MAKE EVERY PARAMETER COUNT: ON-DEVICE SHARED-SUBSPACE STREAMING LORA MERGING

Anonymous authors
Paper under double-blind review

ABSTRACT

Edge-deployed pretrained models typically use LoRA to continually extend their task capabilities. However, retaining a separate adapter for each task can quickly exhaust the limited storage budget on edge devices. Prior work has explored adapter merging to reduce storage costs, but often struggles to avoid interference between task-specific information after merging under a limited budget. Shared-subspace methods can preserve finer-grained task differences through shared bases and task-specific coefficients. However, these methods struggle both to maintain efficient updates and to fully utilize the limited storage budget. We therefore propose MEPC (Make Every Parameter Count), a shared-subspace method for streaming LoRA merging. MEPC uses incremental subspace updates to reduce update overhead and performs importance-aware compression based on joint deletion loss, allocating the limited parameters preferentially to preserving more important cross-task information. Experiments across multiple storage budgets and task-stream settings show that MEPC substantially outperforms baselines without shared subspaces. Compared with the strongest shared-subspace baseline, MEPC improves performance by up to 3.13 percentage points, achieves a per-step update speedup of up to $28.34\times$, and reduces peak memory usage by up to 66.2%.

1 INTRODUCTION

Deploying pretrained models on edge devices can reduce latency and the exposure of sensitive data, and has become an important paradigm in edge intelligence (Dhar et al., 2021). Given device resource constraints, edge devices typically use smaller models and acquire task-specific capabilities through parameter-efficient adaptation methods such as LoRA (Hu et al., 2022; Gunter et al., 2024; Li et al., 2025). However, as new tasks arrive, retaining a separate LoRA adapter for each task incurs growing storage costs that eventually exceed the available capacity of edge devices (Bang et al., 2024). To continually support new tasks within a fixed budget, the system must continually merge existing adapters. In this process, historical task data and previously delivered raw adapters are typically unavailable, so each update can use only the incoming LoRA adapter and the current merged state. A central problem in on-device continual adaptation is therefore how to continually merge LoRA adapters under a fixed storage budget while balancing performance and update efficiency.

Prior work has explored several merging approaches. Conventional merging methods (Wortsman et al., 2022; Ilharco et al., 2023; Yang et al., 2024; Huang et al., 2024) typically assume that all LoRA adapters to be merged are available simultaneously, and some also require additional data. K-Merge (Shenaj et al., 2026) limits storage costs through a fixed number of slots, but as tasks accumulate, multiple tasks are forced to share the same slot, increasing inter-task interference. Shared-subspace methods (Gabrielsson et al., 2025; Kaushik et al., 2025) mitigate interference by constructing reusable shared bases and preserving differences between tasks through task-specific coefficients. Share (Kaushik et al., 2026) extends this approach to streaming merging and demonstrates the feasibility of shared-subspace representations in this setting.

However, two key challenges remain for long-term streaming merging under a fixed storage budget. First, Share (Kaushik et al., 2026) relies on explicit reconstruction of historical LoRA adapters during updates, so its computational and temporary memory costs grow rapidly with the number of tasks. Second, as task-specific coefficients accumulate, the capacity available to the shared representation continually shrinks. Simply reducing the ranks of all shared subspaces uniformly cannot

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.

adequately account for differences in the importance of individual directions, making it difficult to retain more critical cross-task information within the limited budget.

To address these challenges, we propose MEPC (Make Every Parameter Count), a shared-subspace method for streaming LoRA merging under a fixed storage budget. MEPC reuses directions already covered by the shared subspaces and incorporates only the incremental information in each incoming LoRA adapter, avoiding explicit reconstruction of historical LoRA adapters. MEPC further introduces importance-aware compression based on joint deletion loss to assess the importance of different directions in terms of the full weight update and retain more important cross-task information as much as possible within the limited budget. By combining incremental updates with importance-aware compression, MEPC achieves both better overall performance and lower update overhead under a fixed budget. Our main contributions are as follows:

- We propose MEPC, a shared-subspace method for streaming LoRA merging under a fixed storage budget. MEPC continually merges incoming LoRA adapters without retaining historical task data or raw adapters.
- We develop an incremental update mechanism based on subspace reuse and residual decomposition to reduce streaming update overhead. We further use joint deletion loss to assess directional importance and guide compression, thereby preferentially preserving critical cross-task information.
- Experiments show that MEPC consistently outperforms existing methods in overall performance. Compared with the strongest shared-subspace baseline, MEPC improves performance by up to 3.13 percentage points, achieves a per-step update speedup of up to $28.34\times$, and reduces peak memory usage by up to 66.2%.

2 Related Work

Parameter-efficient fine-tuning. Parameter-efficient fine-tuning lowers task adaptation costs by training only a few parameters. (Ding et al., 2023; Han et al., 2024). LoRA expresses weight updates as products of two low-rank matrices while freezing the base model (Hu et al., 2022). Subsequent work further improves the parameter efficiency and expressiveness of LoRA through adaptive rank allocation, parameterization designs, and optimization strategies (Zhang et al., 2023; Zi et al., 2023; Liu et al., 2024; Kopiczko et al., 2024; Renduchintala et al., 2024).

Continual learning. Continual learning aims to acquire knowledge from a task sequence while mitigating catastrophic forgetting (De Lange et al., 2022). Experience replay methods preserve prior knowledge by storing and reusing previous-task samples, incurring additional storage (Rebuffi et al., 2017; Buzzega et al., 2020). Rehearsal-free methods instead use prompt parameters or low-rank adaptation to learn new tasks while freezing the pretrained model (Wang et al., 2022b; Smith et al., 2023; Liang & Li, 2024). Share continually updates a shared subspace to integrate task data or pretrained LoRA adapters without retaining separate historical adapters (Kaushik et al., 2026).

Model merging. Model merging combines task-specific models or adapters into a single model. Early methods use weight averaging or task-vector arithmetic (Wortsman et al., 2022; Ilharco et al., 2023), while later methods improve merging performance by resolving parameter conflicts or learning merging coefficients (Yadav et al., 2023; Yu et al., 2024; Yang et al., 2024). Recent work develops specialized merging strategies for LoRA (Huang et al., 2024; Zhao et al., 2025; Zhang & Zhou, 2025; Tang & Yang, 2026; Jeong et al., 2026) or jointly compresses multiple adapters by extracting shared low-rank structures across adapters (Gabrielsson et al., 2025; Kaushik et al., 2025; Tang et al., 2025b). Other work studies model merging under sequential task arrival (Dziadzio et al., 2025; Tang et al., 2025a; Shenaj et al., 2026; Wei et al., 2026). However, these methods do not jointly address streaming updates and importance-aware compression under a fixed storage budget.

3 Method

3.1 Problem Formulation

Let $h(\cdot; \mathbf{W}_0)$ denote a pretrained model with frozen parameters, and let $\{\tau_1, \tau_2, \ldots\}$ denote a sequence of tasks arriving over time. Each task τ_t is associated with an independently trained LoRA

adapter $\mathcal{L}_t$. At time step t, the system can access only the incoming LoRA adapter $\mathcal{L}_t$ and the merged state $\mathcal{S}^{(t-1)}$ from the previous step; task data and historical LoRA adapters cannot be revisited. Let $\mathcal{S}^{(t)}$ denote the merged state after the update at time step t. The device has a fixed budget b for adapter parameters, and the merged state must satisfy $\mathrm{params}(\mathcal{S}^{(t)}) \le b,\ \forall t$. Let $\widehat{\mathcal{L}}_i^{(t)} = \psi(\mathcal{S}^{(t)}, i)$ denote the adapter for task τ_i reconstructed from $\mathcal{S}^{(t)}$, where ψ is a deterministic reconstruction map. We assume that the task identity i is available at inference time. Let $m_i(\cdot)$ denote the evaluation metric for task τ_i, where higher values indicate better performance. The task performance retention ratio for τ_i at time step t is defined as $\rho_i^{(t)} = \frac{m_i\left(\widehat{\mathcal{L}}_i^{(t)}\right)}{m_i(\mathcal{L}_i)}$. To account for differences in scale across task-specific metrics, the aggregate score $s^{(t)}$ is defined as the mean of the task performance retention ratios over all tasks observed by time step t:

$$s^{(t)} = \frac{1}{t}\sum_{i=1}^{t} \rho_i^{(t)}. \tag{1}$$

Our objective is to maximize $s^{(t)}$ at each step under the fixed storage budget.

3.2 MEPC

MEPC compactly represents all observed LoRA adapters using shared bases and task-specific coefficients. It initializes the shared representation from the first LoRA adapter and then updates it incrementally. When a new LoRA adapter arrives, MEPC appends only the new directions in its orthogonal residuals to the corresponding bases and updates the task-specific coefficients accordingly. If the expanded representation exceeds the storage budget, MEPC selects the direction with the smallest one-sided deletion loss from each shared basis as a local candidate and computes the joint deletion loss for each candidate. It then removes the direction with the smallest joint deletion loss among all local candidates and repeats this process until the storage constraint is satisfied.

3.2.1 Shared Representation and Initialization

Let $\mathcal{M}$ denote the set of modules to which LoRA is applied, and let $\mathcal{F} = \{\mathrm{A}, \mathrm{B}\}$ denote the set of LoRA factor sides. For each module $m \in \mathcal{M}$, let $\mathbf{W}_{0,m} \in \mathbb{R}^{n_m \times d_m}$ be its frozen pretrained weight, where n_m and d_m are the output and input dimensions, respectively. The LoRA adapter for task τ_i at module m consists of low-rank factors $\mathbf{B}_{i,m} \in \mathbb{R}^{n_m \times r_i}$ and $\mathbf{A}_{i,m} \in \mathbb{R}^{r_i \times d_m}$, yielding the weight update $\Delta\mathbf{W}_{i,m} = \mathbf{B}_{i,m}\mathbf{A}_{i,m}$, where r_i is the LoRA rank for task τ_i.

