Static Unified Writeback is Not Enough: Input-Conditioned Correction for Continual LoRA

Anonymous ACL submission

Abstract

Continual LoRA learning is often framed as a choice of which low-rank factors to share online and which to keep task-specific. We argue that this framing misses an important deployment bottleneck in heterogeneous continual adaptation. On a six-task benchmark, fully task-specific LoRA is nearly lossless as a retention reference, and a task-aware direct-merge diagnostic is also nearly lossless as a unified-parameter reference. Yet strict no-task-id static adapters remain far worse, even under post-hoc SVD, alignment, and high-rank static sweeps. These results suggest that the issue is not merely the absence of shared structure or insufficient static rank, but the limited expressivity of a single input-agnostic unified writeback.

We first construct a task-aware teacher by decomposing the task-specific library into a shared trunk plus small residual corrections. The shared-residual adapter family recovers task-specific-reference validation loss with 54.17% of the original task-specific adapter parameters. We then instantiate the missing mechanism as a router-distilled minimal conditional correction student. The student predicts a soft correction mixture from multi-view input representations and applies the corresponding residual correction on top of the shared trunk, without explicit task identity. On the six-task full run, the best single seed improves the best high-rank static unified baseline from 1.520707 to 1.200605, and a three-seed run gives 1.210482 ± 0.012262, approaching the task-specific reference at 1.186326.

Native task metrics further support this picture. Across two additional backbones, Phi-4-mini and Falcon3-3B, the minimal conditional student improves average performance over a sequential single-LoRA baseline and reaches task-specific-reference-level average performance, with non-uniform per-task effects. Overall, these results identify a deployment bottleneck in continual LoRA: under strict no-task-id inference, reusable shared structure alone does not guarantee that heterogeneous local corrections can be collapsed into one input-agnostic writeback. In our six-task setting, minimal input-conditioned residual selection closes most of this static writeback gap.

1 Introduction

Low-rank adaptation makes large pretrained models easier to extend, but continual LoRA learning remains structurally difficult. A practical system must absorb heterogeneous tasks, avoid storing a full adapter per task, and run without explicit task identity when possible.

Most formulations attack this problem by deciding what to share online: share the down-projection factor A_t, share the up-projection/writeback factor B_t, share both, or protect one side with additional constraints. This line of work is useful, but it stays too close to a non-unique matrix factorization. For any invertible R, the effective update $\Delta W_t = B_t A_t$ can be rewritten as $(B_t R)(R^{-1} A_t)$. Factor-side statistics can change while the effective update remains the same. The more stable object is therefore the full correction path and its induced behavior, not either factor alone.

This ambiguity is closely related to the down-projection/up-projection view of LoRA. An adapter can be read as mapping activations into a low-dimensional correction coordinate system and then writing the correction back to the model dimension. Sharing one side of LoRA therefore assumes that heterogeneous tasks reuse either the same input-side coordinates or the same output-side writeback directions. However, the intermediate coordinate system is not identifiable: rotations, scalings, and other invertible changes of basis can alter the apparent geometry of the two factors without changing the effective update. Thus factor-side sharing is a useful implementation constraint, but a fragile object for diagnosing continual adaptation.

We instead treat the effective update and its input-conditioned correction behavior as the object of study.

Our experiments expose a sharper picture. Fully task-specific LoRA is nearly lossless in a six-task continual chain, and a task-aware direct-merge diagnostic of the resulting library is also nearly lossless. Yet strict no-task-id static unified baselines remain substantially degraded, even after SVD reconstruction, alignment, shared-atom unification, and high-rank sweeps. By *static unified writeback*, we mean one fixed LoRA update applied to every input after continual adaptation. The bottleneck is therefore not the absence of a unified parameter path, but the inability of one input-agnostic writeback to express heterogeneous local corrections.

At the same time, the task-specific library is highly redundant. A post-hoc shared trunk plus small residual corrections recovers task-specific-reference average loss with roughly half the adapter-family parameter count. Thus permanent task isolation is also too strong: shared structure exists, but removing all conditional correction is premature.

This leads to the central question: what minimal non-uniformity is required beyond a shared trunk? Our answer is *minimal conditional correction*: build a task-aware shared-residual teacher, then distill a no-task-id student that predicts a soft residual mixture from the input.

The resulting student reduces six-task validation loss from the best static unified baseline 1.520707 to 1.200605, close to the task-specific reference 1.186326. Across two additional backbones, native task metrics also improve over sequential single-LoRA and reach task-specific-reference-level average performance. We use this student as a constructive witness for a narrower mechanism: it tests whether the missing degree of freedom is input-conditioned residual selection, rather than claiming to outperform the full design space of MoE-style LoRA routers or to optimize serving cost.

We make three empirical contributions. First, we separate shared-structure existence from strict static deployment: direct diagnostic reconstruction and shared-residual consolidation show that reusable LoRA structure exists, while static no-task-id writebacks remain substantially worse. Second, we show that the task-specific library can be reorganized into a shared trunk plus small residual corrections, recovering task-specific-reference loss with 54.17% of the adapter-family parameters. Third, we instantiate the missing non-uniformity as a minimal no-explicit-task-ID residual selector. The router is not claimed to discover task-free atoms; rather, it demonstrates that input-conditioned local correction can replace explicit task-aware residual selection and close most of the static writeback gap. Throughout the paper, no-task-id refers to the absence of an externally supplied task label at inference, not to the absence of task information in the input distribution.

2 Related Work

2.1 PEFT and LoRA update geometry

Parameter-efficient fine-tuning adapts pretrained models by training a small set of parameters while keeping most backbone weights fixed. Adapter tuning and AdapterFusion introduced compact task modules and non-destructive composition [1, 2], while prompt tuning, prefix tuning, BitFit, and IA3 showed that small trainable subsets can steer large models [3–6]. LoRA represents adaptation as low-rank updates injected into Transformer weights [7], and AdaLoRA and QLoRA study rank allocation and memory-efficient low-rank finetuning [8, 9]. These methods mainly address single-task fitting or fixed-budget adaptation; our setting asks whether, after heterogeneous continual adaptation, one input-agnostic low-rank writeback is sufficient.

