

CoDe-LoRA: Mitigating the Orthogonality Dilemma in Continual Learning of LLMs via Knowledge Consolidation and Decoupling

Maoqi Liu¹, Quan Fang^{1†}, Yufei He²

¹Beijing University of Posts and Telecommunications

²National University of Singapore

qfang@bupt.edu.cn

Abstract

Continual learning (CL) is essential for Large Language Models (LLMs) to sequentially adapt to evolving tasks. To mitigate catastrophic forgetting, recent advances implement low-rank adaptation with orthogonal projections (e.g., O-LoRA) to isolate task parameters. However, we reveal that such strict geometric constraints trigger an “Orthogonality Dilemma”: rigid parameter isolation impedes the transfer and accumulation of shared representations across semantically related tasks. In this work, we propose a new replay-free method, called Consolidation and Decoupling LoRA (CoDe-LoRA), for CL of LLMs. CoDe-LoRA disentangles the learning process into *Consolidating Universal Knowledge* and *Decoupling Task-Specific Knowledge*. To achieve this, CoDe-LoRA leverages an adaptive null space projection mechanism and semantic routing to balance knowledge accumulation with task-specific adaptation. Experimental results on diverse CL benchmarks demonstrate that CoDe-LoRA consistently outperforms state-of-the-art methods. Our code is available at <https://github.com/Estrellajer/CoDe-LoRA>.

1 Introduction

Large Language Models (LLMs) have redefined the boundaries of natural language processing, exhibiting remarkable capabilities across a wide range of tasks (Wei et al., 2021). However, adapting these models to a continuous stream of evolving tasks, known as *Continual Learning* (CL) (De Lange et al., 2021), remains challenging. While parameter-efficient fine-tuning (PEFT) methods, particularly low-rank adaptation (LoRA) (Hu et al., 2021), have significantly lowered the cost of adaptation, naively applying them to sequential tasks leads to *catastrophic forgetting* (Li and

Hoiem, 2017), where learning new tasks degrades previously acquired capabilities.

To mitigate forgetting, recent work combines PEFT with orthogonal projection strategies (Saha et al., 2021; Wang et al., 2023a). A representative example is O-LoRA (Wang et al., 2023a), which isolates task-specific parameters in mutually orthogonal subspaces. Although effective at reducing inter-task interference, such rigid isolation introduces a trade-off between interference mitigation and knowledge transfer. When sequential tasks share semantic structure, their optimal parameter updates should naturally overlap. Enforcing strict orthogonality removes these shared components, reducing representation reuse and weakening cross-task transfer. As illustrated in Figure 1, this phenomenon reveals an “Orthogonality Dilemma”: the same geometric constraint that protects previous knowledge can also restrict transferable representation learning.

This limitation highlights a key challenge in continual adaptation: how to preserve prior knowledge while still accumulating transferable representations across related tasks. Motivated by this tension, we investigate two questions:

- Q1. *Why does strict orthogonality limit generalization across related tasks, and under what conditions is this limitation most severe?*
- Q2. *How can we design a parameter-efficient framework that preserves transferable representations while maintaining task-specific expertise in a replay-free setting?*

In this paper, we introduce Consolidation and Decoupling LoRA (CoDe-LoRA), a dual-branch continual adaptation framework motivated by the orthogonality-induced transfer limitation. We provide a geometric characterization showing that strict orthogonal isolation becomes increasingly detrimental as task similarity grows.

[†]Corresponding author.

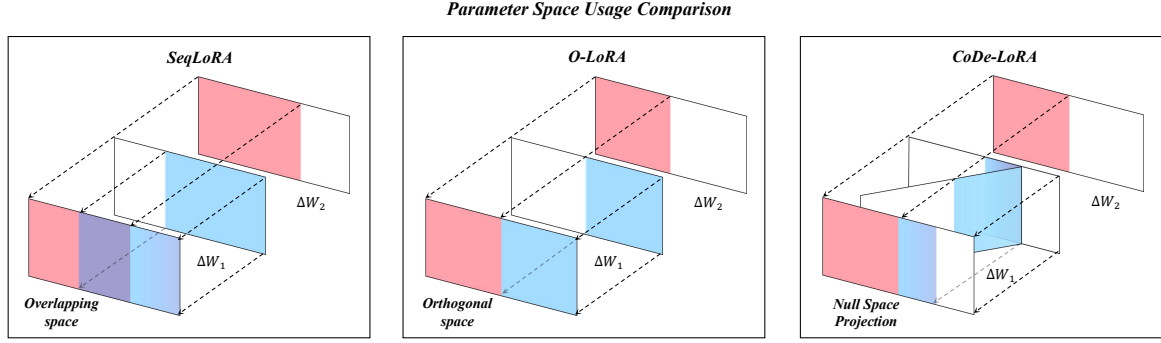


Figure 1: Conceptual illustration of parameter space utilization. Traditional LoRA suffers from interference, while O-LoRA’s strict orthogonality hinders knowledge sharing. CoDe-LoRA leverages null space projection to consolidate shared knowledge.

CoDe-LoRA separates transferable representation consolidation from task-specific specialization through two complementary branches. Unlike strict orthogonal isolation methods that suppress overlapping representations entirely, our framework preserves transferable components while constraining interference-prone directions. Specifically, the *Consolidation Branch* (*Co-LoRA*) maintains shared transferable representations through adaptive null-space projection with dynamic scaling to stabilize long-horizon continual adaptation, as empirically verified in Appendix A.2 (Table 5). Meanwhile, the *Decoupling Branch* (*De-LoRA*) maintains a pool of independent LoRA experts together with a prototype-based semantic routing mechanism for robust inference without explicit task identifiers. Importantly, CoDe-LoRA operates without relying on historical data replay, avoiding additional storage overhead and privacy concerns.

Our main contributions are summarized as follows:

- We identify and analyze the **Orthogonality Dilemma** in PEFT-based continual learning, showing that strict parameter isolation becomes increasingly detrimental as task similarity grows.
- We propose **CoDe-LoRA**, a dual-branch continual adaptation framework that separates transferable representation consolidation from task-specific specialization through adaptive null-space projection and prototype-based expert decoupling.
- Extensive experiments on diverse continual learning benchmarks demonstrate that CoDe-LoRA consistently outperforms strong base-

lines and effectively balances knowledge transfer and interference mitigation.

2 Related Work

2.1 Parameter-Efficient Fine-Tuning

Parameter-Efficient Fine-Tuning (PEFT) adapts Large Language Models (LLMs) with minimal additional parameters. Early approaches such as Adapters (Houlsby et al., 2019) and Prompt/Prefix Tuning (Li and Liang, 2021; Lester et al., 2021) introduced lightweight modules while freezing the backbone. LoRA (Hu et al., 2021) further improves efficiency through low-rank matrices. Although effective for static tasks, sequential adaptation with PEFT methods often leads to catastrophic forgetting.

2.2 Continual Fine-Tuning for LLMs

Continual fine-tuning methods mainly rely on regularization (Kirkpatrick et al., 2017), replay (Qin and Joty, 2021; He et al., 2024), or parameter expansion (Razdaibiedina et al., 2023). Expansion-based methods avoid interference through task-specific parameters, but their parameter cost grows with the number of tasks. Replay and regularization methods preserve prior knowledge through memory buffers or parameter constraints, yet often incur computational overhead or privacy concerns. Recently, mixture-of-experts (MoE) routing frameworks such as MoLE-CIE (Wang et al., 2025) have been introduced for continual adaptation; however, they typically still rely on historical exemplar replay to prevent routing drift and maintain performance.

More recently, orthogonality-based methods such as O-LoRA (Wang et al., 2023a), CLoRA (Lu

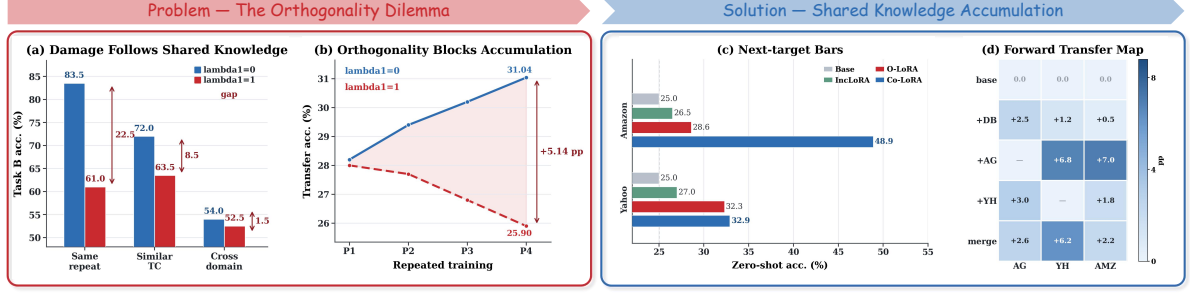


Figure 2: Preliminary evidence for the orthogonality dilemma and transferable representation accumulation. **(a)** Orthogonality-induced degradation under different task similarity levels. **(b)** Transfer trajectories over repeated training with and without orthogonal constraints. **(c)** Next-task zero-shot transfer comparison across continual adaptation methods. **(d)** Forward transfer dynamics across sequential tasks.

et al., 2025), and N-LoRA (Yang et al., 2025b) mitigate forgetting by projecting updates into orthogonal or null subspaces. Although effective at reducing interference, strict orthogonal isolation may limit representation reuse across related tasks. In contrast, CoDe-LoRA separates shared representation consolidation from task-specific specialization, balancing knowledge transfer and interference mitigation through adaptive null-space projection without historical data replay.