MEPC represents the LoRA factors at each module using bases shared across tasks and task-specific coefficients. At time step t, module m has two column-orthonormal shared bases, $\mathbf{U}_{m,\mathrm{A}}^{(t)} \in \mathbb{R}^{d_m \times k_{m,\mathrm{A}}^{(t)}}$ and $\mathbf{U}_{m,\mathrm{B}}^{(t)} \in \mathbb{R}^{n_m \times k_{m,\mathrm{B}}^{(t)}}$, where $k_{m,\mathrm{A}}^{(t)}$ and $k_{m,\mathrm{B}}^{(t)}$ are the dimensions of the corresponding shared subspaces. For each observed task τ_i, $i \le t$, MEPC stores coefficients $\mathbf{C}_{i,m,\mathrm{A}}^{(t)} \in \mathbb{R}^{k_{m,\mathrm{A}}^{(t)} \times r_i}$ and $\mathbf{C}_{i,m,\mathrm{B}}^{(t)} \in \mathbb{R}^{k_{m,\mathrm{B}}^{(t)} \times r_i}$ and reconstructs its LoRA factors as

$$\widehat{\mathbf{A}}_{i,m}^{(t)} = \left(\mathbf{U}_{m,\mathrm{A}}^{(t)}\mathbf{C}_{i,m,\mathrm{A}}^{(t)}\right)^{\top}, \quad \widehat{\mathbf{B}}_{i,m}^{(t)} = \mathbf{U}_{m,\mathrm{B}}^{(t)}\mathbf{C}_{i,m,\mathrm{B}}^{(t)}. \tag{2}$$

The corresponding weight update is

$$\widehat{\Delta\mathbf{W}}_{i,m}^{(t)} = \mathbf{U}_{m,\mathrm{B}}^{(t)}\mathbf{C}_{i,m,\mathrm{B}}^{(t)}\left(\mathbf{C}_{i,m,\mathrm{A}}^{(t)}\right)^{\top}\left(\mathbf{U}_{m,\mathrm{A}}^{(t)}\right)^{\top}. \tag{3}$$

When the first LoRA adapter arrives, MEPC applies thin QR factorizations to $\mathbf{A}_{1,m}^{\top}$ and $\mathbf{B}_{1,m}$ at each module, producing the initial shared bases and task-specific coefficients:

$$\mathbf{A}_{1,m}^{\top} = \mathbf{U}_{m,\mathrm{A}}^{(1)}\mathbf{C}_{1,m,\mathrm{A}}^{(1)}, \quad \mathbf{B}_{1,m} = \mathbf{U}_{m,\mathrm{B}}^{(1)}\mathbf{C}_{1,m,\mathrm{B}}^{(1)}.$$

3.2.2 Incremental Subspace Update

Suppose MEPC has merged the LoRA adapters for the first t tasks and receives the incoming LoRA adapter at time step $t+1$: $\mathcal{L}_{t+1} = \{(\mathbf{A}_{t+1,m}, \mathbf{B}_{t+1,m})\}_{m\in\mathcal{M}}$. We describe the update for the B-side factor of an arbitrary module $m \in \mathcal{M}$. MEPC projects the new factor $\mathbf{B}_{t+1,m}$ onto the current

shared subspace and computes its orthogonal residual:

$$\mathbf{E}_{m,\mathrm{B}} = \left(\mathbf{I} - \mathbf{U}_{m,\mathrm{B}}^{(t)} \left(\mathbf{U}_{m,\mathrm{B}}^{(t)}\right)^{\top}\right) \mathbf{B}_{t+1,m}. \tag{4}$$

If the residual is nonzero, MEPC computes the QR factorization $\mathbf{E}_{m,\mathrm{B}} = \mathbf{Q}_{m,\mathrm{B}}\mathbf{R}_{m,\mathrm{B}}$, where the columns of $\mathbf{Q}_{m,\mathrm{B}}$ form an orthonormal basis for the column space of the residual. MEPC appends these new directions to the shared basis: $\widetilde{\mathbf{U}}_{m,\mathrm{B}} = \left[\mathbf{U}_{m,\mathrm{B}}^{(t)}, \mathbf{Q}_{m,\mathrm{B}}\right]$. Because the residual is orthogonal to the current shared basis, the expanded basis remains column-orthonormal. This update uses only the current shared basis and the incoming LoRA factor, without reconstructing or refactorizing historical LoRA factors.

For each previous task τ_i, $i \leq t$, the existing coordinates remain unchanged and are zero-padded along the new directions: $\widetilde{\mathbf{C}}_{i,m,\mathrm{B}} = \begin{bmatrix} \mathbf{C}_{i,m,\mathrm{B}}^{(t)} \\ \mathbf{0} \end{bmatrix}$. The basis expansion therefore leaves the reconstructions for previous tasks unchanged. The coefficients for the new task are obtained by concatenating the projection coefficients and residual-factorization coefficients:

$$\widetilde{\mathbf{C}}_{t+1,m,\mathrm{B}} = \begin{bmatrix} \left(\mathbf{U}_{m,\mathrm{B}}^{(t)}\right)^{\top} \mathbf{B}_{t+1,m} \\ \mathbf{R}_{m,\mathrm{B}} \end{bmatrix}. \tag{5}$$

MEPC applies the same update to $\mathbf{A}_{t+1,m}^{\top}$. If the number of persisted parameters after the update exceeds b, MEPC applies importance-aware compression to the shared representation until the storage constraint is satisfied.

3.2.3 Importance-Aware Compression

After an incremental update, the number of persisted parameters in the expanded representation may exceed the storage budget. MEPC first selects the local candidate with the smallest one-sided deletion loss from each shared basis. It then compares the candidates globally according to their joint deletion losses and progressively removes the direction with the smallest joint deletion loss.

Local selection identifies the unit direction whose removal has the smallest effect on the one-sided factor reconstructions of all observed tasks. For module $m \in \mathcal{M}$ and factor side $f \in \mathcal{F}$, any unit direction in the shared subspace can be written as $\mathbf{q} = \widetilde{\mathbf{U}}_{m,f}\mathbf{v}$, where $\mathbf{v}$ is its coordinate vector in the current basis and $\|\mathbf{v}\|_2 = 1$. Removing $\mathbf{q}$ deletes the orthogonal projection of each task's factor representation onto this direction. MEPC defines the one-sided deletion loss as the sum of the squared Frobenius norms of these projections:

$$\mathcal{E}_{m,f}(\mathbf{v}) = \sum_{i=1}^{t+1} \left\| \mathbf{q}\mathbf{q}^{\top} \widetilde{\mathbf{U}}_{m,f} \widetilde{\mathbf{C}}_{i,m,f} \right\|_{\mathrm{F}}^2 = \mathbf{v}^{\top} \left(\sum_{i=1}^{t+1} \widetilde{\mathbf{C}}_{i,m,f} \widetilde{\mathbf{C}}_{i,m,f}^{\top} \right) \mathbf{v}. \tag{6}$$

Define the coefficient Gram matrix as $\mathbf{G}_{m,f} = \sum_{i=1}^{t+1} \widetilde{\mathbf{C}}_{i,m,f} \widetilde{\mathbf{C}}_{i,m,f}^{\top}$. Local selection is then equivalent to minimizing $\mathbf{v}^{\top}\mathbf{G}_{m,f}\mathbf{v}$. By the Rayleigh–Ritz theorem, $\mathcal{E}_{m,f}(\mathbf{v})$ is minimized when $\mathbf{v}$ is a unit eigenvector associated with the smallest eigenvalue of $\mathbf{G}_{m,f}$. We therefore eigendecompose $\mathbf{G}_{m,f} = \mathbf{V}_{m,f}\mathbf{\Lambda}_{m,f}\mathbf{V}_{m,f}^{\top}$. The j-th column $\mathbf{v}_{m,f,j}$ of $\mathbf{V}_{m,f}$ is a unit eigenvector of $\mathbf{G}_{m,f}$, and the j-th diagonal entry $\lambda_{m,f,j}$ of $\mathbf{\Lambda}_{m,f}$ is its corresponding eigenvalue. We order the eigenpairs such that $\lambda_{m,f,1} \geq \cdots \geq \lambda_{m,f,\widetilde{k}_{m,f}}$, where $\widetilde{k}_{m,f}$ is the number of columns in the expanded basis. The coordinate vector and direction of the local candidate are $\mathbf{v}_{m,f}^{\mathrm{cand}} = \mathbf{v}_{m,f,\widetilde{k}_{m,f}}$, $\mathbf{q}_{m,f}^{\mathrm{cand}} = \widetilde{\mathbf{U}}_{m,f}\mathbf{v}_{m,f}^{\mathrm{cand}}$.

To make the candidate direction directly removable as a basis vector, MEPC uses $\mathbf{V}_{m,f}$ to transform the shared basis and task-specific coefficients jointly: $\overline{\mathbf{U}}_{m,f} = \widetilde{\mathbf{U}}_{m,f}\mathbf{V}_{m,f}$, $\overline{\mathbf{C}}_{i,m,f} = \mathbf{V}_{m,f}^{\top}\widetilde{\mathbf{C}}_{i,m,f}$.

This orthogonal transformation preserves both the factor reconstructions and the column orthonormality of the basis. Let $\overline{\mathbf{u}}_{m,f,j}$ denote the j-th column of $\overline{\mathbf{U}}_{m,f}$. Its final column satisfies $\overline{\mathbf{u}}_{m,f,\widetilde{k}_{m,f}} = \mathbf{q}_{m,f}^{\mathrm{cand}}$. The Gram matrix of the transformed coefficients satisfies $\sum_{i=1}^{t+1} \overline{\mathbf{C}}_{i,m,f}\overline{\mathbf{C}}_{i,m,f}^{\top} = \mathbf{V}_{m,f}^{\top}\mathbf{G}_{m,f}\mathbf{V}_{m,f} = \mathbf{\Lambda}_{m,f}$. Thus, $\lambda_{m,f,j}$ equals the cumulative one-sided

deletion loss for the j-th transformed basis direction. After deleting a basis vector and the corresponding coefficient rows, the Gram matrix of the remaining coefficients stays diagonal, preserving the losses and their ordering for the remaining directions. Each compression operation therefore requires only one eigendecomposition and a change of basis; subsequent local candidates are obtained by successively selecting the last remaining direction of each basis.

The one-sided deletion loss does not capture the joint effect of both factors on the full LoRA weight update. MEPC therefore compares local candidates globally using the change in the reconstructed weight update caused by their removal. Let $\overline{\mathbf{c}}_{i,m,f,j} \in \mathbb{R}^{r_i}$ be the transpose of the j-th row of $\overline{\mathbf{C}}_{i,m,f}$. Removing the j-th direction on the B side removes the following component from the reconstructed weight update for task τ_i: $\delta\widehat{\mathbf{W}}_{i,m,\mathrm{B},j} = \overline{\mathbf{u}}_{m,\mathrm{B},j} \left(\overline{\mathbf{U}}_{m,\mathrm{A}} \overline{\mathbf{C}}_{i,m,\mathrm{A}} \overline{\mathbf{c}}_{i,m,\mathrm{B},j}\right)^\top$.