A complementary view treats LoRA as a low-dimensional correction path: the down-projection A_t defines compressed adaptation coordinates, while the up-projection/writeback B_t writes corrections back to model space. This explains why factor sharing can preserve reusable directions, but also exposes non-identifiability: different low-dimensional bases can represent the same effective update. Apparent factor similarity therefore need not imply behavioral similarity. We consequently move from factor-side sharing to effective-update diagnostics, asking whether correction paths can be served by one static writeback or must remain input-conditioned.

2.2 Parameter-efficient continual learning

Continual PEFT methods address the tension between sharing reusable knowledge and preserving task-specific behavior. Prompt-based methods such as L2P and Progressive Prompts learn task- or input-conditioned prompt components [10, 11]. O-LoRA reduces interference with approximately orthogonal low-rank subspaces [12], while SAPT

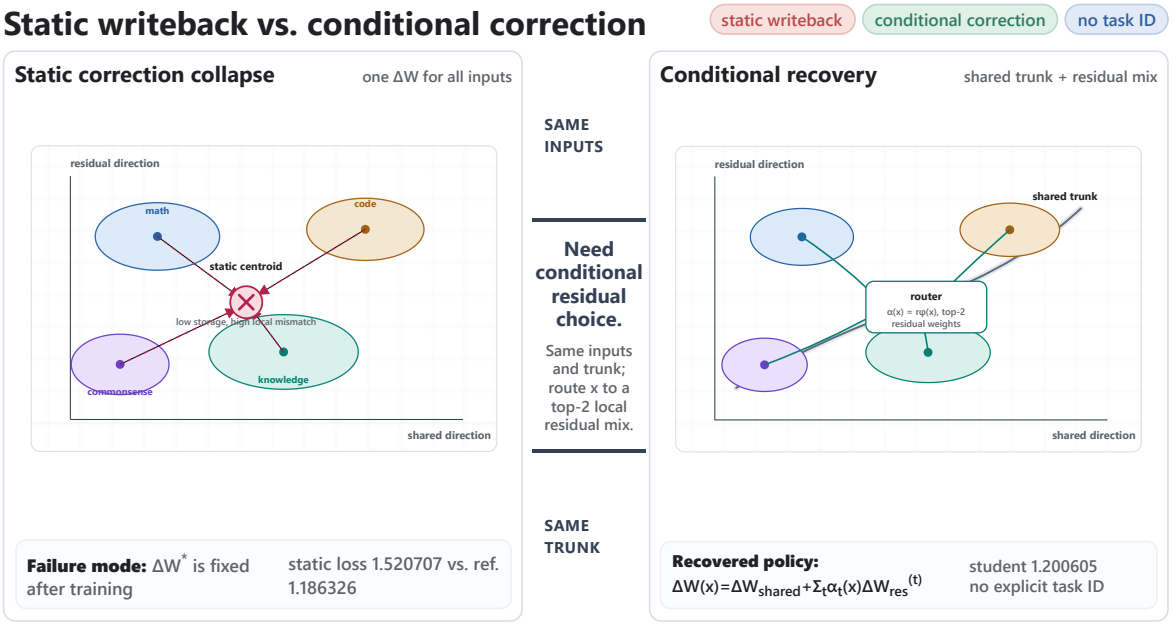

Figure 1: Core intuition. The left panel shows the static failure mode: heterogeneous local corrections are collapsed into one fixed update, leaving a loss gap. The right panel shows the minimal conditional alternative: a shared trunk keeps reusable structure, while an input-conditioned residual mixture restores local correction choice.

jointly learns and selects parameter-efficient modules through shared attention [13]. CL-LoRA and CONEC-LoRA explicitly separate task-shared and task-specific LoRA components [14, 15], and regularization-based variants protect shared low-rank updates with importance-aware constraints [16]. These works usually impose shared/specific structure during online training. We instead build a stable task-specific teacher library, then test whether it can be reorganized into a shared trunk plus the smallest conditional correction needed for no-task-id deployment.

2.3 LoRA composition, merging, and compression

Separate LoRA modules can also be composed, merged, served, or compressed after training. LoraHub dynamically composes existing LoRA modules for cross-task generalization without additional gradient updates [17]. Model soups show that independently finetuned solutions can sometimes be combined, while exposing the role of alignment [18]; KnOTS makes this issue explicit for LoRA merging and uses SVD to improve mergeability [19]. Systems work such as S-LoRA and Punica improves multi-LoRA serving, and Compress then Serve compresses many adapters into a shared basis with LoRA-specific coefficients [20–22]. These results support our premise that LoRA libraries can contain reusable shared structure. Our contribution is to separate this premise from the endpoint: compact shared structure does not imply that all residual variation can be collapsed into one static unified writeback.

2.4 Conditional LoRA routing and mixture experts

Routing-based PEFT treats LoRA modules or adapter components as conditional experts. Mixture-of-LoRA designs route examples or tokens to LoRA experts for multitask or instruction tuning [23, 24]; SLIM combines Soft LoRA with identity mixtures to reduce forgetting [25]; and D-MoLE routes among curriculum LoRA experts for continual multimodal instruction tuning [26]. These methods show that heterogeneous inputs can benefit from different adapter mixtures rather than one fixed update. Our goal is complementary to full MoE-style routing: given an already absorbed task-specific LoRA library, we ask whether its residual variation can be collapsed into one static writeback, or whether deployment must retain input-conditioned local choice. This isolates the central mechanism as the limited expressivity of a single static unified writeback, rather than simply insufficient rank, lack of shared structure, or poor LoRA alignment.

3 Problem Formulation

Let $\{\mathcal{T}_1, \ldots, \mathcal{T}_N\}$ be a sequence of heterogeneous tasks, and let each independently trained task LoRA produce an effective update ΔW_t. The task-specific absorption reference is

$$\mathcal{L}_{\text{ts-ref}} = \frac{1}{N} \sum_{t=1}^{N} L(\Delta W_t).$$

This reference is not the final deployment object; it is the protocol-matched target that preserves each task's local correction.