3 Preliminary

3.1 Problem Definition

We consider continual learning (CL) over a sequence of tasks $\{\mathcal{T}^{(1)}, \mathcal{T}^{(2)}, \dots, \mathcal{T}^{(T)}\}$. Each task $\mathcal{T}^{(t)}$ is associated with a dataset $\mathcal{D}^{(t)} = \{(x, y)\}$, where x denotes the input prompt and y the target sequence. Under the PEFT setting, the backbone parameters θ_{base} remain frozen while task-specific updates $\Delta\theta^{(t)}$ are learned with a small number of trainable parameters. Sequentially merging these updates often leads to catastrophic forgetting on previous tasks.

3.2 Orthogonal Continual Adaptation

To mitigate forgetting, O-LoRA (Wang et al., 2023a) constrains each task update to be orthogonal to previous task subspaces. For task t , the LoRA update is parameterized as $\Delta W_t = B_t A_t$, where $A_t \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B_t \in \mathbb{R}^{d_{\text{out}} \times r}$. Orthogonality is imposed through:

$$A_i A_t^\top = 0, \quad \forall i < t \quad (1)$$

This constraint is typically enforced through orthogonality regularization during training.

3.3 Orthogonality Dilemma

Although orthogonal constraints reduce interference, they may also suppress transferable representations across closely related tasks. Let $\mathcal{T}^{(i)}$ and $\mathcal{T}^{(j)}$ denote two tasks whose optimal update subspaces U_i^* and U_j^* overlap with degree:

$$\rho = \frac{\dim(U_i^* \cap U_j^*)}{\min(\dim(U_i^*), \dim(U_j^*))} \quad (2)$$

When $\rho > 0$, enforcing strict orthogonality removes shared update components, introducing an approximation gap that grows with the degree of subspace overlap.

This creates an ‘‘Orthogonality Dilemma’’: the same constraint that protects previous knowledge can also limit transferable representation learning when tasks are related.

3.4 Preliminary Evidence and Mechanism Validation

Figure 2 demonstrates these dynamics: (a) orthogonality-induced degradation worsens as task similarity grows; (b) unconstrained fine-tuning accumulates transferable knowledge over iterations, whereas strict orthogonality suppresses it; (c) preserving shared representations improves zero-shot transfer; and (d) transfer gains remain positive across task sequences.

Mechanism Validation. To investigate the underlying cause, we define task similarity via frozen centroid cosine similarity $\text{sim}(\mathcal{T}^{(i)}, \mathcal{T}^{(j)}) = c(\mathcal{T}^{(i)}) \cdot c(\mathcal{T}^{(j)})$ and update overlap via prior-subspace energy $\mathcal{E}_{\text{prior}}(\Delta W_t) = \|P_{A_{<t}} \Delta W_t^\top\|_F^2 / \|\Delta W_t^\top\|_F^2$. Sweeping O-LoRA’s orthogonality penalty $\lambda \in [0, 1.0]$ monotonically suppresses this overlap

from $5.19\% \pm 0.29\%$ to $0.16\% \pm 0.02\%$. At $\lambda = 0$, this overlap is $2.96\times$ higher than a dimension-matched random baseline (higher in 9/9 evaluations). Furthermore, directly restoring the prior-aligned update component on Yahoo $\rightarrow$ AGNews improves Exact Match (+8.57) and label margin (+0.296). This confirms that strict orthogonality suppresses non-random, functionally beneficial shared signals, motivating our dual-branch design.

4 Methodology

4.1 Overall Framework

To resolve the orthogonality dilemma (Section 3.3), we propose CoDe-LoRA (illustrated in Figure 3), a dual-branch framework designed to consolidate shared representations while decoupling task-specific expertise. Built upon a frozen backbone θ_{base} , our architecture operates through two parallel pathways: the Consolidation Branch (Co-LoRA, θ_{shared}), which accumulates transferable representations across tasks via orthogonal projection (Sec. 4.2), and the Decoupling Branch (De-LoRA), which maintains a pool of independent task-specific LoRA experts $\{\theta_{\text{spec}}^k\}_{k=1}^T$ that captures domain nuances without interference (Sec. 4.3). These components are ultimately integrated via a confidence-based fallback selection strategy (Sec. 4.4) to ensure robust inference.

4.2 Consolidating Shared Representations

While strict parameter isolation (e.g., O-LoRA) prevents catastrophic forgetting, it often hinders the accumulation of shared representations across closely related tasks. Unlike strict orthogonal isolation methods that entirely discard overlapping update components, our consolidation branch preserves accumulated shared representations while constraining only newly introduced conflicting update directions. To achieve this objective, we propose a consolidation mechanism based on null space projection and dynamic scaling.

Consolidation Algorithm. After training on each new task $\mathcal{T}_t$, we update the parameters of the Consolidation Branch (Co-LoRA) through a three-step consolidation process. The Consolidation Branch is parameterized by low-rank weights $W_{\text{shared}} = B_{\text{shared}}A_{\text{shared}}$. Let $W_{\text{acc}}^{(t-1)}$ denote the accumulated shared weight matrix of Co-LoRA up to task $t-1$, and let $\Delta W_{\text{shared}}^{(t)}$ represent the param-

eter update learned in the Consolidation Branch during the training of task t .

Null Space Projection. We project $\Delta W_{\text{shared}}^{(t)}$ onto the null space of the existing representations represented by $W_{\text{acc}}^{(t-1)}$. Specifically, we perform Singular Value Decomposition (SVD) on $W_{\text{acc}}^{(t-1)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ to obtain its top- d_v left singular vectors $U \in \mathbb{R}^{d_{\text{out}} \times d_v}$, which form an orthonormal basis for its column space. Here, d_v represents the rank of the accumulated column space. The new component ΔW_{null} is then computed by removing the projection of $\Delta W_{\text{shared}}^{(t)}$ onto this subspace:

$$\Delta W_{\text{null}} = \Delta W_{\text{shared}}^{(t)} - UU^\top \Delta W_{\text{shared}}^{(t)} \quad (3)$$

This projection reduces interference with previously consolidated representations. To improve scalability for large LLMs, we implement the projection using randomized low-rank SVD directly on factorized LoRA matrices. Details and efficiency analysis are provided in Appendix A.4.

Dynamic Scaling Constraint. To maintain the stability of the parameter magnitude, we introduce time-varying scaling coefficients $c_t = \sqrt{\frac{t-1}{t}}$ and $s_t = \frac{1}{\sqrt{t}}$ satisfying the normalization identity $c_t^2 + s_t^2 = 1$. Since the null space projection ensures that each new update is orthogonal to the existing parameters, we scale the null space update component:

$$\Delta W_{\text{scaled}} = s_t \Delta W_{\text{null}} \quad (4)$$

This constraint prevents the cumulative parameter magnitude from exploding or drifting as the number of sequential tasks increases. We provide the theoretical proof of the stable parameter bound and its empirical verification in Appendix A.2.

The final consolidated weight matrix is updated by merging the scaled null space component with the previous state:

$$W_{\text{acc}}^{(t)} = c_t W_{\text{acc}}^{(t-1)} + s_t \Delta W_{\text{null}} \quad (5)$$

This recursive formulation ensures that the consolidated weight matrix represents a normalized accumulation of orthogonal task updates. The choice of scaling coefficients stabilizes the parameter magnitude across tasks as follows:

Proposition 1. *Given the recursive update rule defined in Equation (5), assuming task updates are approximately orthogonal ($\langle \Delta W_{\text{shared}}^{(i)}, \Delta W_{\text{shared}}^{(j)} \rangle \approx$*

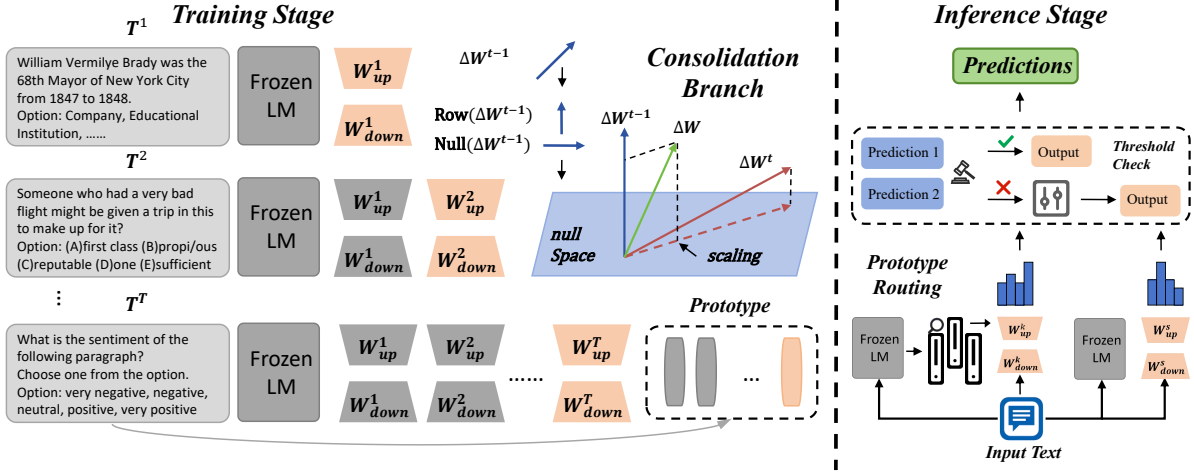


Figure 3: The CoDe-LoRA framework. (Left) Training: Knowledge is disentangled into task-specific experts and a shared branch consolidated via null space projection. (Right) Inference: Inputs are routed to experts via prototype similarity, with a confidence-based fallback to the shared branch for robust prediction.