MEPC defines the squared Frobenius norm of this component as the task-level joint deletion loss. The column orthonormality of the bases gives

$$\ell_{i,m,\mathrm{B},j} = \left\| \delta\widehat{\mathbf{W}}_{i,m,\mathrm{B},j} \right\|_{\mathrm{F}}^2 = \left\| \overline{\mathbf{C}}_{i,m,\mathrm{A}} \overline{\mathbf{c}}_{i,m,\mathrm{B},j} \right\|_2^2 . \tag{7}$$

Summing over all observed tasks and defining the A-side loss symmetrically gives

$$\ell_{m,\mathrm{B},j} = \sum_{i=1}^{t+1} \left\| \overline{\mathbf{C}}_{i,m,\mathrm{A}} \overline{\mathbf{c}}_{i,m,\mathrm{B},j} \right\|_2^2 , \quad \ell_{m,\mathrm{A},j} = \sum_{i=1}^{t+1} \left\| \overline{\mathbf{C}}_{i,m,\mathrm{B}} \overline{\mathbf{c}}_{i,m,\mathrm{A},j} \right\|_2^2 . \tag{8}$$

Let $k_{m,f}$ denote the current number of columns in a shared basis. If the number of persisted parameters exceeds b, MEPC selects the local candidate with the smallest joint deletion loss across all shared bases:

$$(m^\star, f^\star) \in \operatorname*{argmin}_{m \in \mathcal{M},\ f \in \mathcal{F}: k_{m,f} > 0} \ell_{m,f,k_{m,f}} . \tag{9}$$

MEPC removes the selected basis vector and the corresponding row from every task-specific coefficient matrix, and updates the current rank and number of persisted parameters. If the representation still exceeds the budget, the new last direction of the modified basis automatically becomes its next local candidate. MEPC updates only the joint deletion losses of this candidate and the candidate on the other factor side of the same module. This process continues until the storage budget is satisfied, yielding adaptive rank allocations across modules and factor sides. The complete procedure is given in Algorithm 1 in the appendix.

3.3 Complexity Analysis

Time complexity. Let $g = 2|\mathcal{M}|$ denote the number of shared bases, $d = \max_{m \in \mathcal{M}} \{d_m, n_m\}$ the maximum feature dimension, k the maximum expanded shared rank, $r = r_{t+1}$ the rank of the incoming LoRA adapter, and $q_{t+1} = \sum_{i=1}^{t+1} r_i$. Computing the projections, orthogonal residuals, and QR factorizations for all shared bases costs $\mathcal{O}(gdr(k+r))$. If the expanded representation exceeds the storage budget, constructing the coefficient Gram matrices, performing their eigendecompositions, and transforming the shared bases and task-specific coefficients costs $\mathcal{O}\big(g[k^2(q_{t+1}+d) + k^3]\big)$. Computing the joint deletion losses for the initial candidate directions costs $\mathcal{O}(gkq_{t+1})$. Let h_{t+1} denote the number of shared directions removed at the current time step. Because only the affected candidate losses are updated after each deletion, the subsequent iterations cost $\mathcal{O}(h_{t+1}(kq_{t+1}+g))$. The total time complexity of an update that triggers importance-aware compression is therefore

$$\mathcal{O}\big(g\left[dr(k+r) + k^2(q_{t+1}+d) + k^3 + kq_{t+1}\right] + h_{t+1}(kq_{t+1}+g)\big) . \tag{10}$$

Without compression, each update costs $\mathcal{O}(gdr(k+r))$. The high-dimensional projections and residual factorizations involve only the current shared bases and the incoming LoRA adapter.

Persistent storage. At time step t, MEPC persists the shared bases on both factor sides and the task-specific coefficients for all observed tasks. Concatenating the A-side and B-side task-specific coefficients yields matrices of dimensions $k_{m,\mathrm{A}}^{(t)} \times q_t$ and $k_{m,\mathrm{B}}^{(t)} \times q_t$, respectively. The number of persisted parameters at time step t is therefore

$$b_{\mathrm{cur}}^{(t)} = \sum_{m \in \mathcal{M}} \left[k_{m,\mathrm{A}}^{(t)} \left(d_m + q_t\right) + k_{m,\mathrm{B}}^{(t)} \left(n_m + q_t\right) \right] . \tag{11}$$

The budget requires $b_{\text{cur}}^{(t)} \leq b$ at each step. As q_t grows, the total shared rank must decrease accordingly. MEPC reaches structural task capacity when every shared basis reaches minimum rank.

Effective task capacity. Structural capacity measures only the number of tasks that a representation can accommodate, without considering whether the retained tasks still satisfy the performance requirement. We therefore introduce effective task capacity:

$$C_\theta(\mathcal{A}, b; \boldsymbol{\tau}) = \sup\left(\left\{T \in \mathbb{N} : \rho_i^{(t)} \geq \theta, \qquad \forall\, 1 \leq i \leq t \leq T\right\} \cup \{0\}\right). \tag{12}$$

Here, $\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots)$ is the task stream, $\mathcal{A}$ is a streaming LoRA merging method with storage budget b, and θ is the performance threshold. Effective task capacity is the maximum number of tasks a method can accommodate while all observed tasks satisfy the threshold, and thus better reflects storage efficiency than structural capacity. We therefore focus on effective rather than structural capacity. Section 4.4 shows that MEPC achieves higher effective task capacity across thresholds, indicating greater efficiency in storing task information.

4 Experiments

4.1 Experimental Setup

Models and tasks. For language experiments, we use Llama-3.2-1B-Instruct, Qwen2.5-1.5B-Instruct, and Mistral-7B-Instruct-v0.2 (Grattafiori et al., 2024; Qwen Team, 2024; Jiang et al., 2023). Following Lots-of-LoRAs (Gabrielsson et al., 2025), we select 50 tasks from Super-NaturalInstructions (Wang et al., 2022a) and train one LoRA adapter per task to form the stream.Further details are provided in Appendices B.1 and B.2.

Streaming protocol and storage budget. At each time step, a merging method can access only the incoming LoRA adapter and the merged state from the previous time step; neither task training data nor historical raw LoRA adapters can be revisited. All methods are subject to the same persisted-parameter budget b. For a given base model and input LoRA rank r, we set b to the total parameter count of K full LoRA adapters and use K to denote the corresponding budget level.

Baselines. We compare streaming implementations of Linear, TIES, DARE, DARE-TIES, OPCM, K-Merge, K-Merge++, and Share (Yadav et al., 2023; Yu et al., 2024; Tang et al., 2025a; Shenaj et al., 2026; Kaushik et al., 2026). We also report CtM (Compress-then-Merge; He et al., 2026) and LoRA-LEGO (Zhao et al., 2025) as offline performance references. Implementation details are provided in Appendix B.3.

Evaluation metrics. We use ROUGE-L (Lin, 2004) for generative language tasks and accuracy for language tasks formulated as label classification. To align scales across tasks, we normalize merged performance by that of the corresponding single-task LoRA adapter and compute the aggregate score $s^{(t)}$ according to Equation 1. We measure streaming update efficiency by per-step update time and peak memory usage. Further details are provided in Appendices B.4 and B.5.

4.2 Performance under Fixed Storage Budgets

We evaluate performance under fixed storage budgets on a stream of 50 LoRA adapters with rank $r = 16$. We first compare the aggregate score $s^{(50)}$ of all methods on Llama, Qwen, and Mistral under budget $K = 3$. We then vary the budget over $K \in \{1, \ldots, 8\}$ on Llama and Qwen to examine performance across storage budgets.

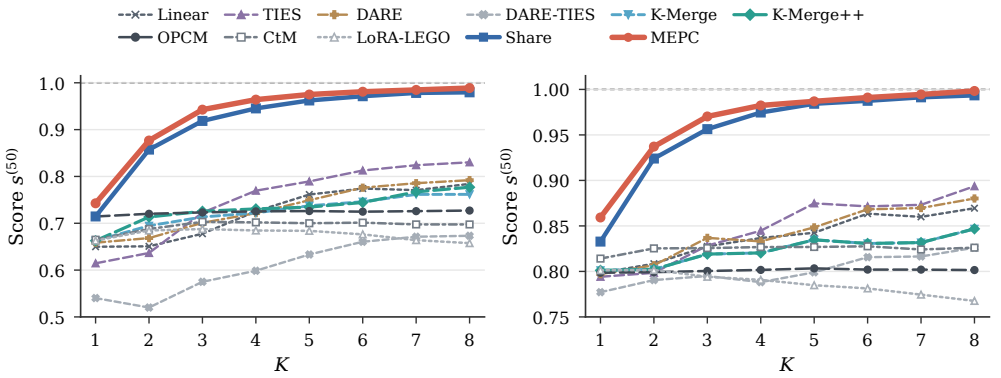
Performance across models. Table 1 reports the final aggregate scores on the three language models. MEPC obtains the highest score on all of them, with gains of up to 2.60 percentage points over Share, the strongest baseline. It also surpasses the offline references, even though those methods retain access to all per-task LoRA adapters simultaneously. The improvement holds across model families and parameter scales, indicating that MEPC preserves per-task adaptation more effectively under a fixed budget. ROUGE-L results and task-level retention distributions are reported in Appendix C.1 and C.2, respectively.

Performance across budgets. Figure 1 shows the final aggregate score across storage budgets. MEPC achieves the highest score at every budget level on both Llama and Qwen and progressively

approaches the reference performance of the single-task LoRA adapters as K increases. Under the most restrictive setting, $K = 1$, MEPC outperforms Share by 2.81 and 2.65 percentage points, respectively. The advantage is more pronounced under smaller budgets, indicating that when the capacity of the shared representation is tightly constrained, MEPC uses importance-aware compression based on joint deletion loss to preferentially preserve more important adaptation directions. Beyond language models, MEPC also achieves the highest final aggregate score on a stream of 19 VTAB-1k tasks using DINOv2 ViT-L/14 under all three tested storage budgets (Appendix C.3).

Table 1: Aggregate score $s^{(50)}$ with input rank $r = 16$ and storage budget $K = 3$.

