}_t$, a structural center is

$$\begin{aligned} c_t &= \mathrm{clip}_{[r_{\min},r_{\max}]}\big(c_{\mathrm{full}} + (c_{\mathrm{zero}} - c_{\mathrm{full}})(1 - b_t)\big), \\ r^-_t &= \max(r_{\min}, c_t - \Delta_r), \qquad r^+_t = \min(r_{\max}, c_t + \Delta_r). \end{aligned} \tag{14}$$

The center specifies a feasible decision region rather than a fixed switch. At an eligible check, earlier tasks' first eligible routing scores form a frozen reference set $\mathcal{H}_t$. For sufficient history,

$$\begin{aligned} z_{t,k} &= \mathrm{clip}_{[-z_r,z_r]}\left(\frac{u_{t,k} - \mathrm{median}(\mathcal{H}_t)}{\max(1.4826\,\mathrm{MAD}(\mathcal{H}_t), \eta|\,\mathrm{median}(\mathcal{H}_t)|, \epsilon_r)}\right), \\ r_{t,k} &= \mathrm{clip}_{[r^-_t, r^+_t]}\big(c_t + \Delta_r \tanh(z_{t,k})\big). \end{aligned} \tag{15}$$

Insufficient history gives $z_{t,k} = 0$. The first eligible check is the interval-aligned ceiling of $B_t r_t^-$, with safeguards for short horizons. The target $B_t r_{t,k}$ is rounded to a check boundary and limited to leave a final check interval where feasible. When the executed count reaches this target, the controller switches once and retains the separated route. Before eligibility or sufficient observations, joint training continues. The current task contributes one reference score for later tasks and does not alter its own reference set.

After switching, the implementation zeros shared-branch gradients in the classification update and task-specific-branch gradients in the distillation update. It retains the native optimizer calls, momentum, and weight decay. Zeroing a loss gradient therefore controls that loss's contribution; it does not strictly freeze the affected parameters. This differs from the inactive-head semantics of TUNA below.

A.3 MOS: Class-Sample Allocation

MOS redistributes a fixed budget of synthetic features across seen classes during classifier alignment. The procedure estimates class-level statistics, converts them into sampling probabilities or integer quotas, and corrects the loss to retain the class-balanced objective. The classifier remains jointly optimized, and the sample count per CA epoch is unchanged. Additional MOS settings for the component comparisons are provided in Appendix B.2.

Class statistics and preferences. Let $g_i = \nabla_\phi \ell(h_\phi(\tilde{x}_i), y_i)$ be the unweighted per-example cross-entropy gradient with respect to the trainable linear-head parameters. For proposal sampling, a gradient-based preference is $a_c = \sqrt{v_c + \epsilon}$, where v_c is the EMA of the class mean $\|g_i\|_2^2$. Alternatively, a loss-based preference uses the EMA class loss $\bar{\ell}_c$: standardize $\log(1 + \bar{\ell}_c)$ across classes, clip to $[-2, 2]$ to obtain ξ_c, and set $a_c = \exp(\xi_c/\tau)$ with $\tau > 0$. Numerically constant log-loss values give $\xi_c = 0$. For exact gradient second moments at a fixed classifier, the second moment of the importance-weighted estimator is minimized by $p_c \propto q_c\sqrt{v_c}$, motivating the gradient-based preference.

Integer-quota allocation uses within-class gradient variance. For each observed class in a minibatch, it estimates the trace of the sample covariance of g_i with denominator $m_c - 1$, where $m_c \geq 2$ is the class count. These estimates are pooled over the epoch with weights $m_c - 1$ to give $\widehat{s}_c$. The smoothed score is updated as $s_c \leftarrow 0.5 s_c + 0.5\widehat{s}_c$, initialized from the first epoch. Only minibatches containing the class contribute to its statistics.

Fixed-budget allocation. Let C denote the number of seen classes, $q_c = 1/C$, and M the native number of synthetic features per CA epoch. Proposal sampling uses

$$p_c = (1 - \alpha)q_c + \alpha \frac{a_c}{\sum_j a_j}, \qquad \mathbf{n} \sim \text{Multinomial}(M, \mathbf{p}), \qquad 0 \leq \alpha < 1. \tag{16}$$

The proposal is computed from a balanced probe and then held fixed; fresh counts summing to M are sampled each subsequent epoch. Insufficient valid probe observations for any class give $p_c = q_c$. The uniform component preserves support for every class.