0 for $i \neq j$), and assuming the initial LoRA weights are zero ($W_{acc}^{(0)} = W^{(0)} = 0$), the distance between the consolidated model and the pre-trained model remains bounded:

$$\left\| W_{acc}^{(t)} - W^{(0)} \right\|_F^2 \leq M_t \quad (6)$$

where $M_t = \max_{i \in [1, t]} \left\| \Delta W_{shared}^{(i)} \right\|_F^2$.

This proposition (proof and empirical verification in Appendix A.2) suggests that the proposed scaling rule stabilizes the parameter magnitude across sequential continual tasks.

4.3 Decoupling Task-Specific Expertise

While the Consolidation Branch captures transferable representations, task-specific nuances are isolated within the Decoupling Branch (De-LoRA), consisting of a pool of task-specific experts $\{\theta_{spec}^k\}_{k=1}^T$. To enable task-agnostic inference, we introduce a semantic routing mechanism that leverages task prototypes to dynamically activate the most relevant De-LoRA expert for each input.

Prototype Construction. For each task, we construct a prototype $P_k \in \mathbb{R}^{d_{model}}$ by averaging normalized backbone embeddings from a small subset of training samples:

$$P_k = \text{Norm} \left(\frac{1}{N} \sum_{i=1}^N f_{\theta_{base}}(x_i) \right) \quad (7)$$

where $f_{\theta_{base}}(\cdot)$ denotes the feature extraction process of the frozen backbone and N is the number

of randomly selected samples (fixed to $N = 10$ throughout experiments). Crucially, because prototypes and runtime query representations are extracted solely using the frozen base model θ_{base} rather than adapter-modulated layers, the semantic matching space remains strictly invariant over sequential learning phases, fundamentally eliminating representation drift.

Routing Procedure. During inference, the system identifies the optimal De-LoRA expert $\theta_{spec}^{k^*}$ for a given input x by measuring the similarity between the input embedding $e_x = f_{\theta_{base}}(x)$ and the stored prototypes $\{P_k\}_{k=1}^T$. Since prototypes are pre-normalized, the routing index k^* is determined by maximizing the cosine similarity:

$$k^* = \arg \max_{k \in \{1, \dots, T\}} \frac{e_x \cdot P_k}{\|e_x\|} \quad (8)$$

The selected expert $\theta_{spec}^{k^*}$ is then activated and integrated with the Consolidation Branch for inference via the fallback strategy described in Section 4.4. This similarity-based routing facilitates positive forward transfer: for inputs that bridge multiple task distributions, the mechanism naturally selects the most semantically related expert, enabling the model to leverage relevant prior knowledge. Robustness across diverse task sequences is further validated in Appendix A.7.

4.4 Knowledge Integration and Fallback Selection

To produce the final prediction, we employ a confidence-based fallback selection strategy that

Method	Standard CL Benchmark				Large Number of Tasks			
	Order-1	Order-2	Order-3	Avg	Order-4	Order-5	Order-6	Avg
Per-LoRA (Oracle)	81.4	81.3	81.4	81.4	77.6	77.4	77.5	77.5
MTL	80.0	80.0	80.0	80.0	76.5	76.5	76.5	76.5
Sequential FT								
SeqFT	18.9	24.9	41.7	28.5	7.4	7.4	7.5	7.4
SeqLoRA	44.6	32.7	53.7	43.7	0.6	1.9	1.6	1.4
IncLoRA	66.0	64.9	68.3	66.4	63.3	58.5	61.7	61.2
Classic CL								
Replay	55.2	56.9	61.3	57.8	55.0	54.6	53.1	54.2
EWC	48.7	47.7	54.5	50.3	45.3	44.5	45.6	45.1
LwF	54.4	53.1	49.6	52.3	50.1	43.1	47.4	46.9
L2P	60.3	61.7	61.1	61.0	57.5	53.8	56.9	56.1
LFPT5	67.6	72.6	77.9	72.7	70.4	68.2	69.1	69.2
Advanced CL								
O-LoRA	75.4	75.7	76.3	75.8	72.3	64.8	71.6	69.6
MIGU	77.1	77.0	75.6	76.6	67.3	68.5	74.2	70.0
N-LoRA	79.2	78.4	78.8	78.8	73.6	70.3	73.2	72.4
CLoRA	79.7	79.1	78.2	79.0	70.7	65.6	68.2	68.1
Co-LoRA	78.4 $\pm$ 0.2	78.1 $\pm$ 0.3	78.3 $\pm$ 0.3	78.3 $\pm$ 0.3	70.8 $\pm$ 0.4	70.5 $\pm$ 0.4	70.7 $\pm$ 0.3	70.7 $\pm$ 0.4
De-LoRA	80.1 $\pm$ 0.2	79.8 $\pm$ 0.3	80.0 $\pm$ 0.2	80.0 $\pm$ 0.2	74.4 $\pm$ 0.3	74.0 $\pm$ 0.4	74.2 $\pm$ 0.3	74.2 $\pm$ 0.3
CoDe-LoRA (Ours)	80.8$\pm$0.2	80.4$\pm$0.2	80.6$\pm$0.2	80.6$\pm$0.2	76.8$\pm$0.3	76.4$\pm$0.3	76.6$\pm$0.3	76.6$\pm$0.3

Table 1: Performance comparison of different methods using the T5 model on Standard CL Benchmark and Large Number of Tasks. The average accuracy after training on the final task is reported. Subscripts denote standard deviations across 3 random seeds.

prioritizes specialized expertise while leveraging the consolidated shared branch as a robust prior. Here, the routing confidence is defined as the maximum cosine similarity between the input embedding and task prototypes. If the routing confidence exceeds a threshold τ (a hyperparameter set to 0.75 by default; see Appendix A.6 for sensitivity analysis), the system routes inference to the task-specific expert $\theta_{\text{spec}}^{k*}$, executing $\hat{y} = \text{Model}(x; \theta_{\text{base}}, \theta_{\text{spec}}^{k*})$. Otherwise, the model falls back to the consolidated shared branch, executing $\hat{y} = \text{Model}(x; \theta_{\text{base}}, \theta_{\text{shared}})$. This fallback strategy dynamically balances stability and plasticity by preserving high-capacity specialized expertise while utilizing the consolidated branch as a robust safety net for ambiguous or out-of-distribution inputs. The complete training and consolidation workflow is detailed in Appendix A.4.

5 Experiments

5.1 Experimental Setup

Datasets. We evaluate CoDe-LoRA on three benchmarks: the Standard CL Benchmark (Zhang et al., 2015), the Large Number of Tasks Benchmark (Razdaibiedina et al., 2023), and the TRACE benchmark (Wang et al., 2023b) which contains diverse generative tasks. Following Qin and Joty

(2021), we evaluate across three distinct task orders for each benchmark to ensure robustness. The complete specifications, including task sequences, dataset statistics, and natural language instructions, are detailed in Appendix A.9 and A.10.

Evaluation Metrics. Let $a_{i,j}$ denote the performance on task i after training on task j . To assess continual learning performance, we report two primary metrics: (1) **Average Performance (AP)**, computed as the average task performance at the end of the continual sequence (Average Accuracy for classification benchmarks and Overall Performance for generative benchmarks); and (2) **Backward Transfer (BWT)**, which measures the retention of previously learned knowledge across tasks. We additionally analyze forward transfer behavior to evaluate the model’s ability to accumulate transferable representations across sequential tasks. Formal definitions are provided in Appendix A.5.