Model	Linear	TIES	DARE	DARE-TIES	K-Merge	K-Merge++	OPCM	CtM	LoRA-LEGO	Share	MEPC
Llama-3.2-1B	0.6777	0.7226	0.7009	0.5750	0.7134	0.7256	0.7232	0.7030	0.6876	0.9186	**0.9446**
Qwen2.5-1.5B	0.8275	0.8284	0.8369	0.7949	0.8190	0.8203	0.8005	0.8256	0.7941	0.9564	**0.9703**
Mistral-7B	0.8091	0.8205	0.7947	0.7478	0.8253	0.8237	0.7948	0.8141	0.7664	0.9676	**0.9763**

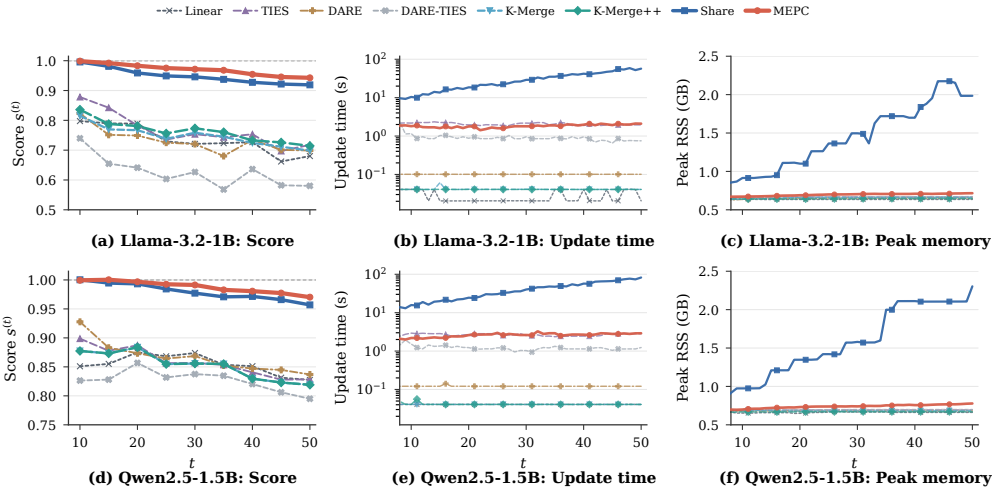


(a) Llama-3.2-1B **(b) Qwen2.5-1.5B**

Figure 1: Aggregate score $s^{(50)}$ under different storage budgets. The horizontal dashed lines indicate the reference performance of single-task LoRA adapters.

4.3 Streaming Scalability and Efficiency

To examine performance and update overhead as the task stream grows, we use a stream of 50 LoRA adapters with rank $r = 16$ on Llama and Qwen, with a storage budget of $K = 3$. We track the aggregate score $s^{(t)}$, per-step update time, and peak memory usage throughout the task stream.



(a) Llama-3.2-1B: Score **(b) Llama-3.2-1B: Update time** **(c) Llama-3.2-1B: Peak memory**

(d) Qwen2.5-1.5B: Score **(e) Qwen2.5-1.5B: Update time** **(f) Qwen2.5-1.5B: Peak memory**

Figure 2: Aggregate score and update overhead over the task stream on Llama-3.2-1B-Instruct (top) and Qwen2.5-1.5B-Instruct (bottom). From left to right: aggregate score $s^{(t)}$, per-step update time, and peak memory usage.

Performance throughout the task stream. As shown in Figure 2(a) and (d), MEPC maintains the highest aggregate score at almost every time step, with its margin over Share generally widening

as more tasks arrive. From 10 to 50 tasks, MEPC's aggregate score drops by 5.60 and 2.91 percentage points on Llama and Qwen, compared with 7.66 and 4.33 for Share, indicating that it more effectively limits the loss of previously acquired task information under a fixed storage budget.

Update efficiency. Panels (b), (c), (e), and (f) of Figure 2 show per-step update time and peak memory usage as the number of tasks increases. For Share, both measures grow steadily with the task stream, whereas MEPC maintains relatively stable update overhead throughout. By avoiding the reconstruction of historical LoRA factors through incremental subspace updates, MEPC achieves up to a $28.34\times$ speedup in per-step updates over Share and reduces peak memory usage by 66.2%.

4.4 Effective Task Capacity under a Fixed Budget

We compare the effective task capacity of all methods on Llama and Qwen under a fixed storage budget and different performance thresholds θ. We use a stream of 50 LoRA adapters with $r = 16$, set the storage budget to $K = 1$, and consider $\theta \in \{0.9, 0.75, 0.5\}$.

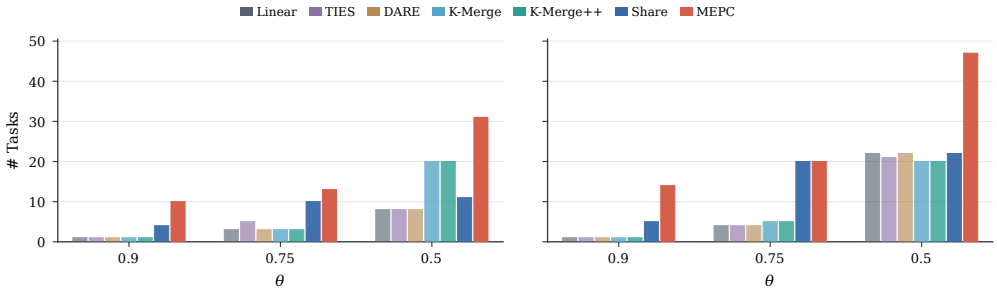
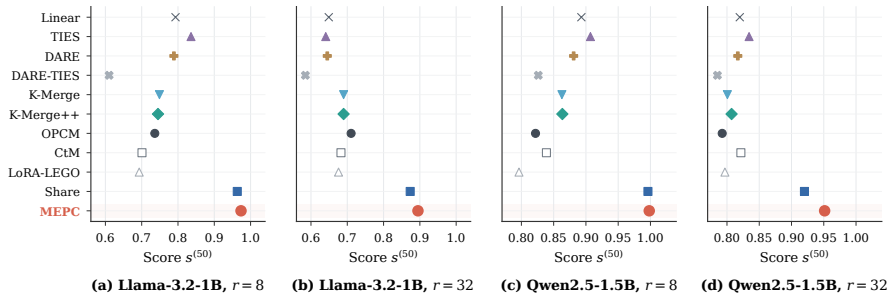


Figure 3: Effective task capacity of each method at different performance thresholds θ.

Effective task capacity. Figure 3 compares the effective task capacity of each method across performance thresholds. MEPC achieves the highest effective task capacity at every threshold on both base models. Under the limited budget, MEPC more effectively balances information retention across tasks and reduces task-related information loss during streaming LoRA merging, resulting in a higher effective task capacity.

4.5 Robustness to Input Rank and Task Order

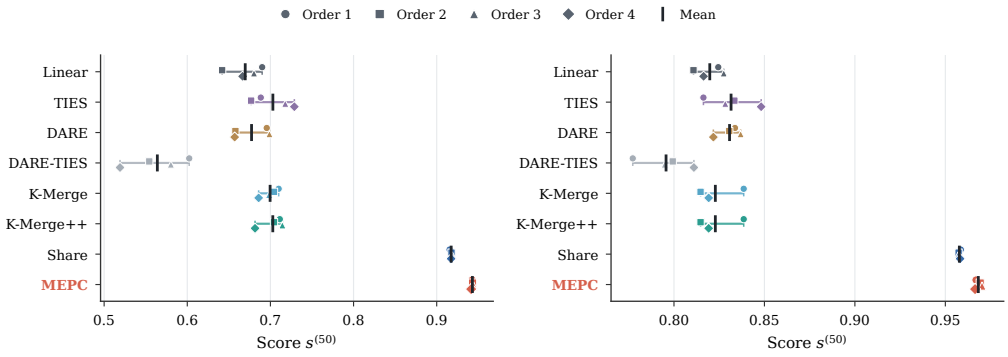
We further evaluate MEPC's robustness to input LoRA rank and task arrival order. For both experiments, we fix the absolute budget to that of four rank-16 LoRA adapters, varying $r \in \{8, 32\}$ for input-rank robustness and testing four task permutations at $r = 16$ for task-order robustness. Details of the task-order construction are provided in Appendix B.6. We also study mixed-rank LoRA streams, with results provided in Appendix C.4.



(a) Llama-3.2-1B, $r = 8$ **(b) Llama-3.2-1B, $r = 32$** **(c) Qwen2.5-1.5B, $r = 8$** **(d) Qwen2.5-1.5B, $r = 32$**

Figure 4: Aggregate score $s^{(50)}$ for different LoRA ranks under the same absolute storage budget.

Input LoRA rank. As shown in Figure 4, MEPC performs best in all settings. As r increases from 8 to 32, its margin over Share grows from 1.03 to 2.14 percentage points on Llama and from 0.20 to 3.13 percentage points on Qwen. With the absolute budget fixed, increasing the input rank increases the number of adapter parameters to represent, thereby increasing compression pressure. The wider margins at $r = 32$ suggest greater robustness to increased compression pressure.



(a) Llama-3.2-1B, $K = 3$ **(b) Qwen2.5-1.5B,** $K = 3$

Figure 5: Aggregate score $s^{(50)}$ under different task arrival orders.

Task arrival order. Figure 5 shows the final aggregate score under different task arrival orders with storage budget $K = 3$. MEPC achieves the highest score under all four task orders on both models. For each model, the maximum range of MEPC across the four orders is only 0.38 percentage points, whereas the maximum range of the comparison methods reaches 8.32 percentage points. The smaller variation across orders indicates that MEPC is robust to task arrival order and that its advantage under a fixed storage budget does not depend on a particular task permutation.

4.6 Ablation Study

We compare MEPC with four variants that combine uniform or importance-aware compression with one-sided or joint deletion loss. The uniform variants remove the same number of directions from each shared subspace and use the corresponding loss to select the directions to remove. The importance-aware variants instead evaluate all directions with the corresponding deletion loss and progressively remove the direction with the smallest loss.

Table 2: Ablation results with input rank $r = 16$ and storage budget $K = 3$.

Variant	Llama-3.2-1B	Qwen2.5-1.5B	Mistral-7B
Uniform + One-sided	0.9212	0.9592	0.9670
Uniform + Joint	0.9198	0.9570	0.9673
Importance-aware + One-sided	0.8669	0.9365	0.9508
Importance-aware + Joint	**0.9457**	0.9682	0.9761
MEPC	0.9446	**0.9703**	**0.9763**

Under uniform compression, the two deletion losses perform similarly. With importance-aware compression, however, replacing the joint loss with the one-sided loss substantially degrades performance. This result shows that evaluating each candidate by its effect on the reconstructed weight update provides a more reliable basis for deletion decisions across shared bases. When using the joint loss, importance-aware compression further outperforms uniform compression, indicating that it preserves more task information under the same storage budget. Appendix C.5 further visualizes and analyzes the importance-aware compression.

Evaluating all directions with the joint loss yields the optimal choice for each individual deletion, but incurs substantially higher computational cost. Since an update typically requires multiple deletions, individually optimal choices do not guarantee a globally optimal deletion set. MEPC instead adopts two-stage greedy screening, achieving comparable or better performance at a lower computational cost. Detailed efficiency comparisons are provided in Appendices C.6 .

5 Conclusion

We introduced MEPC, a streaming LoRA merging method for on-device continual adaptation under a fixed storage budget. MEPC incrementally expands shared subspaces without reconstructing historical LoRA factors and uses joint deletion loss to guide importance-aware compression. Experiments across language tasks and model settings demonstrate improved performance retention and effective task capacity, together with reduced update time and peak memory usage.

