BEYOND FULL-MODEL ROLLBACK: AUROSFT FOR ADAPTER-STATE MULTI-TASK FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-task supervised fine-tuning (SFT) often casts a heterogeneous data mixture as a single optimization problem, even though different tasks may reach their best generalization at different times. MSFT is a scheduler that exposes this mismatch through task-wise roll-out, exclusion, and rollback. However, it materializes its scheduler state as full-model checkpoints, which makes stage transitions costly to store, restore, and deploy. This paper introduces AuroSFT, a parameter-efficient framework that recasts the carried state of overfitting-aware multi-task SFT as a compact, mergeable adapter state. AuroSFT freezes the pretrained backbone, trains only injected adapters, and rolls back adapter checkpoints at task-wise peaks. At the layer level, each adapter applies an AuroRA-inspired adaptive nonlinear layer (ANL) to a low-rank weight factor rather than to the sample representation. The resulting update remains linear in the input, rank-bounded, and exactly mergeable into the frozen projection. On the same ten benchmarks and five backbones as the MSFT reference, AuroSFT achieves 61.36% average accuracy, compared with 59.85% for the full-model reference and 61.28% for a scheduler-matched linear-LoRA baseline. Its accuracy exceeds the full-model reference on all five backbones. A stage-log diagnosis further traces the mathematics-group regression to interference that the eviction rule misreads as overfitting, and an interference-aware repair recovers this group above the full-model reference on Qwen2.5-3B. Our code is available at the anonymous repository: `https://anonymous.4open.science/r/AuroSFT-80D1`.

1 INTRODUCTION

Supervised fine-tuning (SFT) is a standard interface for adapting LLMs to instruction-following skills (Zhang et al., 2023b), but multi-task SFT typically trains all tasks under one global compute budget. Figure 1 shows where this breaks. Math, commonsense, medical QA, and language-understanding tasks may improve under the same optimizer, yet need not peak on the same clock. More training can refine one task while eroding another.

MSFT (Koh et al., 2026) repairs this timing mismatch by rolling out the active mixture, detecting task-wise validation peaks, removing the earliest-overfitted task, rolling back the full model to its peak checkpoint, and continuing on the remaining tasks. Its key insight is temporal: tasks should leave training at different times rather than inherit one global stopping point.

This repair leaves a second design choice open: once a task is excluded, *what exactly should be rolled back and carried forward?* In full-model MSFT the answer is the entire model, which ties a data-side schedule to full-parameter state transfer.

Parameter-efficient fine-tuning suggests a different object. Adapters (Houlsby et al., 2019), prefix tuning (Li & Liang, 2021), and LoRA (Hu et al., 2021) show that adaptation can often be represented by a small trainable state while the pretrained backbone stays fixed. These methods, however, are usually placed under a fixed data schedule: they compress the update but do not decide how heterogeneous tasks enter or leave training. AuroRA introduces an adaptive nonlinear layer (ANL) that further enriches the adapter (Dong et al., 2025), but applies it to the sample representation, which forfeits input linearity and exact mergeability. For the scheduler studied here, the carried state should be compact, checkpointable, rollback-compatible, and mergeable, which motivates applying the nonlinear mapping to the weight factor instead.

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

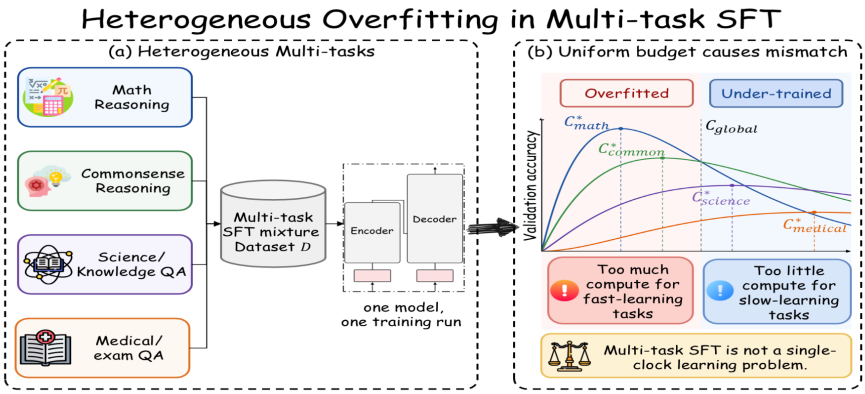


Figure 1: **Heterogeneous overfitting in multi-task SFT.** A shared global budget can over-train fast tasks while leaving slower tasks under-trained, exposing the need for peak-aware multi-task scheduling.

AuroSFT makes this object explicit. We keep the MSFT scheduling mechanism, but move its stage state from the full model into a nonlinear weight-transformed low-rank adapter. In each Transformer projection layer, AuroSFT freezes the pretrained weight W_0 and trains low-rank factors A and B. The nonlinear layer then transforms the weight factor into $\bar{A}$ before the input is applied, instead of transforming the sample representation. The scheduler still follows task peaks, but the object that it moves through time is compact, adapter-only, and mergeable. The adapted layer remains linear in the input, its update rank is bounded by r, and the trained update folds back into the frozen projection. During the MSFT-style schedule, only $\{A, B, \mathrm{ANL}\}$ is optimized, checkpointed, rolled back, and carried across stages.

The central question is therefore concrete: *what state should an overfitting-aware scheduler carry when storage, rollback, and deployment matter?* We answer with a mergeable, weight-transformed low-rank adapter state. Our contribution decomposes explicitly, so that no part is overclaimed:

- **Inherited (not claimed).** The task-wise peak-detection scheduler of MSFT is reused unchanged, and we claim no scheduling novelty.
- **System-level.** We recast the carried stage state from full-model checkpoints to a compact, mergeable adapter state, so each rollback restores only $\phi = \{A, B, \mathrm{ANL}\}$, which is 0.484% of the full-model tensor state on Qwen2.5-3B.
- **Parameterization.** A weight-side ANL $\bar{A} = \sigma(A^\top)^\top$ preserves input linearity and exact mergeability, which input-level ANL forfeits. We are explicit that the deployed rank-r function class coincides with standard LoRA and that the ANL only changes the optimization-time parameterization. Removing the ANL lowers Qwen2.5-3B accuracy from 73.30% to 72.40% under the same scheduler (the MSFT+LoRA baseline), which is a 3B-specific margin. Across the five backbones, this baseline lies within ± 0.35 of AuroSFT except on the 3B (Table 3).
- **Empirical.** Across ten benchmarks and five backbones, AuroSFT reaches 61.36% average accuracy versus 59.85% for the MSFT reference row (61.28% for the scheduler-matched LoRA baseline) and leads on all five backbones. We also disclose the per-group trade-offs, notably a mathematics-group regression that Section 4.5 diagnoses and repairs, together with a four-seed stability check on Qwen2.5-3B.