Baselines. We compare CoDe-LoRA against four categories of continual learning baselines: (1) upper-reference baselines, including Multi-Task Learning (MTL) and Per-LoRA (Oracle); (2) sequential fine-tuning methods, including SeqFT (de Masson D’Autume et al., 2019), SeqLoRA, and IncLoRA; (3) continual learning methods, including EWC (Kirkpatrick et al., 2017),

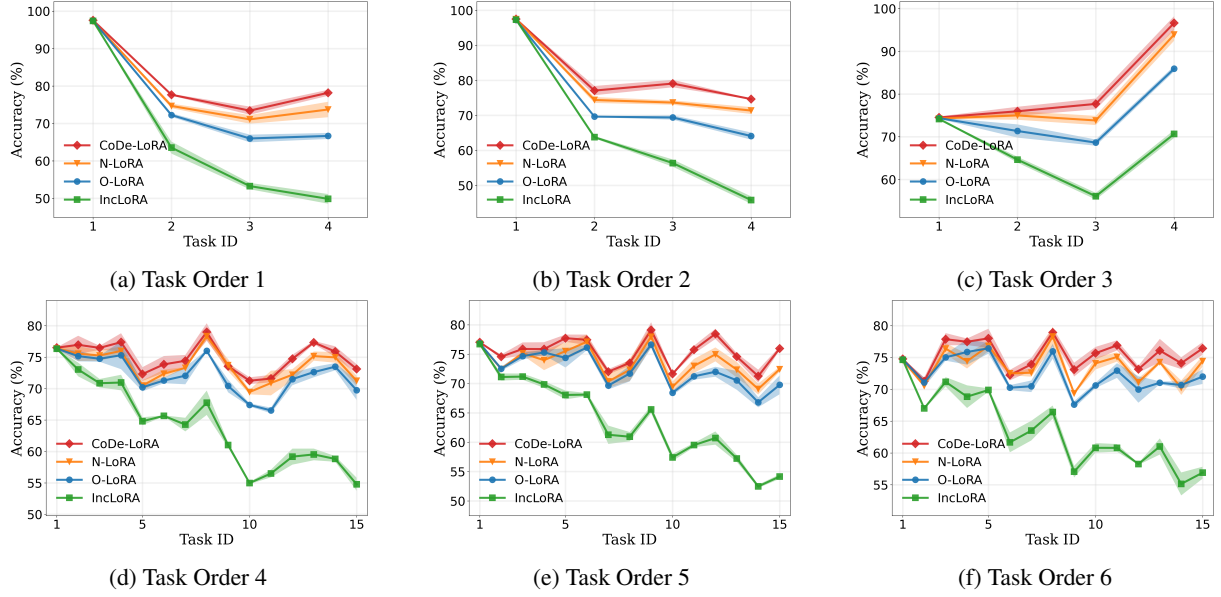


Figure 4: Performance curves of different methods across six task orders on the Standard CL Benchmark (Orders 1-3) and Large Number of Tasks Benchmark (Orders 4-6). Results are averaged over three independent runs.

Method	Llama-2-7B				Qwen3-0.6B				Qwen3.5-4B			
	Std. CL	Long Seq.	Trace OP	Trace BWT	Std. CL	Long Seq.	Trace OP	Trace BWT	Std. CL	Long Seq.	Trace OP	Trace BWT
O-LoRA	76.0	71.3	42.8	-7.2	59.4	54.1	39.8	-8.2	65.0	60.8	42.2	-7.0
N-LoRA	77.5	72.8	44.3	-5.7	62.0	56.6	41.3	-6.7	67.0	62.9	44.1	-5.7
SEEKR	77.0	71.0	55.0	-3.5	66.8	63.5	43.8	-4.8	70.6	67.4	45.8	-4.0
MoLE-CIE	76.5	71.8	43.0	-5.2	68.5	64.2	41.0	-4.9	68.0	64.5	42.5	-4.6
CoDe-LoRA	81.5	76.8	46.0	-1.5	72.1	68.6	42.6	-2.3	72.4	68.7	44.8	-1.9

Table 2: Performance comparison across different backbone models and benchmarks. Std. CL and Long Seq. report average accuracy over three task orders on the standard continual learning benchmark and the long-sequence benchmark, respectively. Trace OP and Trace BWT are the two metrics of the TRACE benchmark. Bold indicates the best result for each model and benchmark.

LwF (Li and Hoiem, 2017), Replay (Lopez-Paz and Ranzato, 2017), L2P (Wang et al., 2022), and LFPT5 (Qin and Joty, 2021); and (4) parameter-efficient continual learning methods, including O-LoRA (Wang et al., 2023a), MIGU (Du et al., 2024), N-LoRA (Yang et al., 2025b), CLoRA (Lu et al., 2025), and SEEKR (He et al., 2024).

Implementation Details. We evaluate CoDe-LoRA across different model families and scales using T5 (Raffel et al., 2020), Llama 2-7B (Touvron et al., 2023), Qwen3-0.6B (Yang et al., 2025a), and Qwen3.5-4B (Team, 2026). All models are optimized with AdamW (Loshchilov and Hutter, 2017). Results are averaged over three independent runs. Hyperparameters, routing details, and implementation settings are provided in Appendix A.4.

5.2 Main Results

On the Standard CL Benchmark (Table 1), CoDe-LoRA achieves the highest average accuracy (80.6%), outperforming prior PEFT-based CL

methods and exceeding the MTL reference (80.0%). This result suggests that the decoupled expert pool provides task-specific capacity while avoiding the forgetting typical of sequential fine-tuning.

On the Large Number of Tasks benchmark, the performance gap is more pronounced: CoDe-LoRA achieves 76.6% average accuracy, outperforming O-LoRA by 7.0% (76.6% vs. 69.6%). While O-LoRA reduces interference through strict orthogonal isolation, its performance degrades on longer sequences where transferable representations become increasingly important. These results indicate stronger scalability in long-horizon scenarios.

Generalization to Generative Tasks. We further evaluate CoDe-LoRA on the TRACE benchmark (Wang et al., 2023b), which contains generative tasks. As shown in Table 2, CoDe-LoRA achieves competitive generation quality (OP) and consistently improves knowledge retention (BWT) without relying on replay buffers, indicating improved retention in generative domains.

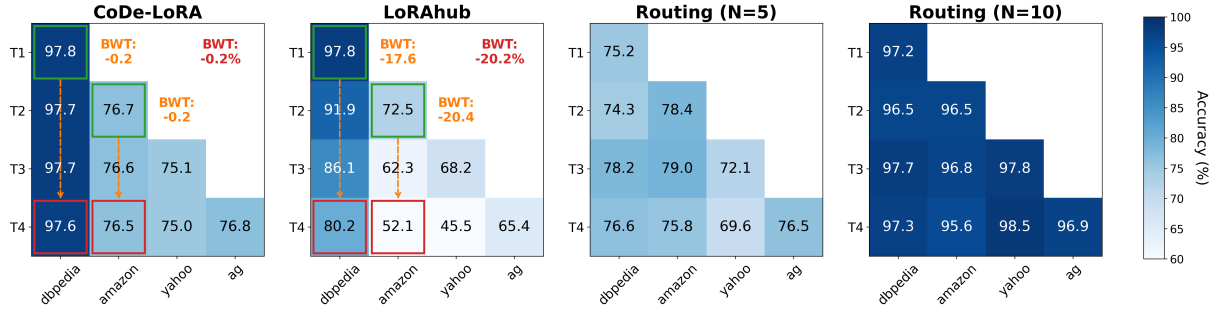


Figure 5: Ablation analysis of CoDe-LoRA showing task-wise accuracy and BWT under different ensemble strategies (left) and routing accuracy across varying prototype sample sizes N (right).

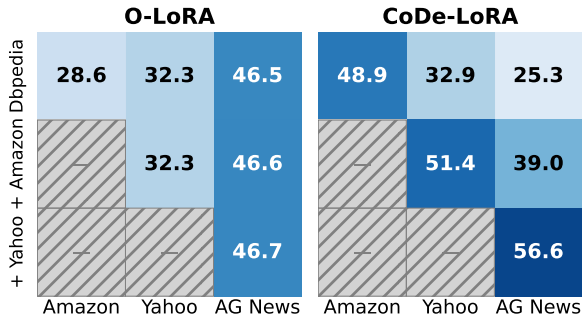


Figure 6: Zero-shot forward transfer on Qwen3-0.6B (Order 1). “N/A” indicates trained tasks.

Generalization across Model Families. To evaluate generalization across model architectures and scales, we evaluate CoDe-LoRA on Qwen3-0.6B, Qwen3.5-4B, and Llama 2-7B across isolation-based PEFT (O-LoRA, N-LoRA), replay-based (SEEKR), and routing-based MoE (MoLE-CIE (Wang et al., 2025)) baselines (Table 2). CoDe-LoRA consistently outperforms all evaluated baselines across backbones. For instance, on Qwen3.5-4B, CoDe-LoRA outperforms O-LoRA by 7.4% (72.4% vs. 65.0%) and MoLE-CIE by 4.4% (72.4% vs. 68.0%) on standard CL, while achieving substantially stronger retention on TRACE BWT (-1.9 vs. -4.6). Furthermore, compared to MoLE-CIE, CoDe-LoRA improves accuracy on Qwen3-0.6B by 3.6% while reducing active trainable parameters by 82.6% and training time by 46.5% without maintaining historical replay buffers. Compared to the replay-based SEEKR on Llama 2-7B, CoDe-LoRA also achieves superior BWT (-1.5 vs. -3.5) without storing raw historical data.

5.3 Ablation Studies and Analysis

Capacity Control. When matching the per-task adapter footprint (evaluating CoDe-LoRA with $r = 8$ against O-LoRA and N-LoRA with $r = 10$

on T5-Large, both $\approx 2.95\text{M}$ per task), CoDe-LoRA achieves 80.58% on the Standard CL Benchmark and 76.62% on the Long Sequence Benchmark, outperforming the strongest capacity-matched baseline by +3.48 and +12.67, respectively. This confirms that gains stem from decoupled knowledge consolidation rather than parameter capacity.