AI USE STATEMENT

We used large language models to polish the writing, including grammar correction, style harmonization, and sentence-level revisions. All suggested edits were reviewed by the authors, who take full responsibility for the final content.

REPRODUCIBILITY STATEMENT

The main paper describes the datasets, base models, and evaluation metrics used in our experiments. All datasets are publicly available. The appendix provides the complete experimental setup and implementation details. We will release the code upon acceptance to facilitate reproduction of our results.

REFERENCES

Jihwan Bang, Juntae Lee, Kyuhong Shim, Seunghan Yang, and Simyung Chang. Crayon: Customized on-device LLM via instant adapter blending and edge-server hybrid inference. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3720–3731. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.204. URL https://aclanthology.org/2024.acl-long.204/.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: A strong, simple baseline. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/b704ea2c39778f07c617f6b7ce480e9e-Abstract.html.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022. doi: 10.1109/TPAMI.2021.3057446.

Sauptik Dhar, Junyao Guo, Jiayi Liu, Samarth Tripathi, Unmesh Kurup, and Mohak Shah. A survey of on-device machine learning: An algorithms and learning theory perspective. *ACM Transactions on Internet of Things*, 2(3):1–49, 2021. doi: 10.1145/3450494. URL https://doi.org/10.1145/3450494.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5: 220–235, 2023. doi: 10.1038/s42256-023-00626-4. URL https://doi.org/10.1038/s42256-023-00626-4.

Sebastian Dziadzio, Vishaal Udandarao, Karsten Roth, Ameya Prabhu, Zeynep Akata, Samuel Albanie, and Matthias Bethge. How to merge your multimodal models over time? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20479–20491, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Dziadzio_How_to_Merge_Your_Multimodal_Models_Over_Time_CVPR_2025_paper.html.

Rickard Brüel Gabrielsson, Jiacheng Zhu, Onkar Bhardwaj, Leshem Choshen, Kristjan Greenewald, Mikhail Yurochkin, and Justin Solomon. Compress then serve: Serving thousands of LoRA adapters with little overhead. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 18062–18095. PMLR, 2025. URL https://proceedings.mlr.press/v267/gabrielsson25a.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, et al. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024. URL https://arxiv.org/abs/2407.21075.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=lIsCS8b6zj.

Zhengbao He, Ruiqi Ding, Zhehao Huang, Ruikai Yang, Tao Li, and Xiaolin Huang. Compress then merge: From multiple LoRAs into one low-rank adapter. *arXiv preprint arXiv:2606.03723*, 2026. URL https://arxiv.org/abs/2606.03723.

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.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient cross-task generalization via dynamic LoRA composition. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=TrloAXEJ2B.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

Wooseong Jeong, Wonyoung Lee, and Kuk-Jin Yoon. Preference-aligned LoRA merging: Preserving subspace coverage and addressing directional anisotropy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 823–835, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Jeong_Preference-Aligned_LoRA_Merging_Preserving_Subspace_Coverage_and_Addressing_Directional_Anisotropy_CVPR_2026_paper.html.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

Prakhar Kaushik, Ankit Vaidya, Shravan Chaudhari, and Alan Yuille. EigenLoRAx: Recycling adapters to find principal subspaces for resource-efficient adaptation and inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 649–659, 2025. doi: 10.1109/CVPRW67362.2025.00070. URL https://openaccess.thecvf.com/content/CVPR2025W/TCV/html/Kaushik_EigenLoRAx_Recycling_Adapters_to_Find_Principal_Subspaces_for_Resource-Efficient_Adaptation_CVPRW_2025_paper.html.

Prakhar Kaushik, Ankit Vaidya, Shravan Sunil Chaudhari, Rama Chellappa, and Alan Yuille. Shared LoRA subspaces for almost strict continual learning. In *European Conference on Computer Vision*, 2026. URL https://eccv.ecva.net/virtual/2026/poster/3502.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based random matrix adaptation. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

Borui Li, Yitao Wang, Haoran Ma, Ligeng Chen, Jun Xiao, and Shuai Wang. MobiLoRA: Accelerating LoRA-based LLM inference on mobile devices via context-aware KV cache optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23400–23410. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1140. URL https://aclanthology.org/2025.acl-long.1140/.

Yan-Shuo Liang and Wu-Jun Li. InfLoRA: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23638–23647, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Liang_InfLoRA_Interference-Free_Low-Rank_Adaptation_for_Continual_Learning_CVPR_2024_paper.html.

Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013/.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaa El-Nouby, et al. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=a68SUt6zFt.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. URL https://arxiv.org/abs/2412.15115.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Rebuffi_iCaRL_Incremental_Classifier_CVPR_2017_paper.html.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-LoRA: Enhancing parameter efficiency of LoRA with weight tying. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8694–8705. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.481. URL https://aclanthology.org/2024.naacl-long.481/.

Donald Shenaj, Ondrej Bohdal, Taha Ceritli, Mete Ozay, Pietro Zanuttigh, and Umberto Michieli. K-Merge: Online continual merging of adapters for on-device large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3013–3029. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.137. URL https://aclanthology.org/2026.acl-long.137/.

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*, pp. 11909–11919, 2023. URL https://openaccess.thecvf.com/content/CVPR2023/html/Smith_CODA-Prompt_COntinual_Decomposed_Attention-Based_Prompting_for_Rehearsal-Free_Continual_Learning_CVPR_2023_paper.html.

Anke Tang, Enneng Yang, Li Shen, Yong Luo, Han Hu, Lefei Zhang, Bo Du, and Dacheng Tao. Merging on the fly without retraining: A sequential approach to scalable continual model merging. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. doi: 10.52202/085713-2747. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/7611a3cb5d733e628081431445cb01fd-Abstract-Conference.html.

Dennis Tang, Prateek Yadav, Yi-Lin Sung, Jaehong Yoon, and Mohit Bansal. LoRA merging with SVD: Understanding interference and preserving performance. In *ICML 2025 R2-FM Workshop*, 2025b. URL https://openreview.net/forum?id=t9FrMviTaP.

Yixuan Tang and Yi Yang. Crowded in B-space: Calibrating shared directions for LoRA merging. *arXiv preprint arXiv:2604.16826*, 2026. URL https://arxiv.org/abs/2604.16826.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. HydraLoRA: An asymmetric LoRA architecture for efficient fine-tuning. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0304. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/123fd8a56501194823c8e0dca00733df-Abstract-Conference.html.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 5085–5109. Association for Computational Linguistics, 2022a. doi: 10.18653/v1/2022.emnlp-main.340. URL https://aclanthology.org/2022.emnlp-main.340/.

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*, pp. 139–149, 2022b. URL https://openaccess.thecvf.com/content/CVPR2022/html/Wang_Learning_To_Prompt_for_Continual_Learning_CVPR_2022_paper.html.

Zhengxuan Wei, Yi Dong, Zonghui Li, Xianhui Lin, Xing Liu, Hong Gu, Shaofeng Zhang, Wenbin Li, and Qi Fan. SSR-Merge: Subspace signal routing for training-free LoRA merging in diffusion models. In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/62664.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 23965–23998. PMLR, 2022. URL https://proceedings.mlr.press/v162/wortsman22a.html.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin A. Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0310. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1644c9af28ab7916874f6fd6228a9bcf-Abstract-Conference.html.

Enneng Yang, Zhenyi Wang, Li Shen, Shiwei Liu, Guibing Guo, Xingwei Wang, and Dacheng Tao. AdaMerging: Adaptive model merging for multi-task learning. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=nZP6NgD3QY.

Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 57755–57775. PMLR, 2024. URL https://proceedings.mlr.press/v235/yu24p.html.

Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019. URL https://arxiv.org/abs/1910.04867.

Haobo Zhang and Jiayu Zhou. Unraveling LoRA interference: Orthogonal subspaces for robust model merging. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 26459–26472, 2025. doi: 10.18653/v1/2025.acl-long.1284. URL https://aclanthology.org/2025.acl-long.1284/.

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*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

Ziyu Zhao, Tao Shen, Didi Zhu, Zexi Li, Jing Su, Xuwu Wang, Kun Kuang, and Fei Wu. Merging LoRAs like playing LEGO: Pushing the modularity of LoRA to extremes through rank-wise clustering. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/b54e0146a82945f01e69c2e3309ba925-Abstract-Conference.html.

Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Haitz Sáez De Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. Asymmetry in low-rank adapters of foundation models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62369–62385. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhu24c.html.

Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang. Delta-LoRA: Fine-tuning high-rank parameters with the delta of low-rank matrices. *arXiv preprint arXiv:2309.02411*, 2023. URL https://arxiv.org/abs/2309.02411.

A MEPC UPDATE

Algorithm 1 summarizes one complete MEPC update, including initialization, incremental subspace update, and importance-aware compression. Here, ThinQR denotes the thin QR factorization, and $\mathrm{Eigh}_{\downarrow}$ returns the eigenpairs of a symmetric matrix in descending order of eigenvalue. The previous representation is empty when $t = 0$, $\mathrm{cols}(\cdot)$ returns the number of matrix columns, and the slice notation selects the indicated rows or columns.