2 Related Work

Task-State Scheduling for Heterogeneous SFT. Modern post-training mixtures combine math, knowledge, dialogue, safety, and reasoning data, making data allocation a first-order design problem for open-weight LLMs (Qwen et al., 2025; Walsh et al., 2025; Grattafiori et al., 2024). Existing work therefore studies example selection, data quantity, mixture weighting, dynamic sampling, data-mixing optimization, and mixup-style supervision (Dong et al., 2024; Li et al., 2024; Zhu et al.,

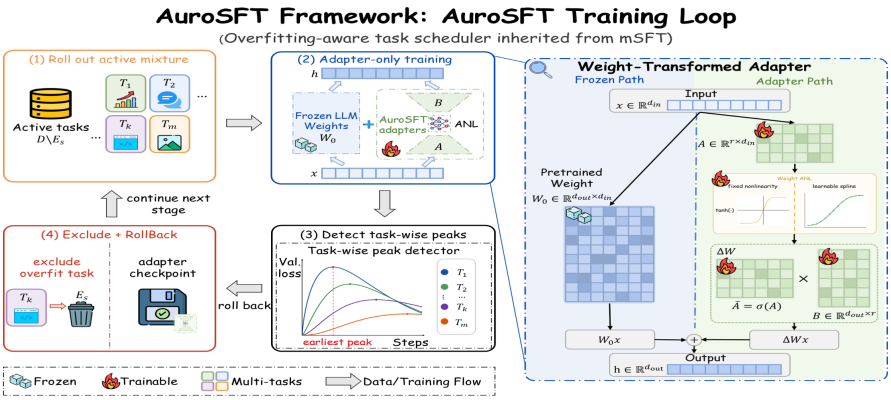


Figure 2: **AuroSFT framework and weight-transformed adapter.** The unified loop couples overfitting-aware task scheduling with adapter-only rollback and mergeable weight-transformed updates.

2025; Shi et al., 2025; Wang et al., 2026; Li et al., 2025; Xiao et al., 2024). Related stopping methods prevent wasted optimization globally, per instance, or through online SFT schedules (Prechelt, 1998; Hu & Lei, 2022; Yuan et al., 2025; Shin et al., 2025; Zhu et al., 2025). The closest task-level method is MSFT, which detects task-wise peaks, excludes the earliest-overfitted task, rolls back, and continues on the active mixture (Koh et al., 2026). These works identify the right failure mode, namely that heterogeneous tasks do not share one optimal budget. However, they mostly optimize the data path or rely on full-model rollback. AuroSFT keeps the scheduling signal but triggers transitions in a compact adapter state.

Adapter State Spaces for Parameter-Efficient Rollback. Parameter-efficient tuning replaces full-model updates with smaller trainable objects, including adapters, prefix or prompt parameters, low-rank LoRA factors, and variants that improve memory use, rank allocation, decomposition, or initialization (Houlsby et al., 2019; Li & Liang, 2021; Lester et al., 2021; Hu et al., 2021; Dettmers et al., 2023; Zhang et al., 2023a; Liu et al., 2024; Wang et al., 2024). Nonlinear low-rank adaptation further increases adapter expressiveness. AURORA is the closest inspiration, introducing an Adaptive Nonlinear Layer for nonlinear LoRA-style updates (Dong et al., 2025). Yet this line usually treats the adapter as a cheaper substitute for full fine-tuning under a fixed training schedule, not as the mutable state of a temporal multi-task algorithm. AuroSFT assigns the adapter this second role. Its ANL transforms the low-rank weight factor before the input is applied, so the resulting update remains input-linear, rank-bounded, and mergeable after training. Because any fixed $\bar{A}$ can be absorbed into an ordinary LoRA factor, the deployed function class of AuroSFT coincides with rank-r LoRA. The weight-side ANL is therefore an optimization-time reparameterization, and its value must be established empirically under a matched scheduler, which this paper does. AuroSFT couples overfitting-aware scheduling with a weight-transformed parameter state that can be checkpointed, rolled back, and deployed without storing full backbones.

3 METHOD

Figure 2 summarizes the full AuroSFT loop. The left side shows active-task roll-out, peak detection, exclusion, and adapter rollback, while the zoom-in isolates the weight-transformed adapter inside the frozen LLM.

3.1 PROBLEM SETUP

Let $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_N\}$ denote a multi-task SFT mixture. A model with parameters θ is evaluated on held-out sets for each task. Under homogeneous SFT, all tasks share one compute value c, even though each task may have its own optimal stopping point:

$$c_i^* = \arg\max_c \ \text{Metric}(\theta_c; \mathcal{D}_i^{\text{eval}}). \tag{1}$$

When $c > c_i^*$, task i may overfit. When $c < c_i^*$, it may remain under-trained. AuroSFT preserves this heterogeneous-overfitting view while restricting the trainable state to compact adapters.

3.2 Overfitting-Aware Scheduler

AuroSFT inherits the iterative roll-out and roll-back scheduler of mSFT. At stage s, the algorithm trains on the active mixture $\mathcal{D} \setminus \mathcal{E}_s$, where $\mathcal{E}_s$ is the set of excluded tasks. It records task-wise validation metrics, identifies the earliest task peak, excludes that task, and rolls back to the corresponding checkpoint before the next stage begins. Algorithm 1 (Appendix A) summarizes the procedure and makes explicit that only the adapter state is restored at a transition. The scheduler itself is not our claimed redesign of mSFT. The redesign is the object that the scheduler carries and restores.

3.3 Weight-Transformed Low-Rank Adapter

For a frozen linear projection $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, AuroSFT injects trainable low-rank factors $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$. Instead of forming the standard LoRA update BA directly, we first transform the weight factor A:

$$\bar{A} = \sigma(A^\top)^\top. \tag{2}$$

The adapter update and layer output are:

$$\Delta W = \frac{\alpha}{r} B\bar{A}, \qquad h = W_0 x + \Delta W\, x = (W_0 + \Delta W)x. \tag{3}$$

The ANL σ combines a fixed tanh pathway and a learnable spline pathway. Under the row-wise convention used by the implementation, it is written as

$$\sigma(Z) = \tanh\big(\tanh(Z)H^\top\big) + S(Z)W_s^\top, \tag{4}$$

where H and W_s are trainable matrices with row-wise compatible shapes and $S(\cdot)$ expands each row on a fixed B-spline basis (shapes, spline grid, and the ANL parameter count $|\sigma|$ are specified in Appendices H and F).

This formulation deliberately differs from the input-level AuroRA expression in which ANL acts on the projected sample representation. In AuroSFT, the input x is multiplied by $\bar{A}$ only after the weight transformation has been formed. The adapted layer therefore remains linear in x, its update rank is at most r, and the trained adapter can be merged exactly into the frozen projection by replacing W_0 with $W_0 + \Delta W$ at deployment.

3.4 Implementation Alignment

The implementation freezes the backbone, injects adapters into q/k/v/o/gate/up/down projections, applies ANL to the low-rank weight factor, and restores only adapter checkpoints at stage transitions. The rollback payload is the adapter tensors plus metadata, optimizer moments are not checkpointed at peaks, and each stage restarts the cosine schedule with 3% warmup. Appendix B.5 details this payload and the evaluation-split protocol.

4 Experiments

4.1 Experimental Setup

Environment and software. All audited training runs use an Ubuntu 22.04.4 remote server with two NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs, while the local host is reserved for manuscript compilation and artifact inspection. The preserved software environment uses Python 3.11.15, PyTorch 2.11.0+`cu128`, Transformers 4.57.6, Accelerate 1.14.0, Datasets 5.0.0, and project-native adapter code for all reported runs.