Ensemble Strategy. We compare CoDe-LoRA with LoRAHub (Huang et al., 2023), which directly merges task-specific LoRA modules. As shown in Figure 5, LoRAHub exhibits severe catastrophic forgetting, yielding a BWT of -20.2% compared to -0.2% for CoDe-LoRA. This suggests that direct parameter merging introduces significantly stronger cross-task interference.

Forward Transfer. Figure 6 shows the zero-shot forward transfer matrix on Qwen3-0.6B (Order 1). CoDe-LoRA achieves consistently higher forward transfer than O-LoRA, particularly after progressively accumulating knowledge across tasks (e.g., +19.1 points on Yahoo after training on dbpedia and amazon). This suggests that the consolidation branch effectively accumulates transferable representations that benefit future unseen tasks.

Impact of Prototype Samples. We analyze the sensitivity of routing performance to the number of prototype samples N . Figure 5 (right) shows that routing accuracy improves as N increases, with performance saturating around $N = 10$. This indicates that only a small number of samples are required for reliable task routing.

Contribution of Each Branch. To further analyze the necessity of the proposed dual-branch design, we compare the Consolidation Branch alone (Co-LoRA), the Decoupling Branch alone (De-LoRA), and the full CoDe-LoRA architecture on the Standard CL Benchmark, as reported in the

corresponding rows of Table 1. The consolidation

Method	Train (s)	Infer (s)	Params
O-LoRA	6384.3	483.0	9.4M (1.26%)
N-LoRA	6153.2	470.5	9.4M (1.26%)
SEEKR	15960	474.5	770M (100%)
CoDe-LoRA	5922.1	458.4	11.8M (1.58%)

Table 3: Efficiency statistics on Order 1. Params denotes trainable parameters and backbone ratio.

branch (Co-LoRA) alone captures shared representations, while the decoupling branch (De-LoRA) alone preserves task-specific adaptation. Combining both branches yields the best overall performance across all task sequences (Table 1). Additional visualizations and prediction analyses are provided in Appendix A.3.

Efficiency and Scalability Analysis. CoDe-LoRA remains highly parameter-efficient, updating only 1.58% of backbone parameters. Despite the dual-branch design, CoDe-LoRA maintains faster training and inference speeds than O-LoRA and N-LoRA (Table 3) by avoiding expensive orthogonalization steps on the backpropagation gradient path. Furthermore, on the 15-task sequence (Appendix Table 9), CoDe-LoRA’s total parameter footprint is only 37.7 MB (a negligible 2.4 MB addition over single-branch baselines), while total training time is reduced from 2.77h to 2.36h. Crucially, during inference only one routed expert is activated alongside the base backbone, keeping active execution cost identical to standard LoRA.

6 Conclusion

We revisit orthogonal continual adaptation from the perspective of transferable representation learning. We show that while strict orthogonal isolation mitigates catastrophic forgetting, it can also suppress transferable representation accumulation across related tasks. To address this limitation, we propose CoDe-LoRA, a dual-branch continual adaptation framework that balances shared representation consolidation and task-specific adaptation. Extensive experiments demonstrate strong continual learning performance together with robust forward and backward transfer across diverse task sequences and model families.

Limitations

Although CoDe-LoRA demonstrates strong performance across both classification and generative

continual learning benchmarks, our evaluation is limited to pre-trained language models. Extending the framework to multi-modal continual learning settings remains an important direction for future work. In addition, the effectiveness of the routing mechanism depends on the quality of task prototypes in the embedding space. Under severe task distribution shifts or highly overlapping task boundaries, routing reliability may degrade and require more adaptive prototype estimation strategies.

Acknowledgements

This work was supported by Beijing Natural Science Foundation (JQ24019). This work was supported by the National Natural Science Foundation of China (No. 62576047). This work is being sponsored by CIPS-SMP-Zhipu Large Model Fund (No. CIPS-SMP20250313) and supported by the Open Fund of the Key Laboratory for Civil Aviation Collaborative Air Traffic Management Technology and Applications (No. 2025-001).

References

- Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. 2021. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385.
- Cyprien de Masson D’Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. 2019. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems*, 32.
- Wenyu Du, Shuang Cheng, Tongxu Luo, Zihan Qiu, Zeyu Huang, Ka Chun Cheung, Reynold Cheng, and Jie Fu. 2024. Unlocking continual learning abilities in language models. In *Conference on Empirical Methods in Natural Language Processing*.
- Jinghan He, Haiyun Guo, Kuan Zhu, Zihan Zhao, Ming Tang, and Jinqiao Wang. 2024. Seekr: Selective attention-guided knowledge retention for continual learning of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3254–3266.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang,

- and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2023. [Lorahub: Efficient cross-task generalization via dynamic lora composition](#). *Preprint*, arXiv:2307.13269.
- James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. [Overcoming catastrophic forgetting in neural networks](#). *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.
- Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.
- Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.
- Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947.
- David Lopez-Paz and Marc’Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.
- Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.
- Yuheng Lu, Bingshuo Qian, Caixia Yuan, Huixing Jiang, and Xiaojie Wang. 2025. Controlled low-rank adaptation with subspace regularization for continued training on large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19165–19181.
- Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150.
- Chengwei Qin and Shafiq Joty. 2021. Lfpt5: A unified framework for lifelong few-shot language learning based on prompt tuning of t5. *arXiv preprint arXiv:2110.07298*.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(1).
- Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madihan Khabisa, Mike Lewis, and Amjad Almahairi. 2023. Progressive prompts: Continual learning for language models. *arXiv preprint arXiv:2301.12314*.
- Gobinda Saha, Isha Garg, and Kaushik Roy. 2021. Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762*.
- Qwen Team. 2026. [Qwen3.5: Accelerating productivity with native multimodal agents](#).
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. [Llama 2: Open foundation and fine-tuned chat models](#). *arXiv preprint arXiv:2307.09288*.
- Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. [GLUE: A multi-task benchmark and analysis platform for natural language understanding](#). In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355. Association for Computational Linguistics.
- Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. 2023a. Orthogonal subspace learning for language model continual learning. *arXiv preprint arXiv:2310.14152*.
- Xiao Wang, Yuansen Zhang, Tianze Chen, Songyang Gao, Senjie Jin, Xianjun Yang, Zhiheng Xi, Rui Zheng, Yicheng Zou, Tao Gui, et al. 2023b. Trace: A comprehensive benchmark for continual learning in large language models. *arXiv preprint arXiv:2310.06762*.
- Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. 2022. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 139–149.
- Zitao Wang, Xinyi Wang, and Wei Hu. 2025. [Mixture of LoRA experts for continual information extraction with LLMs](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 13324–13339. Association for Computational Linguistics.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shuo Yang, Kun-Peng Ning, Yu-Yang Liu, Jia-Yu Yao, Yong-Hong Tian, Yi-Bing Song, and Li Yuan. 2025b. [Is parameter collision hindering continual learning in LLMs?](#) In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4243–4259. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

A Appendix

A.1 Notation

We summarize the key notations used throughout this paper in Table 4.

Notation	Description
$\mathcal{T}^{(t)}$	The t -th task in the sequence
$\mathcal{D}^{(t)}$	Dataset associated with task $\mathcal{T}^{(t)}$
θ_{base}	Parameters of the frozen backbone
θ_{shared}	Parameters of consolidation branch
θ_{spec}^k	Parameters of k -th task expert
$W_{\text{acc}}^{(t)}$	Accumulated shared weights up to task t
$\Delta W_{\text{shared}}^{(t)}$	Consolidation branch update at task t
$\Delta W_{\text{null}}^{(t)}$	Null space projected update component
c_t, s_t	Dynamic scaling coefficients at task t
P_k	Semantic prototype for task k
e_x	Embedding of input x from backbone
τ	Fallback confidence threshold

Table 4: Summary of key notations.

A.2 Theoretical Justification for Dynamic Scaling

In the model merging process, the dynamic time-varying scaling coefficients are introduced to maintain the deviation magnitude between the merged model and the pre-trained model.

General Formulation of Scaling. For clarity, we denote the parameter deviation at step t as $D_t = W_{\text{acc}}^{(t)} - W^{(0)}$, the null-space projected update component as $V_t = \Delta W_{\text{null}}^{(t)} = P_{\text{null}} \Delta W_{\text{shared}}^{(t)}$, and the running average norm of task updates as $\mu_t = \frac{1}{t} \sum_{i=1}^t \|\Delta W_{\text{shared}}^{(i)}\|_F$. In our study, the generalized scaling factor λ_t is defined as the inverse

of the scaling coefficient s_t (i.e., $\lambda_t = 1/s_t$), and can be computed as:

$$\lambda_t = \lambda_{t-1} \cdot \frac{\|D_{t-1} + V_t\|_F}{\mu_t} \quad (9)$$

where P_{null} denotes the null space projection operator.

Derivation of Scaling Coefficients.