Algorithm 1: Complete MEPC Update at Time Step $t+1$

Input: $\{\mathbf{U}_{m,f}^{(t)}, \{\mathbf{C}_{i,m,f}^{(t)}\}_{i=1}^{t}\}_{m\in\mathcal{M},f\in\mathcal{F}}$; $\mathcal{L}_{t+1}$; budget b

Output: $\{\mathbf{U}_{m,f}^{(t+1)}, \{\mathbf{C}_{i,m,f}^{(t+1)}\}_{i=1}^{t+1}\}_{m\in\mathcal{M},f\in\mathcal{F}}$

if $t = 0$ **then** ▷ Initialization
  for all $m \in \mathcal{M}$ **do**
    $(\widetilde{\mathbf{U}}_{m,\mathrm{A}}, \widetilde{\mathbf{C}}_{1,m,\mathrm{A}}) \leftarrow \mathrm{ThinQR}(\mathbf{A}_{1,m}^{\top})$
    $(\widetilde{\mathbf{U}}_{m,\mathrm{B}}, \widetilde{\mathbf{C}}_{1,m,\mathrm{B}}) \leftarrow \mathrm{ThinQR}(\mathbf{B}_{1,m})$
  end for
else ▷ Incremental subspace update
  for all $m \in \mathcal{M}$ **do**
    $\mathbf{X}_{m,\mathrm{A}} \leftarrow \mathbf{A}_{t+1,m}^{\top}$; $\mathbf{X}_{m,\mathrm{B}} \leftarrow \mathbf{B}_{t+1,m}$
    for all $f \in \mathcal{F}$ **do**
      $\mathbf{E}_{m,f} \leftarrow (\mathbf{I} - \mathbf{U}_{m,f}^{(t)}(\mathbf{U}_{m,f}^{(t)})^{\top})\mathbf{X}_{m,f}$
      if $\mathbf{E}_{m,f} \neq \mathbf{0}$ **then**
        $(\mathbf{Q}_{m,f}, \mathbf{R}_{m,f}) \leftarrow \mathrm{ThinQR}(\mathbf{E}_{m,f})$
        $\widetilde{\mathbf{U}}_{m,f} \leftarrow [\mathbf{U}_{m,f}^{(t)}, \mathbf{Q}_{m,f}]$
        $\widetilde{\mathbf{C}}_{i,m,f} \leftarrow [(\mathbf{C}_{i,m,f}^{(t)})^{\top}, \mathbf{0}^{\top}]^{\top}$ for $i = 1, \ldots, t$
        $\widetilde{\mathbf{C}}_{t+1,m,f} \leftarrow [((\mathbf{U}_{m,f}^{(t)})^{\top}\mathbf{X}_{m,f})^{\top}, \mathbf{R}_{m,f}^{\top}]^{\top}$
      else
        $\widetilde{\mathbf{U}}_{m,f} \leftarrow \mathbf{U}_{m,f}^{(t)}$
        $\widetilde{\mathbf{C}}_{i,m,f} \leftarrow \mathbf{C}_{i,m,f}^{(t)}$ for $i = 1, \ldots, t$
        $\widetilde{\mathbf{C}}_{t+1,m,f} \leftarrow (\mathbf{U}_{m,f}^{(t)})^{\top}\mathbf{X}_{m,f}$
      end if
    end for
  end for
end if
$q_{t+1} \leftarrow \sum_{i=1}^{t+1} r_i$; $k_{m,f} \leftarrow \mathrm{cols}(\widetilde{\mathbf{U}}_{m,f})$ for all m, f

$b_{\text{cur}} \leftarrow \sum_{m\in\mathcal{M}}[k_{m,\text{A}}(d_m + q_{t+1}) + k_{m,\text{B}}(n_m + q_{t+1})]$
if $b_{\text{cur}} > b$ **then** ▷ Importance-aware compression
  for all $m \in \mathcal{M},\ f \in \mathcal{F}$ **do**
    $\mathbf{G}_{m,f} \leftarrow \sum_{i=1}^{t+1} \widetilde{\mathbf{C}}_{i,m,f}\widetilde{\mathbf{C}}_{i,m,f}^{\top}$
    $(\mathbf{V}_{m,f}, \mathbf{\Lambda}_{m,f}) \leftarrow \text{Eigh}_{\downarrow}(\mathbf{G}_{m,f})$
    $\overline{\mathbf{U}}_{m,f} \leftarrow \widetilde{\mathbf{U}}_{m,f}\mathbf{V}_{m,f}$
    $\overline{\mathbf{C}}_{i,m,f} \leftarrow \mathbf{V}_{m,f}^{\top}\widetilde{\mathbf{C}}_{i,m,f}$ for $i = 1, \ldots, t+1$
  end for
  for all $m \in \mathcal{M}$ **do**
    if $k_{m,\text{A}} > 0$ **then**
      $\ell_{m,\text{A},k_{m,\text{A}}} \leftarrow \sum_{i=1}^{t+1} \|\overline{\mathbf{C}}_{i,m,\text{B}}(\overline{\mathbf{C}}_{i,m,\text{A}}[k_{m,\text{A}},:])^{\top}\|_2^2$
    end if
    if $k_{m,\text{B}} > 0$ **then**
      $\ell_{m,\text{B},k_{m,\text{B}}} \leftarrow \sum_{i=1}^{t+1} \|\overline{\mathbf{C}}_{i,m,\text{A}}(\overline{\mathbf{C}}_{i,m,\text{B}}[k_{m,\text{B}},:])^{\top}\|_2^2$
    end if
  end for
  while $b_{\text{cur}} > b$ **do**
    $(m^\star, f^\star) \leftarrow \arg\min_{m\in\mathcal{M},\, f\in\mathcal{F}:k_{m,f}>0} \ell_{m,f,k_{m,f}}$
    $k_{m^\star,f^\star} \leftarrow k_{m^\star,f^\star} - 1$
    $\overline{\mathbf{U}}_{m^\star,f^\star} \leftarrow \overline{\mathbf{U}}_{m^\star,f^\star}[:, 1:k_{m^\star,f^\star}]$
    $\overline{\mathbf{C}}_{i,m^\star,f^\star} \leftarrow \overline{\mathbf{C}}_{i,m^\star,f^\star}[1:k_{m^\star,f^\star},:]$ for $i = 1, \ldots, t+1$
    if $f^\star = \text{A}$ **then**
      $b_{\text{cur}} \leftarrow b_{\text{cur}} - (d_{m^\star} + q_{t+1})$
    else
      $b_{\text{cur}} \leftarrow b_{\text{cur}} - (n_{m^\star} + q_{t+1})$
    end if
    if $k_{m^\star,\text{A}} > 0$ **then**
      $\ell_{m^\star,\text{A},k_{m^\star,\text{A}}} \leftarrow \sum_{i=1}^{t+1} \|\overline{\mathbf{C}}_{i,m^\star,\text{B}}(\overline{\mathbf{C}}_{i,m^\star,\text{A}}[k_{m^\star,\text{A}},:])^{\top}\|_2^2$
    end if
    if $k_{m^\star,\text{B}} > 0$ **then**
      $\ell_{m^\star,\text{B},k_{m^\star,\text{B}}} \leftarrow \sum_{i=1}^{t+1} \|\overline{\mathbf{C}}_{i,m^\star,\text{A}}(\overline{\mathbf{C}}_{i,m^\star,\text{B}}[k_{m^\star,\text{B}},:])^{\top}\|_2^2$
    end if
  end while
  $\mathbf{U}_{m,f}^{(t+1)} \leftarrow \overline{\mathbf{U}}_{m,f}$; $\mathbf{C}_{i,m,f}^{(t+1)} \leftarrow \overline{\mathbf{C}}_{i,m,f}$ for all i, m, f
else
  $\mathbf{U}_{m,f}^{(t+1)} \leftarrow \widetilde{\mathbf{U}}_{m,f}$; $\mathbf{C}_{i,m,f}^{(t+1)} \leftarrow \widetilde{\mathbf{C}}_{i,m,f}$ for all i, m, f
end if

B Experimental Details

B.1 Datasets

Language tasks. Following the experimental setup of Lots-of-LoRAs (Gabrielsson et al., 2025), we use 50 tasks from Super-NaturalInstructions (Wang et al., 2022a), spanning eight categories: classification; natural language inference, judgment, and causal reasoning; multiple-choice answer generation; open-domain and reading comprehension; summarization and title generation; structured data-to-text generation; entity and span extraction; and linguistic tagging. We use the public training and test splits without additional sampling, and all three language models use identical data files. Table 3 reports the training and test set sizes for each task.

Table 3: Statistics of the language task datasets.

Task	# Train	# Test
Classification		
`task280_stereoset_classification_stereotype_type`	2,966	371
`task629_dbpedia_14_classification`	2,400	300
`task875_emotion_classification`	2,583	323
`task1283_hrngo_quality_classification`	3,180	398

Task	# Train	# Test
task566_circa_classification	4,380	548
task1384_deal_or_no_dialog_classification	3,367	421
task1534_daily_dialog_question_classification	4,928	617
task616_cola_classification	5,200	650
NLI, Judgment, and Causal Reasoning		
task190_snli_classification	5,200	650
task199_mnli_classification	5,199	650
task1529_scitail1.1_classification	4,048	506
task069_abductivenli_classification	5,200	650
task066_timetravel_binary_consistency_classification	5,200	650
task391_causal_relationship	1,826	229
task290_tellmewhy_question_answerability	2,785	349
Multiple-choice Answer Generation		
task073_commonsenseqa_answer_generation	975	122
task1399_obqa_answer_generation	1,414	177
task229_arc_answer_generation_hard	2,071	259
task228_arc_answer_generation_easy	4,154	520
task1391_winogrande_easy_answer_generation	5,192	650
task453_swag_answer_generation	5,200	650
task580_socialiqa_answer_generation	5,200	650
task080_piqa_answer_generation	5,200	650
task591_sciq_answer_generation	5,200	650
Open-domain and Reading QA		
task1608_xquad_en_answer_generation	948	119
task460_qasper_answer_generation	2,137	268
task247_dream_answer_generation	4,890	612
task344_hybridqa_answer_generation	5,196	650
task075_squad1.1_answer_generation	5,198	650
task582_naturalquestion_answer_generation	5,200	650
task898_freebase_qa_answer_generation	5,200	650
Summarization and Title Generation		
task769_qed_summarization	800	100
task619_ohsumed_abstract_title_generation	1,007	126
task288_gigaword_summarization	1,546	194
task645_summarization	1,600	200
task1358_xlsum_title_generation	4,901	613
task1342_amazon_us_reviews_title	4,973	622
Structured Data-to-text Generation		
task957_e2e_nlg_text_generation_generate	1,728	217
task1409_dart_text_generation	5,185	649
task1598_nyc_long_text_generation	5,195	650
task1728_web_nlg_data_to_text	5,196	650
task210_logic2text_structured_text_generation	5,198	650
task110_logic2text_sentence_generation	5,198	650
Entity and Span Extraction		
task1545_conll2002_person_name_extraction_answer_generation	3,724	466
task1546_conll2002_location_name_extraction_answer_generation	3,724	466
task039_qasc_find_overlapping_words	5,200	650
Linguistic Tagging		
task620_ohsumed_medical_subject_headings_answer_generation	1,008	127
task584_udeps_eng_fine_pos_tagging	4,783	598
task1168_brown_coarse_pos_tagging	5,179	648
task1167_penn_treebank_coarse_pos_tagging	5,184	648

Vision tasks. We use all 19 image classification tasks from VTAB-1k (Zhai et al., 2019), comprising seven natural, four specialized, and eight structured tasks. For each task, the source LoRA is trained on the standard 800 training examples and evaluated on the full test set. Table 4 reports the number of classes and split sizes for each task.

Table 4: Statistics of the VTAB-1k vision task datasets.