A static unified writeback is a single fixed task-agnostic adapter applied to every input, without task identity or input-dependent correction. If its average loss is $\mathcal{L}_{\text{static}}$, we define the static unified writeback gap as

$$\Delta_{\text{static}} = \mathcal{L}_{\text{static}} - \mathcal{L}_{\text{ts-ref}}.$$

Our direct-merge diagnostic shows that a high-quality unified parameter path can exist in this setting, but strict static baselines still leave a large Δ_{static}. The problem is therefore not simply whether parameters can be shared; it is whether one input-agnostic writeback can express the local corrections required by heterogeneous inputs.

No-task-ID does not mean task-free. It means that the inference procedure is not given an external task label and cannot select adapters by oracle task identity. The input itself may still reveal coarse domain structure, and a content router may use that structure. The claim is precisely that such input-conditioned correction is needed when static writeback removes all local choice.

Appendix A.1 summarizes the diagnostic artifacts and deployment endpoints used throughout the paper.

The minimally conditional alternative is

$$\Delta W(x) = \Delta W_{\text{shared}} + \sum_{k=1}^{K} \alpha_k(x)\, \Delta W_k,$$

where ΔW_{shared} is a reusable trunk, $\{\Delta W_k\}_{k=1}^{K}$ are residual correction atoms, and $\alpha_k(x)$ are input-conditioned coefficients. The scientific question is whether the deviations beyond a shared trunk can be recovered by such a small correction policy without explicit task identity.

4 Method: Minimal Conditional Correction

Figure 2 summarizes the pipeline. The teacher organizes task-specific LoRA updates into shared and residual components; the student removes explicit task identity by predicting a local correction mixture from the input.

4.1 Shared-residual teacher from task-specific LoRA libraries

We first train isolated LoRA adapters to obtain a stable teacher library $\{\Delta W_t\}_{t=1}^{N}$. This is not the final deployment solution; it preserves task-specific correction paths and exposes the structure to compress. From it, we construct a task-aware shared-residual teacher:

$$\Delta W_t \approx \Delta W_{\text{shared}} + \Delta W_{\text{res}}^{(t)}.$$

We perform the decomposition on materialized effective updates rather than on individual LoRA factors, because the low-rank factors are not identifiable up to invertible changes of basis. This makes the shared trunk a property of the correction path rather than of an arbitrary low-dimensional coordinate system. For each LoRA target layer, we first materialize the task-specific effective updates $\Delta_t = B_t A_t$. We then concatenate the task updates along the output dimension,

$$M = [\Delta_1; \Delta_2; \ldots; \Delta_N],$$

and take the top right singular vectors of M as the shared input-side basis A_{shared}. For each task, the corresponding shared writeback is obtained by projection,

$$\begin{aligned} B_{\text{shared}}^{(t)} &= \Delta_t A_{\text{shared}}^{\top}, \\ \widehat{\Delta}_{\text{shared}}^{(t)} &= B_{\text{shared}}^{(t)} A_{\text{shared}}. \end{aligned}$$

The remaining error is then factorized again:

$$R_t = \Delta_t - \widehat{\Delta}_{\text{shared}}^{(t)}, \qquad R_t \approx B_{\text{res}}^{(t)} A_{\text{res}}^{(t)}.$$

This task-aware decomposition measures how much reusable structure exists and how much residual correction must remain; it is not itself the deployed no-task-id method.

4.2 Prototype-conditioned teacher routing

Before training a student, we construct a no-task-id teacher routing signal. For each benchmark, we

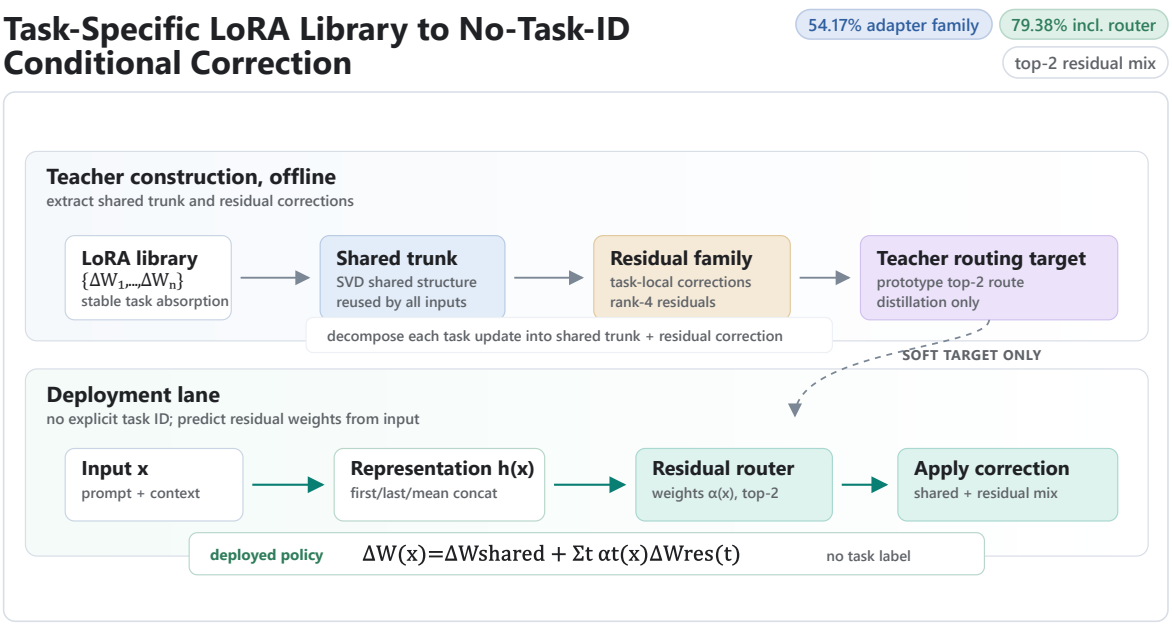

Figure 2: Overview of Minimal Conditional Correction. The upper lane constructs a task-aware teacher by decomposing a task-specific LoRA library into a shared trunk, residual family, and prototype-induced soft routing target. The lower lane deploys a strict no-task-id student that predicts the same conditional update form from the input.

sample training support payloads, extract frozen-backbone target-module representations, and build cosine k-means prototypes. Evaluation examples are not used. Given input x, similarities to all prototypes are converted into a sparse top-k soft routing distribution:

$$\Delta W_{\text{proto}}(x) = \Delta W_{\text{shared}} + \sum_{t=1}^{N} \pi_t(x)\, \Delta W_{\text{res}}^{(t)},$$

where $\pi_t(x)$ is induced by input-prototype similarity rather than explicit task identity.