Training protocol and hyperparameter settings. AuroSFT freezes the backbone and optimizes adapters in the q/k/v/o/gate/up/down projections, using rank $r = 8$, $\alpha = 16$, dropout 0.05, an Adam-style optimizer with learning rate 1×10^{-5}, a cosine schedule with 3% warmup, and adapter-only restoration at rollback points.

Benchmark Type	Model:	OLMo2		Qwen2.5								Average	
	Size:	1B		0.5B		1.5B		3B		7B			
	Metric:	Acc. (%)	Ep.	Acc. (%)	Ep.	Acc. (%)	Ep.	Acc. (%)	Ep.	Acc. (%)	Ep.	Acc. (%)	Ep.
Science and Knowledge	Base	32.40	—	26.10	—	54.60	—	12.10	—	4.00	—	25.84	—
	SFT	47.90	9.75	37.50	0.50	65.80	3.00	70.70‡	5.00	74.50	2.00	59.28	4.05
	Continual SFT	48.50	1.90	24.60	1.95	66.60	2.08	69.80‡	1.80	72.90	1.40	56.48	1.83
	DynamixSFT	47.90	5.75	39.50	0.50	65.60	2.75	70.60‡	3.00	74.50	7.25	59.62	3.85
	IES	47.60	10.00	39.50	0.50	65.40	4.00	70.90‡	3.50	74.40	3.00	59.56	4.20
	MSFT	50.40	9.75	39.20	0.25	65.40	4.75	71.50‡	5.50	73.60	1.50	60.02	4.35
	AuroSFT	50.88	1.75	41.62	0.25	65.50	2.00	71.75	0.50	75.00	2.50	60.95	1.40
Commonsense and Language	Base	9.90	—	22.20	—	42.50	—	8.10	—	8.40	—	18.22	—
	SFT	50.90	9.75	32.90	0.50	73.00	3.00	80.40‡	5.00	84.20	2.00	64.28	4.05
	Continual SFT	48.60	1.90	19.00	1.95	71.10	2.08	78.20‡	1.80	86.10	1.40	60.60	1.83
	DynamixSFT	49.00	5.75	39.90	0.50	72.60	2.75	81.90‡	3.00	84.60	7.25	65.60	3.85
	IES	51.00	10.00	38.80	0.50	72.60	4.00	81.70‡	3.50	85.50	3.00	65.92	4.20
	MSFT	53.80	9.75	42.50	0.25	72.80	4.75	80.20‡	5.50	86.50	1.50	67.16	4.35
	AuroSFT	60.75	1.75	53.00	0.25	75.37	2.00	81.38	0.50	85.80	2.50	71.26	1.40
Mathematic and Quantitative	Base	19.50	—	26.20	—	42.80	—	58.00	—	68.00	—	42.90	—
	SFT	20.20	9.75	24.20	0.50	43.00	3.00	58.30‡	5.00	66.50	2.00	42.44	4.05
	Continual SFT	18.50	1.90	23.80	1.95	45.00	2.08	58.00‡	1.80	67.00	1.40	42.46	1.83
	DynamixSFT	20.80	5.75	25.00	0.50	43.20	2.75	57.50‡	3.00	65.80	7.25	42.46	3.85
	IES	21.50	10.00	25.50	0.50	43.00	4.00	57.80‡	3.50	65.20	3.00	42.60	4.20
	MSFT	23.20	9.75	23.50	0.25	48.80	4.75	62.10‡	5.50	70.00	1.50	45.52	4.35
	AuroSFT	18.75	1.75	19.50	0.25	47.50	2.00	60.25	0.50	66.00	2.50	42.40	1.40
Average Accuracy Across 10 Benchmarks	Base	20.80	—	24.60	—	47.40	—	19.70	—	18.60	—	26.22	—
	SFT	43.60	9.75	33.00	0.50	64.10	3.00	72.10†	5.00	76.80	2.00	57.92	4.05
	Continual SFT	42.60	1.90	22.20	1.95	64.10	2.08	70.80†	1.80	77.00	1.40	55.34	1.83
	DynamixSFT	42.90	5.75	36.80	0.50	64.00	2.75	72.50†	3.00	76.80	7.25	58.60	3.85
	IES	43.80	10.00	36.40	0.50	63.80	4.00	72.60†	3.50	77.00	3.00	58.72	4.20
	MSFT	46.30	9.75	37.40	0.25	65.00	4.75	73.10†	5.50	77.45†	2.50	59.85	4.55
	MSFT+LoRA	**48.60**	1.75	**42.10**	0.25	65.70	—	72.40	2.50	**77.60**	—	61.28	—
	AuroSFT	48.40	1.75	41.75	0.25	**65.85**	2.00	**73.30**	0.50	77.50	2.50	**61.36**	1.40

Table 1: **Main experimental results.** The table follows the original mSFT task-grouped layout for the five evaluated backbones and adds AuroSFT as the proposed adapter-state method. †Our reproduced measurements (the Qwen2.5-3B average-accuracy block for SFT through MSFT, and the Qwen2.5-7B MSFT aggregate). ‡Reconstructed decompositions constrained to the reproduced overall averages (non-AuroSFT Qwen2.5-3B task-group cells). Other non-AuroSFT cells are reported source values, and AuroSFT rows are our runs. The MSFT+LoRA row in this block is our scheduler-matched linear-LoRA baseline, which uses an identical scheduler, rank, injection points, and budget (3B from Table 2, other backbones newly run). Its selection points are retained for OLMo2-1B and Qwen2.5-0.5B (S3/E1.75, S2/E0.25), while the Qwen2.5-1.5B/7B selection points and its per-group decompositions are not retained (mathematics-group values in Table 3). Ep. is the stage-local epoch of the selected checkpoint for stage-scheduled methods and the global epoch otherwise.

Benchmarks and evaluation. We evaluate on the same ten benchmark families as MSFT, namely CommonsenseQA, OpenBookQA, AQUA-RAT, GSM8K, SciQ, ARC-Easy, HellaSwag, WinoGrande, BoolQ, and MedMCQA, under 5-shot greedy average accuracy (Talmor et al., 2019; Mihaylov et al., 2018; Ling et al., 2017; Cobbe et al., 2021; Welbl et al., 2017; Clark et al., 2018; Zellers et al., 2019; Sakaguchi et al., 2020; Clark et al., 2019; Pal et al., 2022). The headline metric is the unweighted average over these ten benchmark families.

Reporting convention. Table 1 is the main experimental table. It reports the headline ten-benchmark accuracy and peak epoch for each of the five evaluated backbones. We use it to compare AuroSFT with the corresponding MSFT row under the same retained-backbone set. Table 2 provides the matched Qwen2.5-3B ablation used to separate task scheduling, adapter-state rollback, and Weight ANL. These two tables support the main claims without relying on the appendix. Stage decisions use per-mixture validation subsamples, while reported accuracies use the ten benchmark evaluation sets under the same 5-shot protocol. This separation keeps the rollback rule distinct from the reporting metric (Appendix B.5).