For integer quotas, let B be the CA minibatch size, with $M = CB$. The first three CA epochs use $n_c = B$. Before each subsequent epoch, the allocator uses scores from completed epochs, initializes every quota to two, and repeatedly applies

$$c^\star \in \arg\max_c s_c \left(\frac{1}{n_c} - \frac{1}{n_c + 2} \right), \qquad n_{c^\star} \leftarrow n_{c^\star} + 2, \tag{17}$$

until $\sum_c n_c = M$. Ties use a rotating class order; identical scores give $n_c = B$. Each increment reduces the fixed-statistics proxy $\sum_c s_c/n_c$. Positive even quotas preserve the total exactly, without a separate rounding step. Same-class sample pairs are randomly permuted and packed into C minibatches of size B, so an observed class has at least two examples in a minibatch, but not every class must appear in every minibatch. Both allocation rules retain the native CA minibatch size and number of updates per epoch.

Balanced-objective correction. Write $r_c = p_c$ for proposal sampling and $r_c = n_c/M$ for integer quotas. For a minibatch $\mathcal{B}$ of size m, training uses

$$\widehat{\mathcal{L}}_{\mathrm{CA}} = \frac{1}{m} \sum_{i \in \mathcal{B}} \frac{q_{y_i}}{r_{y_i}} \ell(h_\phi(\tilde{x}_i), y_i). \tag{18}$$

Weights are not normalized by their realized minibatch sum. For proposal sampling, at fixed classifier parameters and class-conditional feature distributions,

$$\mathbb{E}_{c \sim p, \tilde{x} \sim P_c} \left[\frac{q_c}{p_c} \nabla_\phi \ell(h_\phi(\tilde{x}), c) \right] = \sum_c q_c \, \mathbb{E}_{\tilde{x} \sim P_c} [\nabla_\phi \ell(h_\phi(\tilde{x}), c)]. \tag{19}$$

For integer quotas, the weight is B/n_c, using the epoch quota rather than the realized minibatch count. At fixed ϕ, averaging the minibatch losses over the complete epoch gives

$$\frac{1}{C} \sum_{c=1}^{C} \frac{1}{n_c} \sum_{i : y_i = c} \ell(h_\phi(\tilde{x}_i), c). \tag{20}$$

Thus, the allocation changes sampling effort while retaining equal class weights in the target objective.

A.4 TUNA: Coverage and Soft Quotas for Classifier Heads

Our TUNA instantiation groups classifier parameters by the task that introduced their output classes. The evaluated groups are equally sized, and $K = \lceil \kappa H \rceil$ groups are selected per routed step, with $1 \le K \le H$. Head scores use the squared gradient of the minibatch loss divided by the group's parameter count. This is distinct from MOS's per-example class-gradient statistics.

The probe trains all classifier groups and initializes their scores. Routed steps alternate between a rotating coverage selection and a priority selection. For priority selection, Eq. (7) uses scores normalized by their maximum with a small positive numerical floor; all-zero scores give $\pi_h = K/H$. The capped proportional construction chooses λ so that the rates sum to K. These rates are target inclusion frequencies for priority steps, rather than class-sampling probabilities.

Let a_h be a quota accumulator, initially zero. At a priority step, first set $a_h \leftarrow a_h + \pi_h$. Select the K largest accumulators, using rotating tie-breaking, and subtract one from each selected accumulator. This produces exactly K active heads per routed step while allowing priority opportunities to be distributed across heads over time. Finite training horizons need not realize the target frequencies exactly.

At epoch end, gradients observed in coverage steps update the scores by EMA for use in the next epoch. Priority observations do not enter this refresh, avoiding direct reliance on the score-dependent selection for score measurement. No additional full-head probe or backward pass is introduced. Refreshed scores influence training only when a subsequent routed epoch executes; a probe-only stage has no head allocation after the probe.

For an unselected group, parameters have gradient computation disabled and their gradients are set to `None`, so the optimizer skips both parameter and momentum updates for that group. All classifier outputs remain in the full-class cross-entropy, and gradients for active groups are differentiated through the actual cosine classifier, including normalization and scaling. The synthetic class sampler, adapter fusion, and inference rule remain those of the base learner.

A.5 Execution and Counting Convention

The shared procedure trains a probe, installs an executable horizon, applies the appropriate allocation policy, and records the actual executed count. Probe observations are obtained from training losses and gradients. Calibration uses earlier-task information, with separate histories for budget demand and branch-routing demand where required. No validation or test accuracy enters the allocation decisions.