This prototype teacher tests whether local input-conditioned correction is sufficient and supplies student targets.

4.3 Router-distilled no-task-id student

The deployable student removes prototype lookup at inference time. We train a router r_ϕ that maps multi-view input representations to a soft correction mixture:

$$\alpha(x) = r_\phi(h_{\text{first}}(x), h_{\text{last}}(x), h_{\text{mean}}(x)).$$

The student then applies

$$\Delta W_{\text{student}}(x) = \Delta W_{\text{shared}} + \sum_{t=1}^{N} \alpha_t(x)\, \Delta W_{\text{res}}^{(t)}.$$

The router is a small MLP trained by KL distillation to match the prototype teacher. At inference time, routing uses content, not a task label. The student still inherits a task-indexed residual family from the teacher, so the contribution is not end-to-end atom discovery; it is a closed instance showing that a learned input-conditioned correction selector can replace explicit task identity.

4.4 Deployment cost

Parameter ratio is computed directly from stored factors. The denominator is the full task-specific library $\sum_t(|A_t| + |B_t|)$. The shared-only numerator stores A_{shared} once per layer and $B_{\text{shared}}^{(t)}$ for each task. The shared-plus-residual numerator additionally stores $A_{\text{res}}^{(t)}$ and $B_{\text{res}}^{(t)}$. The deployed student also stores the router MLP, so we report adapter-family compression and total deployed student cost separately.

5 Experiments

5.1 Experimental design

The experiments are organized as a mechanism test rather than a benchmark sweep. We first establish a stable task-specific absorption reference, then ask whether the same task-specific library admits a high-quality unified parameter path through direct merge. This separates the existence of shared structure from the more restrictive deployment question: can all inputs be served by one static no-task-id

writeback? After showing that a shared trunk plus small residuals can recover the reference in a task-aware teacher, we test strict static endpoints under post-hoc SVD, alignment, shared-atom variants, and rank sweeps. The final step asks whether the missing capability can be supplied by minimal conditional correction without explicit task identity.

The six-task sequence spans code generation (MBPP), math reasoning (GSM8K), and knowledge or commonsense multiple-choice reasoning (ARC-Challenge, OpenBookQA, BoolQ, and PIQA). The main reference trains one LoRA checkpoint per benchmark and is used only as the protocol-matched absorption target. The sequential single-LoRA baseline trains one `pure_shared_ab` adapter through the full task order and evaluates the final checkpoint on all tasks. Static unified no-task-id baselines use one fixed adapter for every input, including mean-delta SVD, aligned core merge, shared-atom unification, and high-rank static variants.

5.2 Evaluation and implementation

We use token-level validation loss as the structural metric; in experiment artifacts this field is denoted `loss_only`. This isolates correction geometry from decoding choices and finite-sample generation variance. Native task metrics are therefore reported as a complementary behavioral sanity check rather than as a replacement for the controlled validation-loss diagnostic: pass@1 for MBPP, exact-answer accuracy for GSM8K, and ranked-choice accuracy for ARC-Challenge, OpenBookQA, BoolQ, and PIQA. The main six-task validation-loss result uses Qwen3-4B; the cross-backbone formal study repeats the comparison on `microsoft/Phi-4-mini-instruct` and `tiiuae/Falcon3-3B-Instruct`.

Unless stated otherwise, LoRA rank is 16, LoRA alpha is 32, dropout is 0.05, target modules are `q_proj` and `v_proj`, and max sequence length is 512. The final student uses an $8/4$ shared/residual rank split, `concat_first_last_mean` routing features, router hidden dimension 1024, support size 32, 4 prototypes per benchmark, teacher temperature 0.15, top-2 teacher and router mixing, 64 training examples per benchmark, 10 epochs, and batch size 16.

6 Results

6.1 Static writeback gap and shared-residual consolidation

We first ask whether the difficulty comes from the absence of a unified solution, or from the stricter requirement that the deployed update be static and input-agnostic. The results support the latter. The task-specific boundary condition is stable: in the 2×2 structural study, fully shared LoRA degrades under long continual training, single-sided sharing only partially helps, and fully task-specific LoRA is nearly lossless. We therefore use the independently trained task-specific library as the protocol-matched absorption reference.

Direct merge of the same library is also nearly lossless, reaching 1.186306 average loss, within -0.000020 of the task-specific reference at 1.186326. Direct merge should be interpreted as an existence proof for a unified parameter path under task-aware access, not as a deployable no-task-id endpoint. As summarized in Appendix A.1, this is a diagnostic artifact, not the strict deployment baseline: it is constructed with access to the task-specific library and tests whether a high-quality unified parameter path exists at all. The strict static baselines instead must use one fixed input-agnostic adapter for every example, with no task label, residual selection, or input-dependent correction. Thus direct merge rules out a lack of shared structure as the explanation, while Appendix A.4 tests the harder static no-task-id endpoint.

Table 1 shows that the task-specific library contains substantial reusable structure.

SHARED-ATOMS is compact but leaves a residual gap, while SHARED-ATOMS+RESIDUAL recovers task-specific-reference performance with 54.17% of the task-specific adapter-family parameters. The strict static baselines are detailed in Appendix A.4; all remain far from the task-specific reference, with the best high-rank static endpoint reaching only 1.520707. Together, Table 1 and the static baselines in Appendix A.4 isolate the endpoint rather than the storage budget. Compact shared structure is available, but collapsing all residual variation into one static writeback leaves a large gap.