Proof of Proposition 1. To simplify notation in this proof, we denote the consolidated weight matrix $W_{\text{acc}}^{(t)}$ as W_t , and the null-space projected update component $\Delta W_{\text{null}}^{(t)}$ as V_t . Recall that we assume $W^{(0)} = W_0 = 0$. Under the update rule defined in Eq. (5), the consolidated weight matrix at task t is updated recursively as:

$$W_t = c_t W_{t-1} + s_t V_t \quad (10)$$

where the scaling coefficients $c_t = \sqrt{(t-1)/t}$ and $s_t = 1/\sqrt{t}$ satisfy the normalization identity $c_t^2 + s_t^2 = 1$. By definition, the projected component V_t lies in the null space of the accumulated representations W_{t-1} , which implies they are orthogonal under the Frobenius inner product:

$$\langle W_{t-1}, V_t \rangle_F = 0 \quad (11)$$

Using this orthogonality, and given the boundary condition $W_0 = 0$, we have $\|W_t - W_0\|_F^2 = \|W_t\|_F^2$. Applying the Pythagorean theorem to the update rule in Eq. (10) yields:

$$\|W_t\|_F^2 = c_t^2 \|W_{t-1}\|_F^2 + s_t^2 \|V_t\|_F^2 \quad (12)$$

Let $M_t = \max_{i \in [1, t]} \|\Delta W_{\text{shared}}^{(i)}\|_F^2$. We prove that $\|W_t\|_F^2 \leq M_t$ by induction on t .

For the base case $t = 1$, since $c_1 = 0$ and $s_1 = 1$, we have:

$$\|W_1\|_F^2 = \|V_1\|_F^2 \leq M_1 \quad (13)$$

where the inequality holds because the projection operator is a contraction mapping, i.e., $\|V_t\|_F \leq \|\Delta W_{\text{shared}}^{(t)}\|_F$.

Now, assume that the bound holds up to task $t-1$, i.e., $\|W_{t-1}\|_F^2 \leq M_{t-1}$. For task t , substituting the induction hypothesis and the contraction property into Eq. (12):

$$\|W_t\|_F^2 \leq c_t^2 M_{t-1} + s_t^2 \|\Delta W_{\text{shared}}^{(t)}\|_F^2 \quad (14)$$

$$\leq c_t^2 M_t + s_t^2 M_t \quad (15)$$

$$= (c_t^2 + s_t^2) M_t = M_t \quad (16)$$

This completes the induction, showing that the Frobenius norm of the accumulated parameter deviation is bounded by the maximum individual task update norm:

$$\|W_{\text{acc}}^{(t)} - W^{(0)}\|_F^2 \leq M_t \quad (17)$$

This bound prevents the parameters of the consolidated branch from exploding or drifting away from the pre-trained manifold, mitigating catastrophic forgetting without sacrificing stability. $\square$

Remark on Knowledge Dilution. One might be concerned that the $s_t = 1/\sqrt{t}$ scaling in Eq. 4 dilutes the historical knowledge in the shared memory. We argue that this scaling acts as a **variance normalization** mechanism rather than information loss. Without scaling, the norm of the shared branch would grow linearly with $\sqrt{t}$ (assuming orthogonal updates), eventually overwhelming the task-specific branch. The $s_t = 1/\sqrt{t}$ factor ensures that the shared memory remains a stable “anchor” while the *specific* branch (Section 4.3) preserves the distinct decision boundaries for each task. This dual-branch design effectively decouples stability (via normalized shared memory) from plasticity (via preserved specific experts).

Empirical Verification of $1/\sqrt{t}$ Scaling. To empirically verify the stabilizing effect of our derived $1/\sqrt{t}$ scaling factor against standard linear averaging ($1/t$) or no scaling (1), we conduct a comparative analysis on the Standard CL Benchmark (Order-1, T5-Large). Table 5 reports the final average accuracy and the Frobenius norm of the accumulated parameters $\|W_{\text{acc}}^{(t)} - W^{(0)}\|_F$. Without scal-

Method	Scaling Factor	Avg Accuracy	Parameter Norm
No Scaling	1	77.4%	18.6 (Exploded)
Linear Avg	$1/t$	75.2%	1.3 (Diluted)
CoDe-LoRA	$1/\sqrt{t}$	80.8%	5.2 (Stable)

Table 5: Empirical verification of different scaling factors on the Standard CL Benchmark (Order-1, T5-Large). We report the final average accuracy and the Frobenius norm of the accumulated parameters $\|W_{\text{acc}}^{(t)} - W^{(0)}\|_F$.

ing, the parameter norm explodes, pushing weights outside the LLM’s effective optimization manifold. Conversely, linear averaging ($1/t$) overly dilutes the historical knowledge. Only our derived $1/\sqrt{t}$ factor maintains a stable norm and achieves the peak accuracy.

A.3 Qualitative and Prediction-Behavior Analyses

Figure 7 illustrates that the consolidation branch provides more reliable predictions on ambiguous samples where task-specific experts may fail. Table 6 further shows complementary prediction behaviors between the shared and task-specific branches during inference.

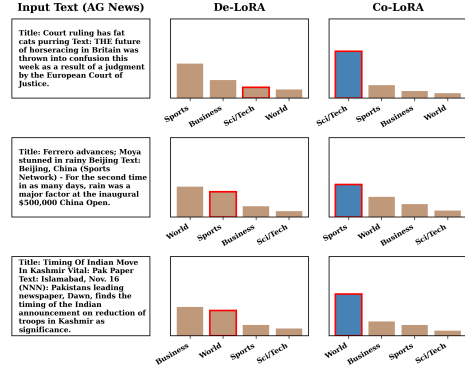


Figure 7: Visualization of the interaction between Co-LoRA and De-LoRA on ambiguous samples.

Case Type	Status	Percentage
Both Correct	$\theta_{\text{shared}} \checkmark, \theta_{\text{spec}} \checkmark$	79.0%
Only Consolidation	$\theta_{\text{shared}} \checkmark, \theta_{\text{spec}} \times$	0.6%
Only Decoupling	$\theta_{\text{shared}} \times, \theta_{\text{spec}} \checkmark$	1.9%
Both Incorrect	$\theta_{\text{shared}} \times, \theta_{\text{spec}} \times$	18.5%

Table 6: Synergy and correctness analysis between the consolidation and decoupling branches.

A.4 Implementation Details

Following existing CL works (Qin and Joty, 2021; Wang et al., 2023a; Yang et al., 2025b), all methods are implemented with instruction tuning (Qin and Joty, 2021) and optimized using AdamW (Loshchilov and Hutter, 2017) with a constant learning rate of 1.0×10^{-3} and a training batch size of 16. During inference, we use a generation batch size of 64 to avoid out-of-memory (OOM) errors. For all LoRA-based architectures, we set rank $r = 8$ and $\alpha = 32$, targeting the query and value projections in each Transformer block. For our prototype-based routing, 10 samples per task are used to construct semantic centroids, and the fallback threshold τ is set to 0.75.

To ensure fair comparisons and accurate runtime measurements, all experiments are executed on a single NVIDIA H800 (80 GB) GPU. This setup avoids potential inter-GPU communication latency

(e.g., DeepSpeed overhead) that could bias wall-clock time calculations. Furthermore, we disable the persistence of large intermediate shared parameter states to minimize I/O-related delays. Each experiment is repeated over three independent runs with different random seeds. The consolidation process is performed offline once per task with negligible overhead, using a projection rank fixed to 10% of the LoRA rank for efficiency.

Practical Implementation Details of SVD and Null Space Projection. To resolve the computational bottleneck of null space projection in large-scale LLMs, we adopt an optimized consolidation strategy. Specifically, we replace full Singular Value Decomposition (SVD) with a randomized low-rank SVD (`torch.svd_lowrank`) to efficiently extract singular vectors U from the column space of consolidated parameters. To further reduce complexity, the projection is performed directly on the low-rank factor $B \in \mathbb{R}^{d_{out} \times r}$ rather than the full update $\Delta W \in \mathbb{R}^{d_{out} \times d_{in}}$. The orthogonalized factor is computed as $B_{orth} = B - U(U^\top B)$, and the resulting update is $\Delta W_{null} = B_{orth}A$. This optimization reduces the projection matrix dimensions from $d_{out} \times d_{in}$ to $d_{out} \times r$, where $r \ll d_{in}$, achieving significant speedup while preserving mathematical accuracy. In practice, we set the number of randomized SVD iterations (`svd_niter`) to 1 to balance precision and performance.

Computational Complexity and SVD Efficiency Bridge. As introduced in Section 4.2, standard SVD on the full weight matrix update $\Delta W \in \mathbb{R}^{d_{out} \times d_{in}}$ requires $\mathcal{O}(d_{out}d_{in} \min(d_{out}, d_{in}))$ operations, which is prohibitively expensive for large language models where d_{out}, d_{in} can scale to thousands or tens of thousands. In contrast, by projecting directly on the low-rank factor $B \in \mathbb{R}^{d_{out} \times r}$ and leveraging the randomized low-rank SVD method (`torch.svd_lowrank`), the computational complexity is reduced to $\mathcal{O}(d_{out}r^2)$ where r is the LoRA rank (typically $r \in [8, 32]$). Because $r \ll d_{in}, d_{out}$, this optimization achieves a speedup of several orders of magnitude. In practice, the offline SVD consolidation takes less than 0.1 seconds per transformer layer, adding negligible overhead to the training pipeline while preserving the strict mathematical guarantee of null space projection. This provides a direct, highly efficient bridge between the theoretical null space projection in Section 4.2 and the practical scalability of our framework.