For learner m, let N_m and N_m^{orig} count native training iterations with the same stage coverage. We report the iteration saving as $100(1 - N_m/N_m^{\mathrm{orig}})$. CL-LoRA counts native training batches, which

can contain separate classification and distillation optimizer calls. Adapter and CA iterations are accumulated for MOS and TUNA. Their counts are compared within each learner against its matched original procedure. Parameter-update exposure, arithmetic cost, and elapsed time are different quantities and are not substituted for this iteration count.

B Additional Experimental Details

B.1 Protocols, Reference Schedules, and Counting

Table 3 specifies the protocols and reference schedules for the paired main results. The initial and subsequent tasks contain the same number of classes within each protocol. CL-LoRA uses 20/40/10 tasks on CIFAR-100/ImageNet-R/ImageNet-A, while MOS and TUNA use 20/10/10. Original and AllocAdapt share the task partition and class order within each pair. The different ImageNet-R partitions preclude interpreting the table as a direct accuracy ranking across base learners.

Table 3: Main-result protocols and iteration counts. T is the number of tasks; C/t is classes per task, including the initial task. A/CA denote Adapter/CA epochs. CA is absent from the initial task. Counts include probe training.

Learner	Dataset	T	C/t	Reference epochs	Original N	Full N
CL-LoRA	CIFAR-100	20	5	10	15,800	6,794
	ImageNet-R	40	5	20	15,420	12,281
	ImageNet-A	10	20	20	3,820	3,243
MOS	CIFAR-100	20	5	20A/30CA	52,550	18,937
	ImageNet-R	10	20	10A/30CA	47,490	23,549
	ImageNet-A	10	20	20A/30CA	34,960	8,241
TUNA	CIFAR-100	20	5	10A/30CA	47,150	19,426
	ImageNet-R	10	20	10A/30CA	39,960	13,048
	ImageNet-A	10	20	20A/30CA	36,220	7,231

Original refers to the full reference schedule used by our implementation. For TUNA/CIFAR-100, the paired reference uses 10A/30CA with batch size 32; the published setting uses 15A/30CA with batch size 16.

Results in Tables 1 and 2 are averaged over multiple independent runs with different random seeds. Within each run, Avg is computed across successive task evaluations, and Final is the accuracy after the last task. The reported accuracies are then averaged across runs. Iteration savings are computed relative to the matched Original run and subsequently averaged across runs.

Iteration units. Counts include the initial task and probe training. CL-LoRA retains its full initial-task schedule, whereas MOS/TUNA may shorten initial Adapter training. One native CL-LoRA iteration can contain separate classification and distillation optimizer calls. MOS/TUNA counts correspond to their native training iterations, including CA iterations on synthetic features. Adapter and CA counts are added without cost weighting; their individual computational costs can differ. Head routing changes the updated parameter groups and is not assigned an additional iteration saving beyond the counted iterations.

Cumulative curves. For method m, the lower curves in Figure 2 report

$$C_t^{(m)} = 100 \frac{\sum_{i=1}^{t} N_i^{(m)}}{\sum_{i=1}^{T} N_i^{(\text{Original})}}. \tag{21}$$

The denominator is the complete Original run, not its count through task t. Original therefore reaches 100, while AllocAdapt reaches $100 - S$. Full counts follow completed-run summaries; reference schedules supply Original counts where native counters are unavailable. The horizontal coordinate $100t/T$ aligns progress across different task counts without resampling accuracy observations. The upper curves show a_t, not a running average of earlier accuracies.

B.2 Component Comparison Configuration

We compare CL-LoRA and MOS on CIFAR-100 and ImageNet-R. Table 4 reports the iteration counts corresponding to the accuracy and savings results in Table 2.

Table 4: Iteration counts for the component comparisons. CL-LoRA Static and MOS/ImageNet-R Static counts follow their fixed schedules; other counts are from completed-run counters.

Learner	Dataset	Static	HowMuch-only	Where-only	Full
CL-LoRA	CIFAR-100	6,794	6,753	6,794	6,794
	ImageNet-R	12,412	12,362	12,412	12,281
MOS	CIFAR-100	18,990	18,937	18,990	18,937
	ImageNet-R	23,745	23,529	23,745	23,549

CL-LoRA. Static retains 40% of the original incremental-task iterations on CIFAR-100 and 80% on ImageNet-R, while keeping the original initial-task schedule. Static and HowMuch-only use the same fixed Joint-to-Specific routing ratio within each dataset: 0.70 on CIFAR-100 and 0.60 on ImageNet-R. The variants with Where enabled use the budget-conditioned routing policy described in Appendix A.2.