6.2 Minimal conditional correction closes the validation-loss gap

The prototype teacher already reduces the gap substantially, indicating that local input-conditioned

Table 1: Six-task task-aware consolidation. Average validation loss, logged as `loss_only` in experiment artifacts, is reported across all six tasks. Lower is better. Parameter ratio is measured relative to the task-specific adapter library. This table evaluates task-aware consolidation artifacts and should not be confused with the strict no-task-id static deployment baselines in Appendix A.4. The ratio for TASK-AWARE ONLINE-SHARED is omitted because it is an online training diagnostic rather than a stored post-hoc adapter-family artifact.

Method	Avg. loss ↓	Δ vs. ref. ↓	Param. ratio
TASK-SPECIFIC	1.186326	0	1.0000
TASK-AWARE ONLINE-SHARED	1.187230	+0.000904	–
SHARED-ATOMS	1.189298	+0.002972	0.2917
SHARED-ATOMS+RESIDUAL	1.186242	-0.000084	0.5417

residual choice is sufficient to recover much of the missing behavior. The deployable student removes prototype lookup and predicts the residual mixture directly from the input. The router-distilled student reaches 1.200605 average loss, compared with 1.520707 for the best high-rank static unified baseline and 1.186326 for the task-specific reference.

Relative to the best static unified baseline, the student reduces average loss by 0.320102 and remains only 0.014279 from the task-specific reference. The important contrast is not only lower loss, but where the improvement appears: after restoring one missing degree of freedom, local residual choice. Since the shared trunk and residual family are fixed, the gap closure is attributed to input-dependent selection rather than additional backbone training or a larger static rank. The three-seed results are shown in Figure 3B and reported in Appendix A.5, giving 1.210482 ± 0.012262, so the main validation-loss result is not a best-seed artifact. Figure 3 summarizes the main trends; Appendix A.2 provides exact loss values, Appendix A.3 gives the parameter breakdown, and Appendix A.8 reports formal averages.

Appendix A.6 supports the final configuration: multi-view input features, top-2 routing, support size 32, and 4 prototypes per benchmark. The cheaper $6/2$ shared/residual setting already reaches 1.205542 with 34.38% of the task-specific adapter-family budget, close to the default $8/4$ setting at 1.200605.

6.3 Parameter cost and routing behavior

The deployed student uses 54.17% of the task-specific adapter-family parameters before the router and 79.38% including the router; Appendix A.3 reports the full breakdown. These costs are separated because they serve different purposes: the adapter family measures how much shared-residual structure is compressible, while the router tests whether input-conditioned residual selection recovers the gap left by a static writeback. The current student therefore prioritizes mechanism isolation over an optimized serving design.

The router is partly domain-separating, which is expected in a heterogeneous sequence where code, math, and multiple-choice reasoning induce visibly different corrections. It should therefore not be interpreted as discovering a fully task-free atom dictionary. On 64 train examples per benchmark, routing entropy is 0.265418, the top-1 route matches the source benchmark for 92.45% of examples, and average non-self residual mass is 0.173945. Off-diagonal mass concentrates among related tasks: ARC-Challenge routes 39.55% to OpenBookQA, OpenBookQA routes 34.15% to ARC-Challenge, and PIQA routes 21.06% to OpenBookQA. Appendix A.7 reports the full matrix. This supports the mechanism claim that the missing capacity is local residual choice from the input, not a single residual centroid.

This source alignment should not be interpreted as a failure of the no-task-id setting. No-task-id inference means that the deployment procedure is not given an external task label; it does not require the input distribution to be task-distribution-free. In a heterogeneous sequence, code, math, and multiple-choice reasoning naturally expose different input statistics, and a content router can exploit these statistics without oracle task identity. The important comparison is therefore not whether the router is perfectly task-agnostic, but whether restoring input-conditioned local choice closes the gap left by a single static writeback. The observed routing pattern supports this mechanism: some residuals are domain-aligned, while off-diagonal mass is concentrated among related reasoning benchmarks.

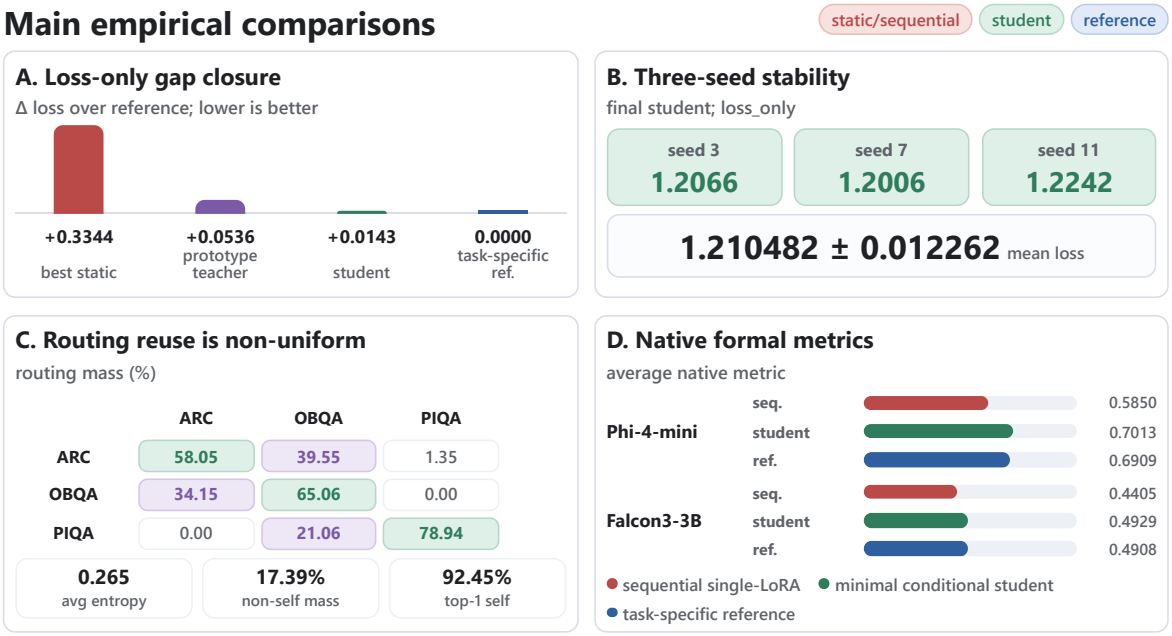

Figure 3: Main empirical summary. A: excess loss over the task-specific reference. B: three-seed student stability. C: routing is mostly source-aligned but shows limited reuse among related benchmarks. D: native formal metrics on two additional backbones.