Sensitivity to Approximate Orthogonality.

While randomized SVD inevitably introduces approximation noise, CoDe-LoRA remains theoretically and architecturally robust to this relaxation. Theoretically, our dynamic scaling factor dampens noise accumulation, bounding the approximation error. Architecturally, the Decoupling Branch serves as a safety net; if approximation noise compromises the shared branch, routing dynamically isolates predictions to task-specific experts. To empirically verify this, we stress-test CoDe-LoRA by altering the SVD retained rank (simulating null-space estimation noise) on the Order-1 benchmark (T5-Large). Table 7 presents the final average accuracy under different retained ranks. Even when the SVD rank is aggressively

Retained SVD Rank	Order-1 Accuracy
Full SVD (Exact)	80.9%
Rank = 32 (Default)	80.8%
Rank = 4 (Noisy)	80.2%

Table 7: Sensitivity analysis of SVD retained rank on the Standard CL Benchmark (Order-1, T5-Large).

reduced to 4 (inducing severe projection noise), the performance drop is extremely small (-0.7pp). This demonstrates that CoDe-LoRA is highly robust to approximate orthogonality, allowing for efficient randomized low-rank computations without sacrificing performance.

Step-by-Step Training and Consolidation Workflow. During the learning of each sequential task $\mathcal{T}_t$, CoDe-LoRA decouples and consolidates knowledge through a structured step-by-step procedure:

- 1. Initialization:** The Consolidation Branch parameters θ_{shared} are initialized from the previously consolidated state $W_{acc}^{(t-1)}$ to enable immediate knowledge reuse and positive transfer, while a new task-specific expert $\theta_{spec}^{(t)}$ is initialized to zero.
- 2. Joint Training:** Both branches are optimized jointly on the current task dataset $\mathcal{D}^{(t)}$ using instruction tuning. This joint optimization allows the Consolidation Branch to assimilate shared representation updates while the Decoupling Branch captures task-specific nuances.
- 3. Null Space Consolidation:** Post-training, the parameter update learned in the Consolidation

Branch, denoted as $\Delta W_{\text{shared}}^{(t)} = W_{\text{shared}}^{(t)} - W_{\text{acc}}^{(t-1)}$, is extracted. We project this update onto the null space of the accumulated representation history $W_{\text{acc}}^{(t-1)}$ using randomized low-rank SVD (as detailed above) to eliminate representational interference.

4. **Dynamic Scaling & Archive:** The projected update is scaled via our dynamic scaling coefficient $s_t = 1/\sqrt{t}$ to prevent parameter norm explosion, and merged into the consolidated weight matrix $W_{\text{acc}}^{(t)}$ recursively using $c_t = \sqrt{(t-1)/t}$. Concurrently, the trained task-specific expert $\theta_{\text{spec}}^{(t)}$ is archived in the expert pool, and its prototype P_t is computed and stored.

Evaluation Protocol and Analysis Setup. For text classification tasks (e.g., Yahoo, DBpedia), candidate labels are obtained from task metadata and tokenized. While the main performance results are reported using the `generate()` method, the synergy analysis in Figure 7 utilizes a logit-based approach via `forward()` to extract exact probabilities, enabling the confidence-based visualization of knowledge consolidation. For the ablation studies and prototype sensitivity analysis in Figure 5, we vary the number of support samples N to find the optimal balance between routing performance and inference speed.

A.5 Evaluation Metrics

We evaluate sequential learning performance using two standard metrics in continual learning:

Average Performance (AP) represents the overall model capacity across all learned tasks after completing the continual learning sequence. For a sequence of T tasks, it is defined as:

$$\text{AP} = \frac{1}{T} \sum_{i=1}^T a_{i,T} \quad (18)$$

where $a_{i,T}$ denotes the evaluation performance (accuracy for classification benchmarks, or task-specific generative metrics like ROUGE/Exact Match for generative benchmarks) on task i after training on the final task T . In classification benchmarks, this corresponds to the standard Average Accuracy (A_T), while in generative benchmarks (such as TRACE), it is referred to as Overall Performance (OP).

Backward Transfer (BWT) (Lopez-Paz and Ranzato, 2017) measures the influence of learning subsequent tasks on the performance of previously acquired knowledge:

$$\text{BWT} = \frac{1}{T-1} \sum_{i=1}^{T-1} (a_{i,T} - a_{i,i}) \quad (19)$$

where $a_{i,i}$ is the performance on task i immediately after training on task i . A negative BWT value indicates catastrophic forgetting (performance degradation), whereas a value closer to 0 or positive indicates stable knowledge retention or positive backward transfer (where learning new tasks improves performance on older tasks). By employing BWT as our unified metric for backward retention across all benchmarks, we avoid the redundancy of using separate forgetting indicators for classification and generative tasks.

A.6 Sensitivity Analysis of Routing Threshold

The routing threshold τ is a critical hyperparameter that balances the use of specialized experts versus the consolidated shared branch. We conduct a sensitivity analysis on the Standard CL Benchmark (Order 1) by varying τ from 0.0 to 1.0. As shown in Figure 8, a very low threshold ($\tau \rightarrow 0$) leads to over-reliance on experts even for ambiguous inputs, while a very high threshold ($\tau \rightarrow 1$) forces the model to fall back to the shared branch too frequently, potentially missing out on specialized knowledge. We find that $\tau = 0.75$ provides the optimal balance, achieving a peak accuracy of 80.8% and maintaining robustness across tasks.

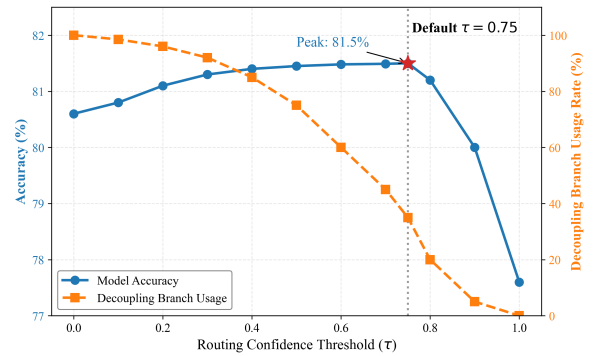


Figure 8: Sensitivity analysis of routing threshold τ on the Standard CL Benchmark (Order 1, T5-Large).

A.7 Stability and Routing Analysis across Task Orders

To further validate the robustness of CoDe-LoRA, we analyze its performance and routing accuracy

Method	Task	Params	Train Time (s)	Infer Time (s)	Peak Reserved
O-LoRA	Task 1: DBPedia	9.4M (1.26%)	2044.71	118.21	51016
	Task 2: Amazon	9.4M (1.26%)	943.97	211.05	49028
	Task 3: Yahoo	9.4M (1.26%)	2854.27	397.72	48794
	Task 4: AG News	9.4M (1.26%)	541.32	483.02	30858
N-LoRA	Task 1: DBPedia	9.4M (1.26%)	2775.82	84.94	25772
	Task 2: Amazon	9.4M (1.26%)	1216.32	189.60	25896
	Task 3: Yahoo	9.4M (1.26%)	3767.64	318.17	25896
	Task 4: AG News	9.4M (1.26%)	714.55	386.41	18520
CoDe-LoRA	Task 1: DBPedia	11.8M (1.58%)	1926.19	93.86	51024
	Task 2: Amazon	11.8M (1.58%)	851.90	201.98	48798
	Task 3: Yahoo	11.8M (1.58%)	2654.21	379.53	48240
	Task 4: AG News	11.8M (1.58%)	489.79	458.35	30906

Table 8: Detailed efficiency statistics for different methods on Order 1. “Train Time” and “Infer Time” denote the time spent on training and testing for each task, respectively. “Peak Reserved” indicates the maximum GPU memory usage recorded during each phase.

across three distinct task sequences. Figure 9 presents the backward transfer (retention) analysis and confusion matrices for these orders on the Standard CL Benchmark. The bar charts (a, c, e) compare the peak accuracy achieved at the end of each task’s training phase with the final accuracy measured after training on all subsequent tasks. Across all orders, the performance drop is negligible, confirming CoDe-LoRA’s effective mitigation of catastrophic forgetting.

Simultaneously, the confusion matrices (b, d, f) visualize the routing accuracy of our prototype-based mechanism. The high values along the diagonal indicate that the router consistently and accurately identifies the corresponding task-specific expert for each input sample. The stability of these results across different task sequences highlights the effectiveness of using semantic centroids for task-agnostic retrieval, ensuring that specialized knowledge is correctly mobilized during inference without significant interference from other tasks.

A.8 Detailed Efficiency Statistics

Table 8 provides the complete efficiency statistics, including training time, inference time, parameter counts, and peak memory usage for different methods. Peak memory usage remains manageable as only the selected expert and shared branch are active during inference. Training memory stays well within standard GPU capacities despite the dual-branch architecture.