Task	# Classes	# Train	# Test
Natural			
caltech101	102	800	6,084
cifar	100	800	10,000
dtd	47	800	1,880
oxford_flowers102	102	800	6,149
oxford_iiit_pet	37	800	3,669
svhn	10	800	26,032
sun397	397	800	21,750
Specialized			
patch_camelyon	2	800	32,768
eurosat	10	800	5,400
resisc45	45	800	6,300
diabetic_retinopathy	5	800	42,670
Structured			
clevr_count	8	800	15,000
clevr_dist	6	800	15,000
dmlab	6	800	22,735
kitti	4	800	711
dsprites_loc	16	800	73,728
dsprites_ori	16	800	73,728
smallnorb_azi	18	800	12,150
smallnorb_ele	9	800	12,150

B.2 Source LoRA Training

Language models. We independently train a task-specific source LoRA for each language task and reuse the same source LoRAs across all merging methods. Table 5 summarizes the training configurations for different input ranks.

Table 5: Training configurations for source LoRAs on language tasks.

Hyperparameter	$r = 8$	$r = 16$	$r = 32$
α	16	32	64
Target modules	q_proj, k_proj, v_proj, o_proj		
Dropout	0.05	0.05	0.05
Optimizer	AdamW	AdamW	AdamW
Learning rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
Weight decay	0.01	0.01	0.01
Training epochs	2	2	2
LR schedule	Linear warmup + cosine decay		
Warmup ratio	0.03	0.03	0.03
Effective batch size	64	64	64
Precision	bfloat16	bfloat16	bfloat16
Random seed	42	42	42

Vision models. For each VTAB-1k task, we independently train a DINOv2 ViT-L/14 source LoRA and a task-specific linear classification head. All merging methods use the same source LoRAs and reuse the corresponding task-specific classification heads for evaluation. The classification heads are neither merged with the LoRAs nor counted toward the LoRA storage budget. Table 6 reports the training configuration.

B.3 Baseline Implementations

Common setup. All baselines use the same source LoRAs, task orders, and storage budgets. The persisted state of a streaming method cannot exceed the parameter count of K source LoRAs at any time step. Offline methods are not subject to this per-step constraint and are included only to compare final task performance and batch processing overhead.

Linear. We implement Linear as a simple streaming slot baseline. An incoming LoRA is stored directly when an empty slot is available. Once all K slots are occupied, the incoming LoRA is assigned to the slot with the fewest previous merges, and its A and B factors are averaged with the existing factors using equal weights of 0.5.

TIES and DARE variants. TIES and DARE were originally designed to merge task vectors from full models (Yadav et al., 2023; Yu et al., 2024). To adapt them to streaming LoRA merging, we

Table 6: Training configuration for source LoRAs on VTAB-1k vision tasks.

Hyperparameter	Value
r	16
α	16
Target modules	`query`, `key`, `value`, `dense`
LoRA dropout	0.0
Optimizer	AdamW
Learning rate	3×10^{-4}
Weight decay	0.01
Training epochs	50
LR schedule	CosineAnnealingLR
Loss	CrossEntropyLoss
Training batch size	16
Precision	bfloat16
Random seed	42

use the slot assignment strategy of Linear and independently apply each merging operator to the A and B factors. TIES, DARE, and DARE-TIES use unit weights for both inputs and a density of 0.5. Each update accesses only the current state of the target slot and the incoming LoRA, without retaining or revisiting historical source LoRAs.

Continual baselines. K-Merge and OPCM follow their original implementations (Shenaj et al., 2026; Tang et al., 2025a). For K-Merge++, we set the similarity threshold to 0.02. Share (Kaushik et al., 2026) uses the adapter-only, data-free setting: each update receives only the LoRA factors and uses neither task data nor additional training. To satisfy the storage constraint, we retain the largest basis rank allowed by the budget after each update and count both the shared bases and task-specific coefficients as persisted parameters.

Offline references. Compress-then-Merge (He et al., 2026) and LoRA-LEGO (Zhao et al., 2025) follow their original implementations. Both methods access all source LoRA adapters simultaneously and produce a single merged adapter that is applied to every task without task-dependent reconstruction. We set their output rank to Kr so that the number of persisted adapter parameters matches the corresponding storage budget. In contrast, MEPC stores shared bases and task-specific coefficients and, given the task identity, reconstructs a rank-r adapter for the queried task. Consequently, the offline references perform inference using a shared rank-Kr adapter, whereas MEPC uses a task-specific rank-r adapter. CtM uses $\beta = 0.25$, a 25% top-k ratio, and a scaling coefficient of 0.75. LoRA-LEGO uses k-means++ initialization, 10 restarts, at most 300 iterations, and a convergence threshold of 10^{-4}.

B.4 Efficiency Measurement

We measure streaming efficiency using per-step update time and resident set size (RSS). Per-step update time is the wall-clock time of a single merging step. We use RSS to measure the physical memory held by the merging process and obtain it directly from `/proc/<pid>/status`. During each subspace update, a dedicated monitoring thread samples RSS at 20 ms intervals, and we report the maximum sampled value as the peak RSS of that step. Both metrics are recorded at every step and plotted as raw per-step values.

B.5 Hardware and Software

All experiments are run on a single MLU590-M9DK. The compute node has two Intel Xeon Platinum 8457C processors and 2 TB of host memory, and runs Debian GNU/Linux 12, PyTorch 2.7.0a0, and `torch_mlu` 1.27.121.

B.6 Task Order Construction

Language tasks. The language task stream consists of the 50 tasks in Table 3. The category order follows the eight task categories sequentially while preserving the original order within each cate-

gory. Each random order is generated by applying `random.Random(seed).shuffle` to the full task list. We use seed 42 for the default experiments. The task-order robustness experiments compare the category order with three random orders generated using seeds 1, 42, and 7. For a given order, all methods, base models, and storage budgets use the same task stream and source LoRAs.

Vision tasks. We apply the same random permutation procedure to the 19 VTAB-1k tasks in Table 4 and fix the seed to 42. All vision methods and storage budgets share this task order.

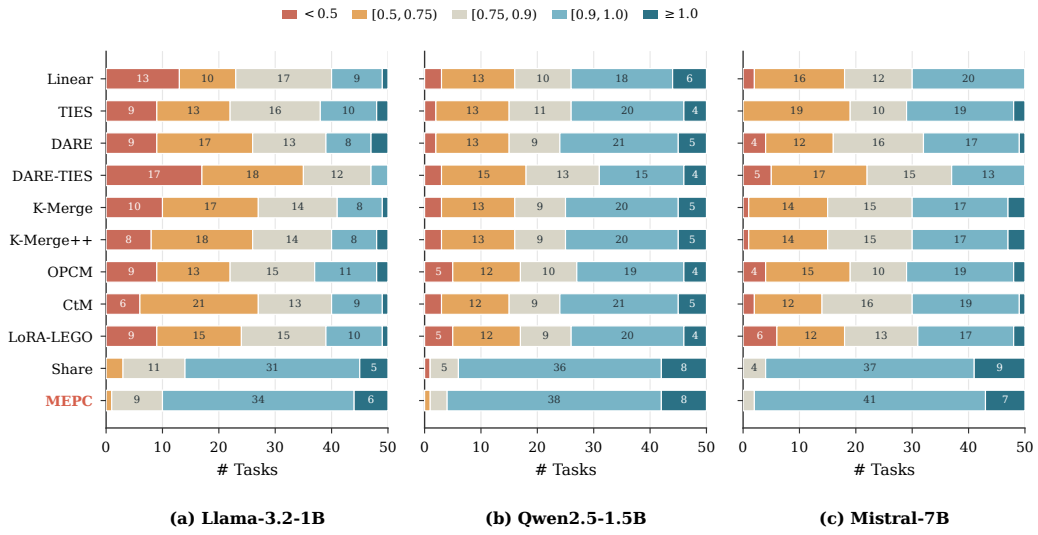
C Additional Analyses

C.1 ROUGE-L Results

Table 7 reports the results in Table 1 using ROUGE-L rather than task-wise retention ratios. Each score is the unweighted mean over the 50 tasks. MEPC achieves the highest score among the merging methods on all three models. Results for the base model and task-specific LoRA are included as references.

Table 7: ROUGE-L over 50 language tasks before task-wise normalization, with input rank $r = 16$ and storage budget $K = 3$. Best/second-best merging results are bold/underlined.

Method	Llama	Qwen	Mistral
Linear	42.87	56.29	60.93
TIES	46.16	56.59	61.30
DARE	45.41	56.80	59.62
DARE-TIES	37.28	54.19	57.22
K-Merge	44.64	55.23	61.92
K-Merge++	46.50	55.91	61.86
OPCM	46.53	53.35	58.48
CtM	46.06	55.88	60.83
LoRA-LEGO	43.98	52.69	56.16
Share	<u>61.68</u>	<u>65.60</u>	<u>72.65</u>
MEPC	**63.29**	**66.78**	**73.30**
Base model	5.70	50.83	24.89
Task-specific LoRA	67.69	69.18	75.26



(a) Llama-3.2-1B **(b) Qwen2.5-1.5B** **(c) Mistral-7B**

Figure 6: Task-level performance retention ratio distributions over a stream of 50 tasks on three language models with input LoRA rank $r = 16$ and storage budget $K = 3$.

C.2 Task-Level Performance Retention Distribution

The aggregate score $s^{(50)}$ summarizes average performance across tasks but does not capture variation among individual tasks. To examine task-level performance, Figure 6 shows the final task

performance retention ratio distribution of each method on the three language models with input LoRA rank $r = 16$ and storage budget $K = 3$. We partition tasks into five intervals according to their final task performance retention ratio $\rho_i^{(50)}$ and report the number of tasks in each interval.

Figure 6 shows that MEPC maintains high performance on a broader set of tasks under the same storage budget. Across all three models, MEPC retains more tasks with $\rho_i^{(50)} \geq 0.9$ than Share, while the number of tasks with $\rho_i^{(50)} < 0.75$ never exceeds 1. Thus, the gains of MEPC are not driven by a small number of high-scoring tasks but reflect broader and more consistent task-level performance retention.

C.3 Applicability to Vision Models

We further evaluate MEPC on DINOv2 ViT-L/14 (Oquab et al., 2024) and all 19 tasks in VTAB-1k (Zhai et al., 2019) to examine its applicability to vision task streams. We fix the input LoRA rank at $r = 16$, set the storage budget to $K \in \{1, 3, 5\}$, and report the aggregate score $s^{(19)}$ after merging all tasks.

As shown in Table 8, MEPC achieves the highest final aggregate score under all three storage budgets, outperforming Share by 1.28, 1.41, and 0.72 percentage points, respectively. As the budget increases from $K = 1$ to $K = 5$, the aggregate score of MEPC rises from 0.6735 to 0.9708 while remaining the highest at every budget level. Share and MEPC both substantially outperform methods that operate on complete LoRA adapters as merging units, indicating that reusing low-rank directions across tasks is also effective for vision task streams. The consistent improvement of MEPC over Share further indicates its applicability across model architectures and task modalities.