6.4 Native formal metrics provide a task-level sanity check

Native task metrics provide a task-level check that the validation-loss conclusion also corresponds to benchmark behavior. The comparison uses the same protocol-matched reference: one independently trained task-specific LoRA per benchmark. The baseline is sequential single-LoRA, a single adapter trained through the full task order and evaluated at the final checkpoint. Figure 3D summarizes the average formal metrics, and Appendix A.8 reports the exact values. Across both Phi-4-mini and Falcon3-3B, the student improves over sequential single-LoRA and reaches task-specific-reference-level average performance: $0.5850 \rightarrow 0.7013$ on Phi-4-mini and $0.4405 \rightarrow 0.4929$ on Falcon3-3B. Because generation metrics have finite-sample variance and per-task effects are non-uniform, the slight positive Student–Ref differences should be interpreted as reference-level behavior, not as strict improvements over the task-specific reference.

Appendix A.9 shows non-uniform per-task effects: Phi-4-mini mainly recovers GSM8K, while Falcon3-3B gains on BoolQ, ARC-Challenge, OpenBookQA, and MBPP. The supported claim is average improvement with non-uniform behavior, not a guarantee on every benchmark.

7 Discussion and Conclusion

The central lesson is that heterogeneous continual LoRA can contain substantial shared structure while still requiring local correction choice at deployment. In the six-task setting, direct-merge diagnostics and task-aware consolidation show that unified or shared structure exists, but strict static writeback remains much worse. This separates the question of compressible shared parameters from the stricter endpoint question of whether every input can be served by one input-agnostic update. Under this protocol, the supported endpoint is a shared trunk plus input-conditioned residual choice.

8 Limitations

This work is a mechanism study rather than a general-purpose continual LoRA algorithm. First, the main six-task evidence is evaluated under one task order and uses token-level validation loss as the primary structural metric, while native task metrics are used as task-level sanity checks on two additional backbones. Second, the proposed student still stores a task-indexed residual family inherited from the task-specific teacher library; it should therefore not be interpreted as end-to-end task-free atom discovery. Third, although the shared-trunk-plus-residual adapter family uses 54.17% of the task-specific adapter parameters, the router in-

creases the total deployed ratio to 79.38%. Thus, the current student is intended as a minimal constructive witness for input-conditioned residual selection, not as an optimized compression or serving design. Reducing router cost, testing more task orders, and scaling to larger task libraries are left for future deployment-oriented work.

References

[1] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

[2] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rueckle, Kyunghyun Cho, and Iryna Gurevych. AdapterFusion: Non-Destructive Task Composition for Transfer Learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2021.

[3] Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[4] Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

[5] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

[6] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[7] 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 (ICLR)*, 2022.

[8] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. In *International Conference on Learning Representations (ICLR)*, 2023.

[9] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[10] 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 (CVPR)*, 2022.

[11] Anastasiia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, and Amjad Almahairi. Progressive Prompts: Continual Learning for Language Models. In *International Conference on Learning Representations (ICLR)*, 2023.

[12] Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. Orthogonal Subspace Learning for Language Model Continual Learning. In *Findings of the Association for Computational Linguistics: EMNLP*, 2023.

[13] Weixiang Zhao, Shilong Wang, Yulin Hu, Yanyan Zhao, Bing Qin, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. SAPT: A Shared Attention Framework for Parameter-Efficient Continual Learning of Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

[14] Jiangpeng He, Zhihao Duan, and Fengqing Zhu. CL-LoRA: Continual Low-Rank Adaptation for Rehearsal-Free Class-Incremental Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

[15] Naeem Paeedeh, Mahardhika Pratama, Weiping Ding, Jimmy Cao, Wolfgang Mayer, and Ryszard Kowalczyk. Continual Knowledge Consolidation LoRA for Domain Incremental Learning. arXiv preprint arXiv:2510.16077, 2025.

[16] Yaoyue Zheng, Yin Zhang, Joost van de Weijer, Gido M. van de Ven, Shaoyi Du, Xuetao Zhang, and Zhiqiang Tian. Revisiting Weight Regularization for Low-Rank Continual Learning. In *International Conference on Learning Representations (ICLR)*, 2026.

[17] Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. LoraHub: Efficient Cross-Task Generalization via Dynamic LoRA Composition. arXiv preprint arXiv:2307.13269, 2023.

[18] Mitchell Wortsman, Gabriel Ilharco, Samir Yitzhak 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 (ICML)*, 2022.

[19] George Stoica, Pratik Ramesh, Boglarka Ecsedi, Leshem Choshen, and Judy Hoffman. Model Merging with SVD to Tie the Knots. In *International Conference on Learning Representations (ICLR)*, 2025.

[20] Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. SLoRA: Scalable Serving of Thousands of LoRA Adapters. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

[21] Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-Tenant LoRA Serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024.

[22] 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 (ICML)*, 2025.

[23] Tongxu Luo, Jiahe Lei, Fangyu Lei, Weihao Liu, Shizhu He, Jun Zhao, and Kang Liu. MoELoRA: Contrastive Learning Guided Mixture of Experts on Parameter-Efficient Fine-Tuning for Large Language Models. arXiv preprint arXiv:2402.12851, 2024.

[24] Dengchun Li, Yingzi Ma, Naizheng Wang, Zhengmao Ye, Zhiyuan Cheng, Yinghao Tang, Yan Zhang, Lei Duan, Jie Zuo, Cal Yang, and Mingjie Tang. MixLoRA: Enhancing Large Language Models Fine-Tuning with LoRA-based Mixture of Experts. arXiv preprint arXiv:2404.15159, 2024.

[25] Jiayi Han, Liang Du, Hongwei Du, Xiangguo Zhou, Yiwen Wu, Yuanfang Zhang, Weibo Zheng, and Donghong Han. SLIM: Let LLM Learn More and Forget Less with Soft LoRA and Identity Mixture. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 4792–4804, 2025.