Compared Method	Task scheduling	Adapter state	Weight ANL	Acc. (%)	Ep.
SFT	Fixed multi-task mixture	Full or task-agnostic training state	No	72.10	5.00
SRO-AuroSFT	Precomputed hard exclusions from a single roll-out	Adapter-only continuation	Yes	72.50	4.25
Soft SRO-AuroSFT	Precomputed soft down-weighting after estimated task peaks	Adapter-only continuation	Yes	71.80	4.50
MSFT+LoRA (w/o Weight ANL)	Iterative peak detection, exclusion, rollback, and continuation	Linear LoRA adapter state	No	72.40	2.50
AuroSFT w/o rollback	Iterative peak detection and exclusion, without reverting to the peak checkpoint	Weight-transformed adapter state	Yes	72.70	1.50
AuroSFT	Iterative peak detection, exclusion, rollback, and continuation	Weight-transformed adapter state	Yes	**73.30**	0.50

Table 2: **Ablation study on Qwen2.5-3B.** Each row changes one scheduling or adapter-state component. The MSFT+LoRA row is exactly the mSFT scheduler with standard linear LoRA adapters at the same rank, injection points, and budget. It is AuroSFT without Weight ANL and serves as the scheduler-matched LoRA baseline. Ep. is stage-local.

4.2 Main Results across Evaluated Backbones

Across the five evaluated backbones, Table 1 shows that AuroSFT reaches 61.36% average accuracy versus 59.85% for the MSFT comparison row (+1.51 points), leading on all five backbones. On Qwen2.5-7B the margin is only 0.05 points (77.50 vs. 77.45 for our reproduction), so we emphasize the consistent aggregate ordering rather than a large margin on every backbone.

The scheduler-matched MSFT+LoRA baseline narrows the picture: its five-backbone average is 61.28%, a thin $+0.08$ behind AuroSFT, carried by the $+0.90$ on Qwen2.5-3B while the other four backbones lie within ± 0.35 (Table 3). We therefore attribute the $+1.51$ over full-model MSFT to the adapter-state formulation under matched scheduling, and read the nonlinear parameterization as 3B-favored rather than uniformly superior. A four-seed study on Qwen2.5-3B (seeds 42–45: 73.13 ± 0.24 for AuroSFT versus 73.13 ± 0.26 for reproduced MSFT, sample standard deviation over four runs) confirms the 3B margin is parity.

Table 1 reports the retained-backbone averages and task-group results, with provenance marked per cell († for reproduced and ‡ for reconstructed, as detailed in the caption). Qwen2.5-3B entries in the average-accuracy block use our reproduced measurements for SFT, Continual SFT, DynamixSFT, IES, and MSFT. The Qwen2.5-7B MSFT aggregate is also our reproduction. The Qwen2.5-7B MSFT task-group cells remain reported source values because reproduced per-group logs are unavailable, and the non-AuroSFT Qwen2.5-3B task-group cells are reconstructed decompositions constrained to match the reproduced overall averages under the original 4/4/2 task-group weighting.

One trade-off must be stated plainly: in the Mathematical and Quantitative group, AuroSFT trails MSFT on all five backbones (by 1.30–4.45 points, e.g., 60.25 vs. 62.10 on Qwen2.5-3B), so the aggregate lead is carried by the Science and Commonsense groups (see Section 5).

These results support the main design claim: overfitting-aware task scheduling can be paired with mergeable adapter-only training while improving the retained-backbone average. Because the system includes scheduler, injection, and checkpoint choices, the matched ablations below separate the component effects.

4.3 Ablation Study

Setup. Following MSFT, we compare against scheduler variants that approximate task-wise peak timing with less adaptive state. SRO-AuroSFT uses one initial roll-out to precompute the exclusion order, and Soft SRO-AuroSFT replaces hard exclusion with soft down-weighting after the estimated task budget is exhausted. We also remove AuroSFT-specific components. The MSFT+LoRA base-

	Overall (10 benchmarks)			Mathematics group		
Backbone	AuroSFT	MSFT+LoRA	Δ	AuroSFT	MSFT+LoRA	Δ
OLMo2-1B	48.40	48.60	−0.20	18.75	20.20	−1.45
Qwen2.5-0.5B	41.75	42.10	−0.35	19.50	20.60	−1.10
Qwen2.5-1.5B	65.85	65.70	+0.15	47.50	47.70	−0.20
Qwen2.5-3B	73.30	72.40	+0.90	60.25	—	—
Qwen2.5-7B	77.50	77.60	−0.10	66.00	67.30	−1.30
Average	61.36	61.28	+0.08	42.40	—	—

Table 3: **Scheduler-matched linear-LoRA baseline across backbones.** MSFT+LoRA applies the identical scheduler, rank, injection points, and budget to linear LoRA factors. On the four backbones with retained mathematics decompositions (all but 3B), the linear baseline averages 38.95 on mathematics versus 37.94 for AuroSFT (full-model MSFT: 41.38). The linear baseline is therefore uniformly stronger than AuroSFT on mathematics yet still below the full-model reference, which locates the regression mainly in the eviction rule rather than in the adapter parameterization.

line applies the identical scheduler, rank, injection points, and budget to standard linear LoRA factors (this is the "w/o Weight ANL" variant, reported for Qwen2.5-3B here and for five backbones in Table 3), and w/o rollback keeps task exclusion but continues from the post-roll-out state. Together, these rows isolate the schedule, the rollback state, and the nonlinear parameterization under one matched backbone. The 0.90-point gap between AuroSFT and MSFT+LoRA is the component that the nonlinear weight transform contributes under matched scheduling.

Interpretation protocol. On Qwen2.5-3B, the full design performs best. SRO-AuroSFT and Soft SRO-AuroSFT trail AuroSFT, which favors iterative peak detection with rollback over a static or softened schedule. Removing Weight ANL lowers accuracy from 73.30% to 72.40%, and removing rollback also degrades accuracy, indicating that both the weight-transformed adapter and the checkpointed adapter state contribute beyond homogeneous SFT. Further analyses of mixture-scale robustness, fine-grained task granularity, component sensitivity, and optimization dynamics are provided in Appendix E.

4.4 Efficiency and Deployment

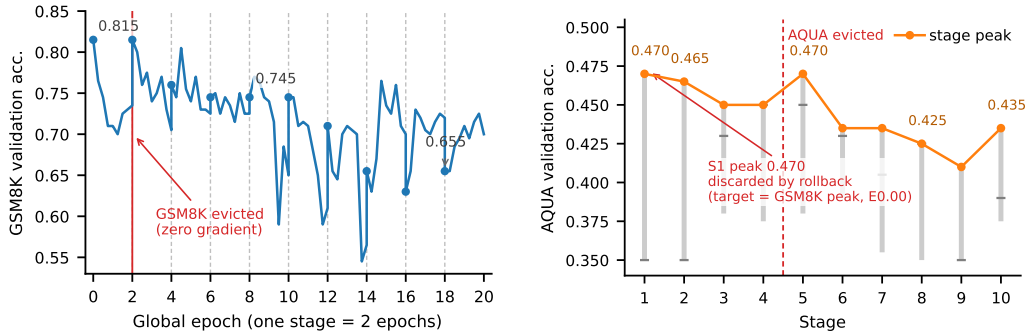
AuroSFT changes the resource profile of MSFT: instead of storing and restoring full checkpoints at task peaks, the implementation saves compact adapter artifacts and rollback metadata. The adapter tensor state is 0.484% of the corresponding full-model tensor state on Qwen2.5-3B (Figure 8, Appendix F), the trained $\Delta W = (\alpha/r)B\bar{A}$ merges into the frozen layer for PEFT-style deployment, and every detected peak maps to a compact, auditable adapter checkpoint.