MOS. Static uses native MOS with 12A/6CA on CIFAR-100 and 5A/15CA on ImageNet-R. The adaptive variants use the allocation-enabled implementation. When Where is enabled, CIFAR-100 uses gradient-based class sampling with $\alpha = 0.25$, and ImageNet-R uses variance-guided class quotas. Full and HowMuch-only share the same HowMuch configuration, probe procedure, and all other training settings. They differ only in whether adaptive class allocation is enabled during classifier alignment.

B.3 Additional Allocation Analysis

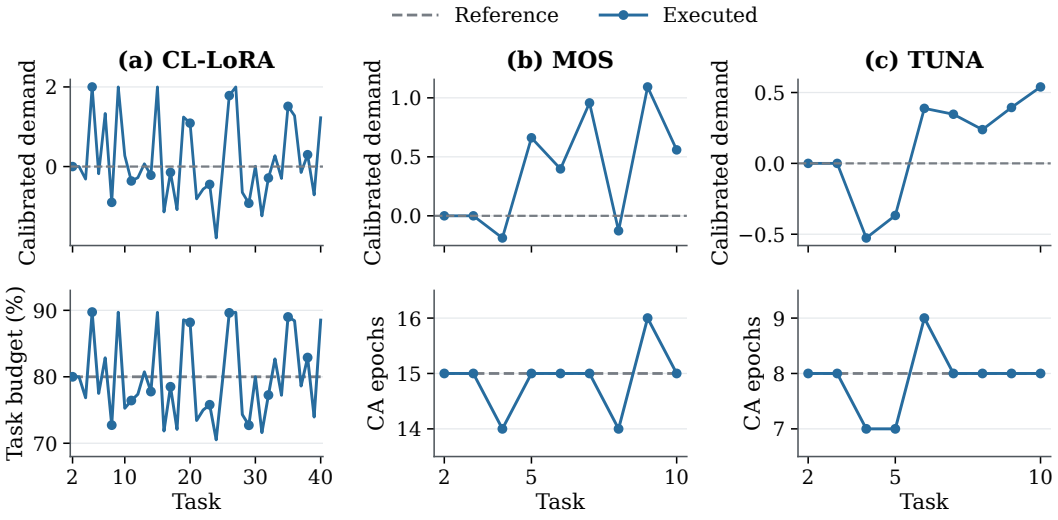

Figure 5: Full HowMuch trajectories on ImageNet-R. Top: historically calibrated demand. Bottom: configured reference and executed horizons, including probes and budget constraints. CL-LoRA budgets are percentages of each task's original iterations; MOS/TUNA CA budgets are epochs. The initial task is omitted.

HowMuch trajectories. Figure 5 relates historically calibrated demand to the executed horizons on ImageNet-R. The configured reference horizons are 80% of the original task iterations for CL-

LoRA, 15 CA epochs for MOS, and eight CA epochs for TUNA. Historical statistics calibrate the current probe signals, with zero scores during the initial calibration warm-up.

CL-LoRA executes 70.5–89.7% of each incremental task's original iterations, while MOS and TUNA use 14–16 and 7–9 CA epochs, respectively. The controller maps calibrated demand to feasible horizons through integer rounding, admissible ranges, and the available budget. These trajectories illustrate task-dependent allocation within the configured training constraints.

Reading the Where allocations. Figure 4 uses different units for the three interfaces. CL-LoRA's joint phase occupies 61.2–82.3% of its executed horizon. During the routed phase, classification gradients update the task-specific branch and distillation gradients update the shared branch. For MOS, let $n_{t,c}$ be the post-probe sample count of class c, M_t the total count, and K_t the number of observed classes. The heatmap reports $r_{t,c} = n_{t,c}/(M_t/K_t)$, whose mean over observed classes is one. The sample total for the assigned horizon is unchanged. TUNA reports $f_{t,h} = 100u_{t,h}/U_t^{\text{route}}$, where $u_{t,h}$ counts routed iterations updating head h. All heads participate in forward computation and loss, and several heads may update in one iteration; row totals need not equal 100%. Gray cells indicate classes or heads that have not yet appeared.

The TUNA allocation plot shows the Full configuration evaluated in Table 1.