[26] Chendi Ge, Xin Wang, Zeyang Zhang, Hong Chen, Jiapei Fan, Longtao Huang, Hui Xue, and Wenwu Zhu. Dynamic Mixture of Curriculum LoRA Experts for Continual Multimodal Instruction Tuning. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pages 19011–19033, 2025.

A Additional Experimental Details

A.1 Artifact taxonomy

Table 2 separates diagnostic objects from deployment endpoints. This distinction is important for interpreting the direct-merge result: it tests whether a high-quality unified parameter path exists, but it is not the strict no-task-ID inference endpoint evaluated by the static baselines.

Table 2: Diagnostic artifacts and deployment endpoints. The main distinction is whether an object is task-aware evidence used to diagnose the library, or a strict no-task-ID endpoint used at inference.

Artifact	Task-aware	Input-cond.	Fixed inference adapter	Role
Task-specific library	Yes	No	No	Retention reference
Direct-merge diagnostic	Yes	No	N/A	Tests unified-path existence
Shared-residual teacher	Yes	Oracle	No	Tests reusable residual structure
Static unified baselines	No	No	Yes	Strict static endpoint
Router-distilled student	No	Yes	No	No-task-ID conditional endpoint

A.2 Evidence-chain summary

The main text compresses the diagnostic chain to stay within the page limit. Table 3 records the corresponding artifacts in one place. The key separation is between three objects: the protocol-matched task-specific absorption reference, the existence of a high-quality unified parameter path, and the strict static no-task-id deployment endpoint.

Table 3: Diagnostic evidence chain used by the main text. Lower validation loss is better.

Question	Artifact	Avg. loss ↓
Stable absorption reference	Task-specific LoRA library	1.186326
Unified parameter path exists?	Direct merge of task-specific library	1.186306
Compact task-aware consolidation?	Shared trunk + residual family	1.186242
Strict static no-task-id endpoint?	Best high-rank static unified	1.520707
Prototype conditional correction?	Prototype-conditioned teacher	1.239904
Input-conditioned correction?	Router-distilled minimal student	1.200605

This table should not be read as a single deployment comparison. The first three rows are diagnostic or task-aware artifacts that test whether shared structure exists; the fourth row is the strict static endpoint; the fifth row is a prototype-based conditional teacher; and the final row is the deployable no-task-id conditional student. The important contrast is that the task-specific library, direct merge, and shared-residual teacher all show that high-quality shared or unified structure is available, while the best strict static endpoint remains much worse. The student does not merely compress the library; it restores part of the missing local choice by predicting residual weights from the input.

A.3 Parameter cost breakdown

Table 4 reports the parameter accounting behind the main-text cost summary. The 54.17% row is the shared-trunk-plus-residual adapter family; the 79.38% row additionally includes the router MLP.

A.4 Strict static unified baselines

Table 5 expands the static no-task-id comparison from the main text. All rows use one input-agnostic adapter at inference time. These baselines are intentionally stricter than the task-aware shared-residual

Table 4: Parameter cost of the final minimal conditional student. Ratios are relative to the six-task task-specific LoRA library.

Component	Params	Ratio
Task-specific LoRA library	35,389,440	1.0000
Shared trunk only	10,321,920	0.2917
Shared trunk + residual family	19,169,280	0.5417
Router MLP	8,921,095	0.2521
Deployable student incl. router	28,090,375	0.7938

teacher: they do not route residual corrections by task or by input.

Table 5: Strict no-task-id static unified baselines.

Method	Avg. loss ↓
`pure_shared_ab`	1.509003
`mean_delta_svd`	1.521766
`aligned_core_merge`	1.520707
`shared_atom_unified`	1.526899

The static baselines are close to one another and all remain far from the task-specific reference. This is why the main text treats the static writeback gap as an endpoint failure rather than a failure of one particular merger. Alignment and SVD-style reconstruction can improve parameter organization, but under this protocol they do not provide the input-dependent residual choice needed by heterogeneous examples.

A.5 Final student seeds

Table 6 reports the three independent student runs used for the stability claim. The main text reports the same values compactly; the appendix keeps the run-level numbers visible next to the other diagnostic artifacts.

Table 6: Three-seed validation-loss stability for the final student.

Run	Avg. loss ↓
Seed 3	1.206636
Seed 7	1.200605
Seed 11	1.224206
Mean ± std	1.210482 ± 0.012262

The best seed is not used as the only evidence for the method. The mean remains close to the task-specific reference relative to the static baseline, and the standard deviation is small compared with the static gap. The three-seed table therefore supports the stability of the validation-loss conclusion, while not claiming that every seed is indistinguishable from the reference.

A.6 Full student ablations

Table 7 reports the complete minimal conditional student ablation grid used for the final student configuration. All runs use the six-task validation-loss protocol. Lower is better.

Table 7: Full minimal conditional student ablations.

Axis	Setting	Avg. loss ↓	Adapter-family ratio
Input view	first only	1.309122	0.5417
Input view	last only	1.309274	0.5417
Input view	mean only	1.239900	0.5417
Input view	first+last+mean	**1.200605**	0.5417
Router sparsity	dense	1.388069	0.5417
Router sparsity	top-1	1.810236	0.5417
Router sparsity	top-2	**1.200605**	0.5417
Router sparsity	top-4	1.220422	0.5417
Prototypes	$p = 1$	1.217147	0.5417
Prototypes	$p = 2$	1.214082	0.5417
Prototypes	$p = 4$	**1.200605**	0.5417
Support size	16	1.305614	0.5417
Support size	32	**1.200605**	0.5417
Support size	64	1.207660	0.5417
Ranks	shared 6 / residual 2	1.205542	0.3438
Ranks	shared 8 / residual 4	**1.200605**	0.5417
Ranks	shared 10 / residual 4	1.228955	0.6146
Ranks	shared 8 / residual 6	1.204301	0.6667

The ablations show three consistent patterns. First, the router benefits from using multiple representation views; mean-only is already useful, but concatenating first, last, and mean representations is strongest. Second, sparse routing matters: top-2 routing is much better than top-1 and dense mixing in this grid, suggesting that the correction policy needs limited choice rather than either hard isolation or uniform averaging. Third, the shared/residual rank split controls a real trade-off. The cheaper $6/2$ setting is close to the default while using a smaller adapter-family budget, whereas simply increasing ranks does not monotonically improve loss.