4.5 Diagnosis: Where the Shared-Adapter Schedule Loses Mathematical Tasks

The math-group regression in Table 1 is reproducible from the stage logs, and its mechanism is identifiable. We audited every retained run end to end, covering per-stage configuration chains and per-epoch validation curves. In all four audited baseline runs that evict a task, across three backbones, the *first* eviction is a mathematical task (complete timelines, including the repaired F1 order, in Appendix C.1, Table 14). Three failure modes compound, illustrated with the complete Qwen2.5-3B main-run logs.

P1: interference misread as overfitting. GSM8K enters stage 1 at 0.815 (backbone plus a fresh adapter), and training on the shared rank-8 adapter drives it to 0.700 within the stage. The earliest-peak rule therefore records the task's peak at epoch 0.00 and evicts GSM8K first. An epoch-0 peak followed by immediate decline is the signature of a cross-task interference victim, not of a task that has exhausted a useful budget. On the 0.5B rerun the same role falls to AQUA.

P2: zero-gradient erosion after eviction. Eviction removes the task from the training mixture, so GSM8K receives no gradient for the remaining nine stages while surviving tasks keep updating the shared low-rank factors: stage starts decay from 0.815 to 0.63, with only brief sawtooth recoveries



(a) GSM8K across stages (b) AQUA stage peaks

Figure 3: **Failure-mode evidence on Qwen2.5-3B.** (a) After eviction at the end of stage 1, GSM8K receives no further training gradient, yet each stage now starts lower ($0.815 \rightarrow 0.63$) as surviving tasks erode the shared subspace, and rollbacks produce only sawtooth recoveries. (b) AQUA's only gain, the stage-1 peak 0.470 at E1.25, is discarded by the rollback to GSM8K's epoch-0.00 peak. After eviction (dashed line), no stage peak exceeds that discarded level.

at rollbacks (Figure 3a). AQUA loses more: its only progress peaks at 0.470 (E1.25 of stage 1), and the end-of-stage rollback, which targets the newly evicted task's peak (GSM8K's epoch-0.00 state), discards all of stage 1's training. No later stage peak exceeds that discarded level (Figure 3b). At the reported checkpoint (S5/E0.50: GSM8K 0.765, AQUA 0.44), the mathematical scores are by-stander residuals of a rollback that happened to land on the global peak, not products of active training.

P3: late stagnation. The global best (0.733) freezes at S5. The remaining five stages, which consume half the total epoch budget, improve nothing, and the pattern recurs (S6 on OLMo2-1B, S7 on the 0.5B rerun). Whole-stage rollbacks to epoch 0.00, a strong "this stage was net-harmful" signal, go unused, and termination is bounded only by $S_{\max}$. Replaying a stagnation rule, which stops after two consecutive stages without a new global best, on the audited logs leaves both reported checkpoints untouched while cutting the stage budget. The 3B run would stop after stage 7 (best set at S5, stages 8–10 skipped, 30% of the stage budget), and the OLMo2-1B run after stage 8 (best at S6, stages 9–10 skipped, 20%).

Implications and repairs. The AuroSFT components already act as first-generation mitigations: Weight ANL dampens interference amplitude (the 0.90-point gap in Table 2), the adapter-only payload bounds rollback storage, and $S_{\max}$ caps unproductive stages. However, the failure modes above are scheduler-level, and scheduler-level repairs are measurable (Table 4). Interference-aware eviction (F1) declares overfitting only for post-epoch-0 peaks that are followed by sustained decline, and it protects epoch-0-peak victims. This rule lifts the 3B ten-benchmark average from 73.30 to 73.60 and the mathematics group from 60.25 to 62.95 (GSM8K 76.50→77.80, AQUA 44.00→48.10), above the 62.10 full-model MSFT reference. Behaviorally, the repaired run evicts CommonsenseQA first and never evicts GSM8K, which keeps training under the protection rule (Table 14). The GSM8K gain is therefore trained rather than a rollback residual. On Qwen2.5-0.5B the same rule holds the overall score (41.70 vs. 41.75) while raising mathematics from 19.50 to 21.20. Stagnation-aware termination (F3) is validated by the replay above. When run online together, F1 and F3 reach 73.50 on the 3B configuration. The MSFT+LoRA comparison (Table 3) completes the attribution: the linear baseline is uniformly stronger on mathematics yet still trails the full-model reference, so the regression is dominated by the eviction rule, not the adapter parameterization. Per-group rank allocation (F2), which uses block-diagonal adapter subspaces so that evictions stop erasing other groups' capacity, remains future work.

5 Discussion and Limitations

Novelty and broader value. AuroSFT does not claim heterogeneous overfitting (from MSFT) or input-level AuroRA. Its new object is the *stage-transfer state*, a mergeable nonlinear low-rank

Scheduler variant	Qwen2.5-3B Overall	Qwen2.5-3B Math	Qwen2.5-0.5B Overall	Qwen2.5-0.5B Math
AuroSFT (reported)	73.30	60.25	41.75	19.50
+ interference-aware eviction (F1)	**73.60**	**62.95**	41.70	21.20
+ F1 and stagnation-aware termination (F3)	73.50	—	—	—
MSFT (full-model reference)	73.10	62.10	37.40	**23.50**
MSFT+LoRA (linear baseline)	72.40	—	**42.10**	20.60

Table 4: **Scheduler-level repairs on the audited configurations.** F1 replaces the earliest-peak eviction rule and protects epoch-0-peak interference victims, while F3 stops after two consecutive no-improvement stages (offline replay: reported checkpoints unchanged, 30%/20% of the 3B/OLMo2-1B stage budget saved). Bold marks the column best. — = decomposition not retained.

weight parameterization that the scheduler carries instead of a full checkpoint. The design helps most when tasks peak at different times and intermediate states must be retained. When peaks are nearly aligned, a global stopping point suffices.

Evidence scope and limitations. Main-table rows are single-seed point estimates. A four-seed Qwen2.5-3B study (seeds 42–45) gives 73.13 ± 0.24 for AuroSFT versus 73.13 ± 0.26 for reproduced MSFT, so the 3B margin is parity, and the stable summary is the aggregate ordering ($+1.51$ over the MSFT reference, $+0.08$ over scheduler-matched LoRA). The aggregate lead also coexists with a consistent regression: AuroSFT trails MSFT in the Mathematical and Quantitative group on every backbone (1.30–4.45 points), so the average is carried by the science and commonsense groups. Section 4.5 traces this regression to a reproducible scheduler failure mode, namely interference misread as overfitting, and the interference-aware eviction rule repairs it on the audited configurations (mathematics 60.25→62.95 on 3B, above the 62.10 full-model reference, as shown in Table 4). Per-group rank allocation and parameter-count-matched linear controls remain open. The MSFT+LoRA baseline matches rank, injection points, and budget but not exact parameter count, because ANL adds spline and self-projection parameters. It spans all five backbones, where it is uniformly stronger on mathematics while trailing on aggregate. Input-level AuroRA is not compared empirically because its deployed form cannot merge into the frozen projection. The evidence is strongest on five lightweight backbones and Qwen2.5-3B diagnostics, and all ten benchmarks are English-language, which leaves multilingual generality open. Because the update remains input-linear and rank-bounded, we avoid claims from input-level AuroRA theory. Stronger causal evidence requires multi-seed studies beyond the 3B pair with significance testing, together with hardware profiling. Our efficiency claim covers rollback storage and mergeable deployment, not wall-clock training time.