Long Sequence Efficiency (T=15). To address concerns regarding storage scalability over a long sequence of tasks, we report the training time, inference testing time, and the end-to-end total param-

eter footprint after training on all 15 tasks of the Long Sequence Benchmark (Order 4, T5-Large) in Table 9. As shown, the end-to-end parameter

Method	Train Time (h)	Infer Time (s)	Total Params (MB)
O-LoRA	2.77	523.0	35.3
N-LoRA	2.54	520.4	35.3
CoDe-LoRA	2.36	483.5	37.7

Table 9: End-to-end efficiency and storage footprint on the Long Sequence Benchmark (T=15 tasks).

growth of CoDe-LoRA is extremely small, adding only an additional 2.4 MB over O-LoRA/N-LoRA after 15 tasks. Furthermore, because only one task-specific expert is activated during inference via semantic routing, our inference memory overhead remains constant and identical to a single-task LoRA model. This compares very favorably to replay methods; for instance, SEEKR consumes 76 MB to store even a minimal 1% data replay buffer, which is double the size of our entire parameter footprint (37.7 MB). Additionally, CoDe-LoRA trains faster than O-LoRA by avoiding expensive SVD updates on the backpropagation path.

A.9 Datasets

In our experiments, we utilize 15 datasets from standard classification and NLU benchmarks (as detailed in Table 10), as well as the generation-heavy TRACE benchmark (Wang et al., 2023b). Specifically, the classification datasets are sourced from well-established suites such as the CL benchmark (Zhang et al., 2015), GLUE (Wang et al., 2018), and SuperGLUE (Wang et al., 2019), as well as the IMDB movie review dataset (Maas et al., 2011). These cover natural language infer-

No.	Dataset name	Category	Task	Domain	Metric
1	Yelp	CL Benchmark	sentiment analysis	Yelp reviews	accuracy
2	Amazon	CL Benchmark	sentiment analysis	Amazon reviews	accuracy
3	DBpedia	CL Benchmark	topic classification	Wikipedia	accuracy
4	Yahoo	CL Benchmark	topic classification	Yahoo Q&A	accuracy
5	AG News	CL Benchmark	topic classification	news	accuracy
6	MNLI	GLUE	NLI	various	accuracy
7	QQP	GLUE	paragraph detection	Quora	accuracy
8	RTE	GLUE	NLI	news, Wikipedia	accuracy
9	SST-2	GLUE	sentiment analysis	movie reviews	accuracy
10	WiC	SuperGLUE	word sense disambiguation	lexical databases	accuracy
11	CB	SuperGLUE	NLI	various	accuracy
12	COPA	SuperGLUE	QA	blogs, encyclopedia	accuracy
13	BoolQA	SuperGLUE	boolean QA	Wikipedia	accuracy
14	MultiRC	SuperGLUE	QA	various	accuracy
15	IMDB	SuperGLUE	sentiment analysis	movie reviews	accuracy

Table 10: Datasets and their tasks, categories, domains, and metrics.

ence (NLI), sentiment classification (SC), and topic classification (TC).

Crucially, to evaluate generalization on more complex, generation-heavy tasks, we also incorporate the TRACE benchmark (Wang et al., 2023b). Unlike standard classification tasks, TRACE features multi-turn, complex generative tasks that assess the structural reasoning and domain adaptation of LLMs. In particular, we evaluate on code generation (completion) using the Py150 dataset, where even a single syntax error can lead to a compilation failure, making it highly fragile under sequential fine-tuning.

A.10 Task Sequence and Instructions

To evaluate the robustness of our method, we employ a diverse set of task instructions and evaluate across multiple task orders. Table 11 summarizes the specific natural language prompts used for each task category, following the instruction-tuning format described in Qin and Joty (2021). The task sequences for the T5 and Llama 2-7B experiments, together with the TRACE benchmark sequence, are provided in Table 12.

Task	Prompts
NLI	What is the logical relationship between the “sentence 1” and the “sentence 2”? Choose one from the option.
QQP	Whether the “first sentence” and the “second sentence” have the same meaning? Choose one from the option.
SC	What is the sentiment of the following paragraph? Choose one from the option.
TC	What is the topic of the following paragraph? Choose one from the option.
BoolQA	According to the following passage, is the question true or false? Choose one from the option.
MultiRC	According to the following passage and question, is the candidate answer true or false? Choose one from the option.
WiC	Given a word and two sentences, whether the word is used with the same sense in both sentences? Choose one from the option.

Table 11: Task descriptions and corresponding prompts.

Order	Setting	Task Sequence
1	T5, Llama 2-7B	dbpedia → amazon → yahoo → ag
2	T5, Llama 2-7B	dbpedia → amazon → ag → yahoo
3	T5, Llama 2-7B	yahoo → amazon → ag → dbpedia
4	T5	mnli → cb → wic → copa → qqp → boolqa → rte → imdb → yelp → amazon → sst-2 → dbpedia → ag → multirc → yahoo
5	T5	multirc → boolqa → wic → mnli → cb → copa → qqp → rte → imdb → sst-2 → dbpedia → ag → yelp → amazon → yahoo
6	T5	yelp → amazon → mnli → cb → copa → qqp → rte → imdb → sst-2 → dbpedia → ag → yahoo → multirc → boolqa → wic
1	TRACE	C-STANCE → FOMC → MeetingBank → Py150 → ScienceQA → NumGLUE-cm → NumGLUE-ds → 20Minutes

Table 12: Task sequences for different model and benchmark settings.

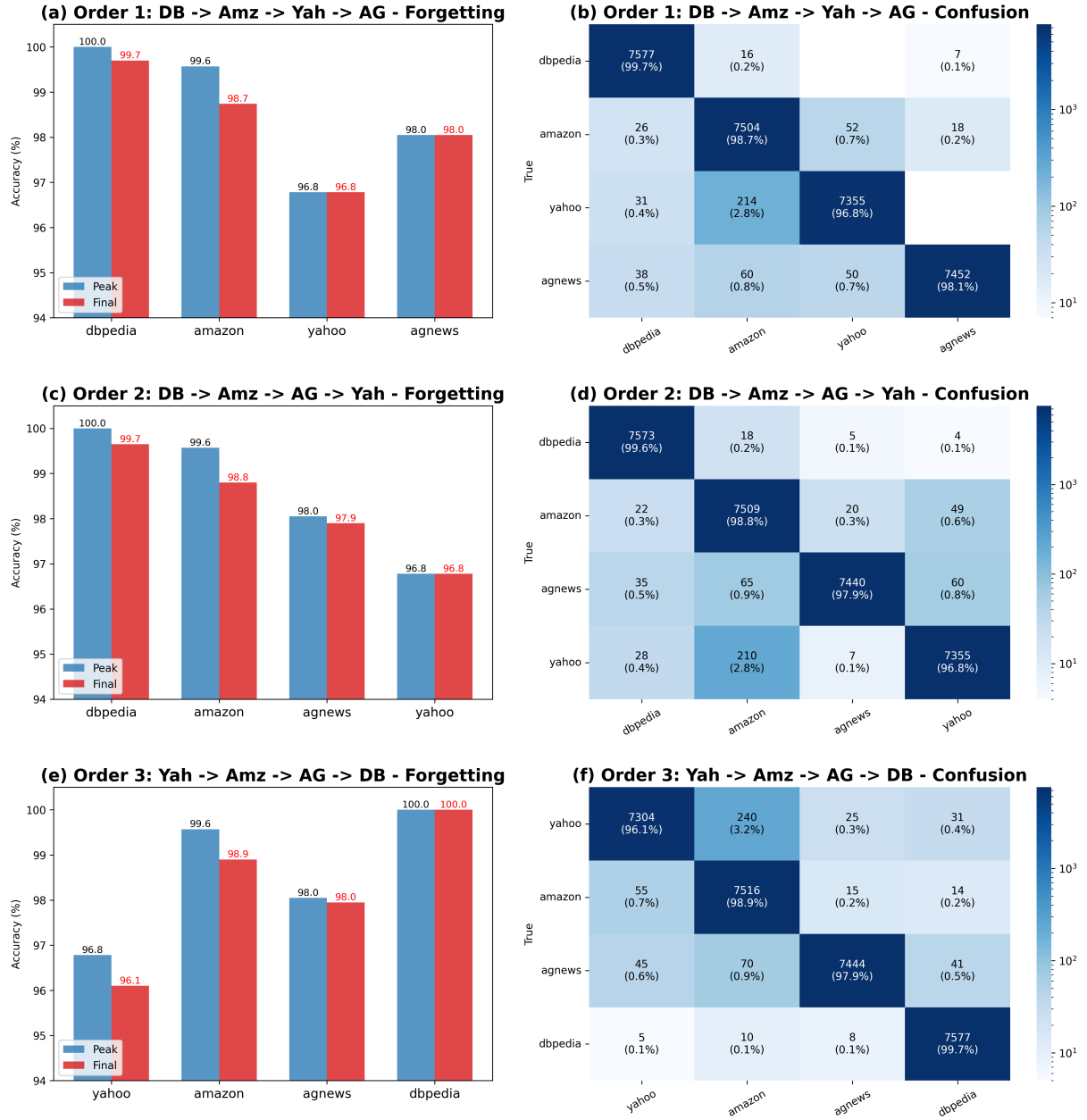


Figure 9: Stability and routing analysis across three task orders. (a, c, e) Forgetting analysis comparing peak vs. final accuracy for each task. (b, d, f) Confusion matrices of the prototype-based router, illustrating the frequency of expert selection for each input task across different sequences.