A.7 Routing usage matrix

Table 8 reports the average residual-domain mixture used by the final router on 64 training examples per benchmark. Rows are input benchmarks; columns are residual domains.

The matrix supports the conservative routing claim made in the main text. Some domains, such as MBPP and GSM8K, are routed almost entirely to their own residuals, which indicates that the router

Table 8: Routing usage matrix for the final student.

Input	MBPP	GSM8K	ARC-Challenge	OpenBookQA	BoolQ	PIQA
MBPP	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000
GSM8K	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000
ARC-Challenge	0.0000	0.0000	0.5805	0.3955	0.0048	0.0192
OpenBookQA	0.0000	0.0000	0.3415	0.6506	0.0000	0.0079
BoolQ	0.0000	0.0000	0.0642	0.0000	0.9358	0.0000
PIQA	0.0000	0.0000	0.0000	0.2106	0.0000	0.7894

is partly domain-separating. The off-diagonal mass is concentrated among related multiple-choice or commonsense tasks, especially ARC-Challenge, OpenBookQA, and PIQA. Thus the observed reuse is structured and limited; it should not be interpreted as uniform sharing across all tasks.

A.8 Cross-backbone formal averages

Table 9 gives the exact average formal metrics summarized visually in Figure 3D. The table is included in the appendix to keep the main EMNLP body within the page limit while preserving the numeric comparison behind the task-level sanity check.

Table 9: Cross-backbone formal task metrics. Values are averaged over six native metrics: MBPP pass@1 and accuracy for the other five benchmarks. Higher is better. Student–Ref differences should be interpreted as reference-level average behavior, not as a strict bound.

Backbone	Sequential	Student	Reference	Student–Seq.	Student–Ref.
Phi-4-mini	0.5850	0.7013	0.6909	+0.1163	+0.0104
Falcon3-3B	0.4405	0.4929	0.4908	+0.0524	+0.0022

The averages support the same conservative conclusion used in the main text: the student improves over the sequential single-LoRA endpoint and reaches reference-level average behavior on two additional backbones. They are not interpreted as strict improvements over the task-specific reference because native generation metrics have finite-sample variance and the per-task effects are non-uniform.

A.9 Per-task formal metrics

Table 10 reports the per-task native metrics for the cross-backbone comparison. The average formal result in the main text is computed over these six native metrics per backbone.

The averaged formal metric in the main text hides substantial per-task variation. Phi-4-mini gains primarily by recovering GSM8K after sequential overwrite, while Falcon3-3B gains most clearly on BoolQ, ARC-Challenge, OpenBookQA, and MBPP. Some tasks still favor the sequential adapter or the task-specific reference. This is why the main claim is average reference-level behavior with non-uniform per-task effects, not universal dominance on every benchmark.

Table 10: Per-task formal metrics for the cross-backbone comparison. Higher is better.

Backbone	Task	Metric	Sequential	Student	Reference	Student–Seq.
Phi-4-mini	mbpp	pass@1	0.0640	0.0520	0.0560	-0.0120
	gsm8k	acc.	0.1607	0.8446	0.7506	+0.6839
	arc_challenge	acc.	0.8242	0.8294	0.8311	+0.0051
	openbookqa	acc.	0.8200	0.8520	0.8480	+0.0320
	boolq	acc.	0.8379	0.8291	0.8483	-0.0089
	piqa	acc.	0.8030	0.8009	0.8118	-0.0022
Falcon3-3B	mbpp	pass@1	0.0340	0.0480	0.0420	+0.0140
	gsm8k	acc.	0.1835	0.1221	0.1365	-0.0614
	arc_challenge	acc.	0.6433	0.7039	0.6655	+0.0606
	openbookqa	acc.	0.6580	0.7000	0.7080	+0.0420
	boolq	acc.	0.4092	0.7459	0.7609	+0.3367
	piqa	acc.	0.7149	0.6377	0.6317	-0.0773

A.10 Compute resources

All reported runs use CUDA execution with bfloat16 model weights. The main experiments are single-backbone LoRA or router-distillation jobs over 3–4B-class instruction models with LoRA rank 16, max sequence length 512, and the batch sizes reported in Section 5; they are intended to run on a single 48GB-class GPU or an equivalent multi-GPU configuration. The experiment scripts emit per-run wall-clock timing fields such as `train_time_sec` and `eval_time_sec` in the JSON artifacts under `experiment/outputs`. These timings are included for the validation-loss structural runs, ablations, multi-seed student runs, and formal evaluation jobs, so the experiment artifacts provide run-level resource accounting rather than only aggregate claims. The compute accounting is deliberately reported at the run-artifact level because the paper combines structurally different jobs: isolated LoRA absorption, task-aware consolidation, router distillation, multi-seed reruns, and formal evaluation. A single aggregate GPU-hour number would obscure these differences. The JSON timing fields make it possible to audit each stage separately.

A.11 Reproducibility pointers

The main artifacts used by the final paper tables are stored under the experiment-output directory:

validation-loss ablations	`minimal_conditional_ablation_results.*`
validation-loss multi-seed	`minimal_conditional_multiseed_results.*`
parameter and routing	`qwen3-main-routing64-cost`
cross-backbone formal	`three_way_formal_results.*`

These pointers identify the artifacts used to construct the paper tables rather than an additional benchmark. They are included so that readers can

map each reported value back to a concrete run family and distinguish diagnostic teacher artifacts from the final deployable student.

A.12 Artifact licenses and intended use

This work uses publicly available research benchmarks, including MBPP, GSM8K, ARC-Challenge, OpenBookQA, BoolQ, and PIQA, and publicly released pretrained or instruction-tuned backbones, including Qwen3-4B, `microsoft/Phi-4-mini-instruct`, and `tiiuae/Falcon3-3B-Instruct`. These artifacts are used only for research evaluation of continual LoRA adaptation, and the paper does not redistribute the original datasets or pretrained model weights. Users of these artifacts should consult the original dataset and model providers for the applicable licenses, terms of use, and intended-use restrictions.