6 Conclusion

We introduced AuroSFT, a parameter-efficient framework that combines overfitting-aware scheduling with nonlinear weight-transformed low-rank adapters. Its compact, mergeable stage-transfer state enables adapter-only rollback while remaining input-linear after merging. Across five lightweight backbones, AuroSFT improves the comparison-row average from 59.85% to 61.36% and leads on all five (61.28% for the scheduler-matched linear-LoRA baseline). Qwen2.5-3B diagnostics support iterative scheduling, rollback, and Weight ANL. The stage-log audit further shows that the remaining gap to full-model scheduling is diagnosable and, for two of the three failure modes, already repaired: interference-aware eviction recovers the mathematics group above the full-model reference, and stagnation-aware termination cuts 30%/20% of the 3B/OLMo2-1B stage budget at unchanged accuracy. More broadly, the state carried by a dynamic data schedule is a design object: task-specific timing can be preserved without full-model checkpoints.

Use of Generative AI. LLM assistants were used only to polish English wording and to assist with manuscript formatting. All technical content, the experimental design, and every reported number are the authors' own work, derived from the archived training and evaluation logs of the audited runs.

REFERENCES

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

Guanting Dong, Hongyi Yuan, Keming Lu, Chengpeng Li, Mingfeng Xue, Dayiheng Liu, Wei Wang, Zheng Yuan, Chang Zhou, and Jingren Zhou. How abilities in large language models are affected by supervised fine-tuning data composition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 177–198, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.12. URL https://aclanthology.org/2024.acl-long.12/.

Haonan Dong, Wenhao Zhu, Guojie Song, and Liang Wang. Aurora: Breaking low-rank bottleneck of lora with nonlinear mapping. *arXiv preprint arXiv:2505.18738*, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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*, 2019.

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. *arXiv preprint arXiv:2106.09685*, 2021.

Ting Hu and Yunwen Lei. Early stopping for iterative regularization with general loss functions. *Journal of Machine Learning Research*, 23(339):1–36, 2022. URL http://jmlr.org/papers/v23/21-0983.html.

Woosung Koh, Jeyoung Jeon, Youngjin Song, Yujin Cheon, Soowon Oh, Jaehyeong Choi, and Se-Young Yun. msft: Addressing dataset mixtures overfitting heterogeneously in multi-task sft. *arXiv preprint arXiv:2603.21606*, 2026.

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*, 2021.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7602–7635, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.421. URL https://aclanthology.org/2024.naacl-long.421/.

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

Yuan Li, Zhengzhong Liu, and Eric Xing. Data mixing optimization for supervised fine-tuning of large language models. *arXiv preprint arXiv:2508.11953*, 2025.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1015. URL https://aclanthology.org/P17-1015/.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260/.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In Gerardo Flores, George H Chen, Tom Pollard, Joyce C Ho, and Tristan Naumann (eds.), *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pp. 248–260. PMLR, 07–08 Apr 2022. URL https://proceedings.mlr.press/v174/pal22a.html.

Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00010-0. URL https://www.sciencedirect.com/science/article/pii/S0893608098000100.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020.

Kai Shi, Jun Yang, Ni Yang, Binqiang Pan, Qingsong Xie, Chao Zhang, Zhenyu Yang, Tianhuang Su, and Haonan Lu. Damo: Data mixing optimizer in fine-tuning multimodal llms for mobile phone agents, 2025. URL https://arxiv.org/abs/2510.19336.

Haebin Shin, Lei Ji, Xiao Liu, Zhiwei Yu, Qi Chen, and Yeyun Gong. Dynamixsft: Dynamic mixture optimization of instruction tuning collections. *arXiv preprint arXiv:2508.12116*, 2025.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421/.

Evan Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James Validad Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 furious (COLM's version). In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=2ezugTT9kU.

Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *arXiv preprint arXiv:2407.05000*, 2024.

Weixuan Wang, Minghao Wu, Barry Haddow, and Alexandra Birch. HBO: Hierarchical balancing optimization for fine-tuning large language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=JnhahbMvRE.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin (eds.), *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Yuxin Xiao, Shujian Zhang, Wenxuan Zhou, Marzyeh Ghassemi, and Sanqiang Zhao. Sftmix: Elevating language model instruction tuning with mixup recipe. *arXiv preprint arXiv:2410.05248*, 2024.

Suqin Yuan, Runqi Lin, Lei Feng, Bo Han, and Tongliang Liu. Instance-dependent early stopping. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=P42DbV2nuV.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023a.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023b.

Tong Zhu, Daize Dong, Xiaoye Qu, Jiacheng Ruan, Wenliang Chen, and Yu Cheng. Dynamic data mixing maximizes instruction tuning for mixture-of-experts. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1663–1677, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.80. URL https://aclanthology.org/2025.naacl-long.80/.

Reproducibility Statement. The full protocol, which covers hardware and software versions, hyperparameters and the search ledger, datasets and evaluation rules, the audit ledger, and reproduction commands, is documented in Appendices B–I. The anonymous code release is linked in the abstract.

This appendix expands the experimental audit trail for AuroSFT. It is organized around the scheduler algorithm, reproducibility, checkpoint selection, task-level evidence, further analysis, resource

analysis, and the correspondence between the mathematical notation in the main paper and the released implementation. The material is intentionally evidence-facing: it reports the evidence role of each result, highlights the formal main-result trail, and makes the checkpoint-selection rule explicit.

Unless otherwise stated, accuracy values are percentages. The overall score is the unweighted mean over the ten benchmark families used in the main paper. The preserved experimental artifacts include configuration files, stage summaries, per-stage accuracy logs, train logs, and generated plots. Large model weights, generated checkpoints, prediction dumps, and raw run logs are excluded from the public repository.

A AuroSFT Scheduler Algorithm

This section restates the full scheduling loop that Section 3 summarizes. The pseudocode makes explicit that only the adapter state is restored at a stage transition, while the scheduling logic itself is inherited unchanged from MSFT.

Algorithm 1 AuroSFT Scheduler. The loop terminates when the active mixture is empty or the stage limit $S_{\max}$ (at most ten stages in all audited runs) is reached, guaranteeing termination.