Table 8: Aggregate score $s^{(19)}$ on all 19 VTAB-1k tasks with DINOv2 ViT-L/14 and input LoRA rank $r = 16$. Best/second-best results are bold/underlined.

Budget	Linear	TIES	DARE	DARE-TIES	K-Merge	K-Merge++	OPCM	CtM	LoRA-LEGO	Share	MEPC
$K = 1$	0.6051	0.5721	0.6019	0.6001	0.6065	0.6088	0.6424	0.6385	0.6139	<u>0.6607</u>	**0.6735**
$K = 3$	0.6353	0.6339	0.6302	0.6201	0.6269	0.6272	0.6631	0.6410	0.6224	<u>0.9027</u>	**0.9168**
$K = 5$	0.6561	0.6892	0.6564	0.6301	0.6635	0.6671	0.6670	0.6474	0.6383	<u>0.9636</u>	**0.9708**

C.4 Robustness to Mixed-Rank LoRA Streams

In practical deployments, LoRAs for different tasks often have different ranks. To evaluate the robustness of MEPC to mixed-rank LoRA streams, we construct task streams comprising rank-8, rank-16, and rank-32 LoRAs on Llama-3.2-1B-Instruct and Qwen2.5-1.5B-Instruct. We consider three rank mixtures, denoted by M_1, M_2, and M_3, which contain $(10, 35, 5)$, $(20, 20, 10)$, and $(30, 5, 15)$ rank-8, rank-16, and rank-32 LoRAs, respectively. All three mixtures contain the same number of tasks and have the same total input rank, differing only in their rank compositions. For slot-based baselines, we use rank-32 slots and zero-pad lower-rank LoRAs.

Table 9: Aggregate score S under three mixed-rank configurations.

	Llama-3.2-1B-Instruct				Qwen2.5-1.5B-Instruct			
Method	M_1	M_2	M_3	Avg.	M_1	M_2	M_3	Avg.
Linear	62.1	64.5	64.9	63.8	81.5	80.8	80.2	80.8
TIES	65.2	68.7	68.7	67.5	83.7	82.6	82.0	82.8
DARE	66.5	64.5	66.2	65.7	81.4	80.9	80.8	81.0
DARE-TIES	51.8	42.9	52.2	49.0	77.8	78.6	78.6	78.4
K-Merge	65.2	65.7	66.7	65.8	79.9	80.3	80.8	80.3
K-Merge++	64.6	66.0	64.5	65.0	79.9	80.3	80.8	80.3
CtM	71.1	70.8	71.1	71.0	82.4	83.2	82.2	82.6
LoRA-LEGO	69.1	70.1	70.1	69.8	81.1	81.3	80.8	81.1
Share	<u>90.5</u>	<u>89.9</u>	<u>89.8</u>	<u>90.1</u>	<u>93.8</u>	<u>93.5</u>	<u>93.8</u>	<u>93.7</u>
MEPC (Ours)	**92.4**	**90.9**	**90.8**	**91.4**	**95.8**	**96.0**	**96.2**	**96.0**

Table 9 reports the aggregate results under different rank mixtures. MEPC achieves the highest score across all configurations on both models, outperforming Share by an average of 1.3 and 2.3 percentage points on Llama and Qwen, respectively. These results show that the advantage of MEPC under uniform-rank LoRA merging extends to the mixed-rank setting. Moreover, MEPC consistently

maintains its advantage as the relative proportions of rank-8, rank-16, and rank-32 LoRAs change, indicating that its overall performance does not depend on a particular rank composition.

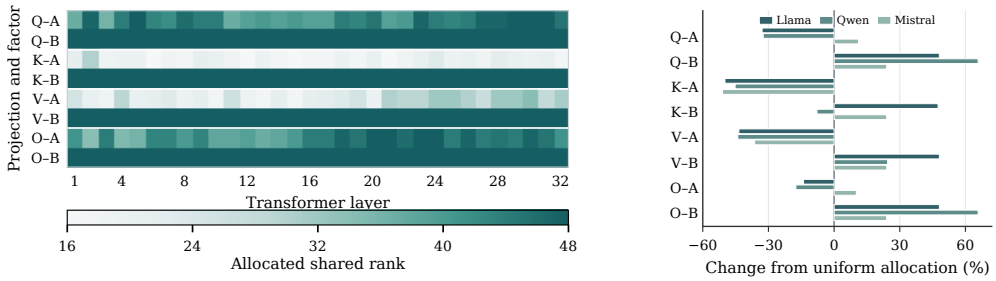
Table 10 further reports the results grouped by input rank. MEPC consistently outperforms Share across all rank groups and mixture configurations, showing that its aggregate improvement is not dominated by a particular input rank or achieved at the expense of other rank groups. MEPC also exhibits smaller performance variation across ranks overall. In contrast, although some baselines perform well for individual rank groups, their performance varies substantially across input ranks and does not translate into better aggregate results. The smaller cross-rank variation suggests that MEPC is less sensitive to any particular input rank, enabling more balanced retention of LoRAs with different ranks. Overall, MEPC accommodates varying input ranks while preserving both strong aggregate performance and balanced performance across rank groups.

Table 10: Rank-wise score S_r under three mixed-rank configurations.

	M_1			M_2			M_3		
Method	r=8	r=16	r=32	r=8	r=16	r=32	r=8	r=16	r=32
Llama-3.2-1B-Instruct									
Linear	52.2	63.9	68.9	68.0	70.9	44.4	64.6	79.4	60.8
TIES	59.3	65.6	73.8	69.8	75.7	52.5	70.0	83.7	61.1
DARE	64.3	71.5	36.0	68.4	70.7	44.3	66.3	79.6	61.3
DARE-TIES	50.6	56.7	19.9	47.0	47.4	25.8	49.1	58.0	56.6
K-Merge	50.0	69.0	68.5	68.9	71.6	47.3	68.9	72.4	60.4
K-Merge++	53.3	70.1	48.8	69.4	71.6	47.7	67.5	69.2	56.8
CtM	60.3	75.8	59.5	69.7	80.2	54.0	73.6	82.3	62.4
LoRA-LEGO	61.5	72.1	63.4	68.8	80.1	52.5	71.3	**90.2**	61.2
Share	86.2	92.0	88.3	87.8	92.8	88.3	89.5	81.8	93.0
MEPC (Ours)	**87.3**	**94.1**	**90.3**	**88.5**	**93.3**	**91.2**	**89.9**	86.9	**93.9**
Qwen2.5-1.5B-Instruct									
Linear	71.3	86.9	64.1	78.9	90.4	65.3	80.4	95.9	74.8
TIES	71.8	87.2	83.1	79.2	91.5	71.5	82.4	95.5	76.5
DARE	71.7	86.7	63.5	79.0	90.5	65.6	81.0	97.2	74.9
DARE-TIES	66.2	84.7	52.8	76.9	88.5	62.3	78.7	95.6	72.8
K-Merge	64.6	87.5	56.9	78.0	89.8	65.9	80.9	94.4	76.0
K-Merge++	64.6	87.5	56.9	78.0	89.8	65.9	80.9	94.4	76.0
CtM	70.3	86.6	77.3	79.4	92.9	71.4	82.1	96.6	77.5
LoRA-LEGO	69.3	84.7	79.5	76.7	91.9	69.5	81.0	**97.6**	74.8
Share	87.9	95.4	93.8	92.4	96.5	89.8	93.2	94.2	94.9
MEPC (Ours)	**91.4**	**97.0**	**96.6**	**94.4**	**97.7**	**95.8**	**96.1**	96.3	**96.3**

C.5 Importance-Aware Compression Analysis

To satisfy the storage constraint, MEPC applies importance-aware compression by removing low-importance directions across attention projections and LoRA factor sides, and produces a non-uniform rank configuration.



(a) Rank allocation on Mistral **(b) Mean relative allocation**

Figure 7: Shared rank allocation learned by MEPC with input LoRA rank $r = 16$ and storage budget $K = 3$. (a) Shared ranks across Transformer layers of Mistral-7B-Instruct-v0.2. (b) Average changes relative to uniform rank allocation on Llama, Qwen, and Mistral.

Figure 7(a) shows the shared-rank allocation across Transformer layers on Mistral. The $\mathbf{B}$ side generally retains higher shared ranks, whereas the $\mathbf{A}$ side exhibits greater variation across layers. Figure 7(b) summarizes the average changes relative to uniform rank allocation across the three

models. The average ranks on the $\mathbf{B}$ side of the query, value, and output projections increase by 24% to 66%, whereas those on the $\mathbf{A}$ side of the key and value projections decrease by 36% to 51%. These results indicate that attention projections and LoRA factor sides have different capacity requirements, which MEPC captures through non-uniform rank configurations across modules, factor sides, and layers.

This allocation pattern is consistent with prior observations of the distinct roles of the two LoRA factors. Zhu et al. (2024) show that $\mathbf{A}$ and $\mathbf{B}$ play different roles during adaptation and that optimizing $\mathbf{B}$ is more important for adaptation performance. HydraLoRA (Tian et al., 2024) shares $\mathbf{A}$ while using multiple $\mathbf{B}$ branches to represent heterogeneous task components.

C.6 Computational Cost of Joint Evaluation

We compare MEPC with *Importance-aware + Joint* under the same setting as the ablation study, with input rank $r = 16$, storage budget $K = 3$, and a stream of 50 tasks. The latter evaluates all directions with the joint deletion loss at every deletion, whereas MEPC restricts joint evaluation to the local candidates selected by the one-sided loss. As shown in Figure 8, MEPC requires consistently less update time across all three models. Together with the results in Table 2, this shows that confining joint evaluation to local candidates substantially reduces computation while retaining comparable or better performance.

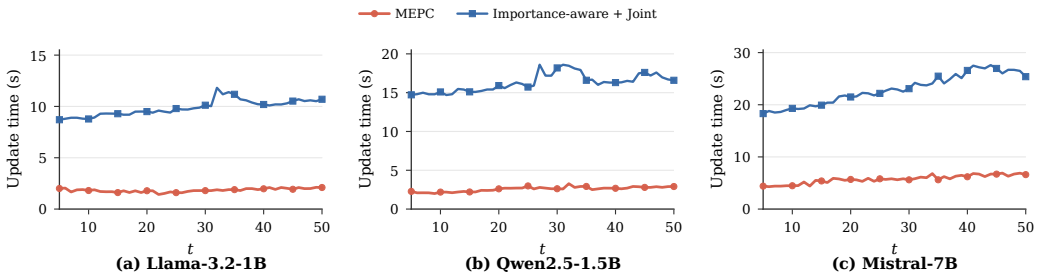


Figure 8: Per-step update time of MEPC and *Importance-aware + Joint* over a stream of 50 tasks.