Input: Dataset mixture $\mathcal{D}$, frozen backbone W_0, adapter state $\phi_0 = \{A, B, \mathrm{ANL}\}$, per-stage budget C, maximum stages $S_{\max}$
Output: Final adapter state $\hat{\phi}$ and optionally merged model $W_0 + \Delta W$
$\mathcal{E}_0 \leftarrow \emptyset$; $\hat{\phi} \leftarrow \phi_0$
for $s = 0, 1, \ldots, S_{\max}$ **do**
  if $\mathcal{D} \setminus \mathcal{E}_s = \emptyset$ **then**
    break
  end if
  $\phi(c), \{\mathrm{acc}_i(c)\}_{i,c} \leftarrow \mathrm{AdapterRollout}(W_0, \hat{\phi}, \mathcal{D} \setminus \mathcal{E}_s, C)$
  $c_i^* \leftarrow \arg\max_c \mathrm{acc}_i(c)$ for each $\mathcal{D}_i \notin \mathcal{E}_s$
  $(c_{\min}, i_{\min}) \leftarrow \arg\min_i c_i^*$
  if $c_{\min} = C$ **then**
    $\mathcal{E}_{s+1} \leftarrow \mathcal{E}_s$
    $\hat{\phi} \leftarrow \phi(C)$
  else
    $\mathcal{E}_{s+1} \leftarrow \mathcal{E}_s \cup \{\mathcal{D}_{i_{\min}}\}$
    $\hat{\phi} \leftarrow \phi(c_{\min})$ {rollback only adapter state}
  end if
end for

B Experimental Details

B.1 Hardware Environment

The audited experiments separate the remote training environment from the local manuscript workflow. Table 5 records the compute, precision, evaluation, and checkpoint conditions needed to interpret the reported accuracy and storage results. In particular, rollback refers to restoring adapter tensors and metadata rather than the frozen backbone.

Item	Configuration used in audited runs
Compute host	Remote Linux training server used for all audited training and evaluation runs. The local workstation is used only for manuscript compilation and artifact inspection.
Accelerators	Two NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs, addressed as GPU 0 and GPU 1 in the archived run scripts and logs. Each GPU reports 97,887 MiB total memory in `nvidia-smi`, corresponding to 94.97 GiB visible to PyTorch.
Driver and CUDA visibility	NVIDIA driver 595.58.03. `nvidia-smi` reports CUDA 13.2, while the preserved PyTorch environment is built with CUDA 12.8 and detects both GPUs.
Training stack	PyTorch with Hugging Face Transformers-style model loading and generation. Checkpoints are saved as adapter artifacts plus metadata.
Evaluation backend	Local Hugging Face generation/evaluation with 5-shot greedy decoding for the ten-benchmark suite.
Precision and state	Adapter tensors are stored as compact low-rank states. Storage comparisons use bf16 tensor-payload estimates unless otherwise noted.
Checkpoint policy	During stage transitions, AuroSFT restores only the adapter state $\phi = \{A, B, \mathrm{ANL}\}$ and associated metadata, rather than a full backbone checkpoint.

Table 5: **Experimental environment.** The table summarizes the environment and artifact policy used by the audited AuroSFT experiments.

B.2 Software and Library Versions

We preserve package versions because small changes in model loading, generation, or serialization can affect reproducibility. Table 6 reports the inspected server environment rather than an inferred dependency list. Project-specific adapters are implemented directly in the codebase, so absent third-party PEFT packages are noted explicitly.

Software component	Version observed in the AuroSFT environment
Operating system	Ubuntu 22.04.4 LTS. Linux kernel 5.15.0-78-generic. glibc 2.35.
Python environment	Dedicated AuroSFT Python environment. Python 3.11.15.
PyTorch stack	PyTorch 2.11.0+`cu128`. Torch CUDA build 12.8. cuDNN version code 91900.
Hugging Face stack	Transformers 4.57.6. Accelerate 1.14.0. Datasets 5.0.0. Tokenizers 0.22.2. Hugging Face Hub 0.36.2.
Numerical and data libraries	NumPy 2.4.4. SciPy 1.17.1. Pandas 3.0.3. scikit-learn 1.9.0. tqdm 4.68.3.
Serialization and tokenization	Safetensors 0.8.0. SentencePiece 0.2.1. Triton 3.6.0. Packaging 26.0.
Project-specific adapter code	External `peft`, `bitsandbytes`, `deepspeed`, `evaluate`, and `flash-attn` packages were not installed in the inspected environment. AuroSFT adapters are implemented in the project codebase.
Compiler	GCC 11.4.0.

Table 6: **Software environment.** Version information was read from the preserved AuroSFT Python environment and complements the hardware summary in Table 5.

B.3 Hyperparameter Selection and Settings

The primary setting is deliberately conservative and shared wherever the audited runs permit. Model-specific selected runs may change rank or stage budget, but retain the same adapter definition and scaling convention. Table 7 distinguishes these implementation constants from search-dependent choices. Table 8 lists the hyperparameter dimensions varied during tuning and audit, separating mSFT training-protocol variables from AuroRA-specific variables so that the comparison keeps the adapter-state mechanism distinct from the full-model training recipe.

Hyperparameter	Main manuscript setting
Backbones	OLMo2-1B and Qwen2.5 backbones at 0.5B, 1.5B, 3B, and 7B scales.
Trainable parameters	Frozen backbone. Only injected AuroSFT adapters are optimized and checkpointed.
Target modules	Transformer projection layers `q/k/v/o/gate/up/down`.
Adapter form	Weight-transformed low-rank adapter with $\bar{A} = \sigma(A^\top)^\top$ and $\Delta W = (\alpha/r)B\bar{A}$.
Rank and scaling	Rank-8, $\alpha = 16$ is the primary Qwen2.5-3B configuration. Model-specific selected runs keep the scaling convention $\alpha/r = 2$ when rank changes.
Dropout	Adapter dropout 0.05 in the primary conservative configuration.
Optimizer and learning rate	Adam-style optimizer over adapter parameters. Learning rate 1×10^{-5} for the primary reported configuration.
Schedule and warmup	Cosine schedule with 3% warmup for the primary conservative 2-epoch-per-stage AuroSFT runs.
Stage budget	Up to ten overfitting-aware stages. Each stage rolls out the active mixture, detects task-wise peaks, excludes the earliest-overfitted task, rolls back the adapter, and continues.
Evaluation	Evaluation every 0.25 epoch when logs are available. 5-shot greedy decoding. Evaluation batch size is selected to avoid GPU memory overflow, with 192 used in validated Qwen runs when feasible.

Table 7: **Experimental hyperparameters.** The table records the main protocol values used for the reported AuroSFT experiments. When model-specific searches select a different rank or budget, the corresponding result is treated as an audited selected run rather than a change to the method definition.

Dimension	Tried values or branches	Selected setting	Role
Learning rate	1×10^{-5}	1×10^{-5}	Training protocol
Scheduler	cosine, constant	backbone-specific (conservative 3B uses cosine)	Training protocol
Warmup ratio	0.03, 0.0	0.03 for conservative 2-epoch runs	Training protocol
Epochs per stage	2, 5, 10	selected by backbone/run family	Stage budget
Test batch size	32, 64, 128, 192, 256	largest stable value per model	Evaluation throughput
Target strategy	full, attention-only	full for main conservative runs	AURORA module
Adapter rank	8, 16	rank 8 main, rank 16 for selected small/enhanced runs	AURORA module
Adapter alpha	16, 32	keeps $\alpha/r = 2$ when rank changes	AURORA module
Adapter dropout	0.05, 0.10	0.05 main	AURORA module
ANL mode	tanh spline, gated residual variants	`baseline_tanh_spline` main	AURORA module
Scale noise	0.1, 0.05	0.1 main	AURORA module

Table 8: **Hyperparameter search ledger.** The ledger makes explicit which variables belong to the training protocol and which belong to the proposed adapter.

B.4 Datasets, Prompts, and Evaluation Rules

Evaluation covers ten public benchmark families spanning knowledge, language and commonsense, and quantitative reasoning. We retain the grouping used in the MSFT comparison so that group-level and overall averages remain directly comparable. The overall score is the unweighted mean over the ten benchmark families under the stated 5-shot protocol. Table 9 records the grouping, Table 10 the local dataset mixtures, and Table 11 the prompt and evaluation rules.

Task group	Number of tasks	Benchmarks
Science and Knowledge	4	ARC-Easy, OpenBookQA, SciQ, MedMCQA
Commonsense and Language	4	CommonsenseQA, HellaSwag, WinoGrande, BoolQ
Mathematic and Quantitative	2	AQUA-RAT, GSM8K
Overall suite	10	The average-accuracy block reports the mean over all ten benchmark families under 5-shot greedy evaluation.

Table 9: **Task grouping and datasets.** We follow the task grouping used by the mSFT main experiment and report averages over the five evaluated backbones in the main manuscript table.

Dataset file	Train	Validation	Task families	Output-style tasks
`mix-5cat-9k.json`	8,996	1,000	ARC-Easy, BoolQ, GSM8K, MedMCQA, HellaSwag	GSM8K
`mix-10cat-18k.json`	17,994	2,000	Ten main benchmark families	AQUA-RAT, GSM8K
`mix-15cat-27k.json`	26,994	3,000	Fifteen-task heterogeneous mixture	AQUA-RAT, GSM8K
`mix-medmcqa-19k.json`	18,900	2,100	Twenty-one MedMCQA subcategories	None

Table 10: **Dataset construction summary.** Counts are reconstructed from preserved dataset metadata and local files. Each mixture contains a train split and validation split, and validation accuracy is computed with the same five-shot prompting protocol used for the ten-benchmark suite.

Item	Evaluation rule
Prompting	Five demonstrations are prepended using task-specific templates.
Decoding	Greedy generation is used for all reported accuracy logs.
Answer extraction	Multiple-choice tasks map generated text to normalized option labels. Quantitative tasks use task-specific answer extraction.
Aggregation	Overall accuracy is the unweighted mean of the ten benchmark-family accuracies.
Stage decision	Validation curves determine peak checkpoint selection and active task-group updates.

Table 11: **Prompt and evaluation rules.** Validation curves are used for stage decisions and checkpoint selection. Reported benchmark accuracies are then computed under the same five-shot greedy evaluation protocol and unweighted ten-family aggregation used in the main manuscript. This separation keeps the rollback decision rule distinct from the final reporting metric.

B.5 Rollback Payload and Evaluation-Split Protocol

Rollback payload. At each stage transition, the implementation restores only the adapter tensors ($\{A, B, \text{ANL}\}$) and the `aurora_metadata.json` manifest. Optimizer moments are not checkpointed at peaks and are therefore not restored: each stage continues from the restored peak checkpoint with fresh optimizer state, and the cosine schedule with 3% warmup restarts per stage. The footprint comparisons in the main paper (Figure 8) count exactly this payload, namely adapter tensors plus metadata, and exclude the frozen backbone.

Validation/test split protocol. Stage decisions and checkpoint selection are driven by the per-mixture validation subsamples listed in Table 10 (1,000–3,000 held-out examples per mixture). Reported benchmark accuracies are computed on the ten benchmark evaluation sets under the same 5-shot greedy protocol and the unweighted ten-family aggregation. The rollback decision rule is thus evaluated on separate data from the reporting metric, so checkpoints are not selected directly on the reported evaluation sets.

Epoch notation. "Ep." in the main and ablation tables denotes the epoch position of the selected checkpoint: stage-local within its stage for stage-scheduled methods (written Sx/Ey in the audit ledger, e.g., S5/E0.50 for the Qwen2.5-3B main run) and global for single-run baselines.

C Experiment Audit Ledger and Checkpoint Selection

Table 12 summarizes the principal experimental evidence used in the paper-facing comparison. The "source" column separates formal main-table results from reproduction rows used for matched comparison, keeping the headline five-backbone comparison aligned with the main manuscript.

Backbone	Method	Run family	Key setting	Selection point	Best acc.	Source
OLMo2-1B	AuroSFT	conservative	rank 8, cosine, 2 epochs	S6/E1.75	48.40	archived logs and main table
Qwen2.5-0.5B	AuroSFT	small-model enhanced	rank 16, constant, 5 epochs	S2/E0.25	41.75	archived logs and main table
Qwen2.5-1.5B	AuroSFT	conservative selected	rank 8, cosine, 2 epochs	S2/E2.00	65.85	archived logs and main table
Qwen2.5-3B	AuroSFT	conservative main	rank 8, cosine, 2 epochs	S5/E0.50	73.30	archived logs and main table
Qwen2.5-7B	AuroSFT	main selected run	selected 7B run	E2.50	77.50	main table result
Qwen2.5-3B	MSFT	strict repro	full fine-tuning protocol	S3/E2.50	73.00	archived repro logs
Qwen2.5-7B	MSFT	strict repro	full fine-tuning protocol	S1/E1.25	77.45	archived repro logs
OLMo2-1B	MSFT+LoRA	scheduler-matched	rank 8, cosine, 2 epochs	S3/E1.75	48.60	archived logs and main table
Qwen2.5-0.5B	MSFT+LoRA	scheduler-matched	rank 16, constant, 5 epochs	S2/E0.25	42.10	archived logs and main table

Table 12: **Experiment audit ledger.** The table records the most important result-bearing runs and explicitly marks whether each value is a formal main-table result or a reproduction row.

AuroSFT follows the same peak-aware stage logic as the main paper for all formal results, including the Qwen2.5-7B 77.50% row: roll out the active mixture, identify task-level validation peaks, update the active task group, and restore the adapter state at the selected peak. Table 13 gives representative rollback events reconstructed from the stage summaries and accuracy logs. The selection rule is validation-driven and stage-aware.

Run	Selection point	Peak acc.	Selection note
3B main	S5/E0.50	73.30	Global best adapter checkpoint.
7B main selected	E2.50	77.50	Formal manuscript result under the same peak-aware selection rule.
1.5B conservative	S2/E2.00	65.85	Selected manuscript value.
0.5B rank16	S2/E0.25	41.75	Early peak under longer small-model training.
OLMo2-1B	S6/E1.75	48.40	Best cross-stage checkpoint.

Table 13: **Representative checkpoint selection events.** Adapter checkpoints are selected by validation accuracy and stage state, not by manually inspecting the test outcome after training finishes.

C.1 Cross-Backbone Exclusion Audit

The exclusion timelines cited in Section 4.5 are reconstructed from the per-stage configuration logs of each retained run. Every stage directory records which tasks were excluded from the training mixture and which checkpoint the stage resumed from, so the resume chain recovers the true eviction order even across restarted stages. Table 14 lists the complete sequences and each run's best accuracy. The global best freezes at S5 on the 3B main run (0.733), at S6 on OLMo2-1B (0.484), and at S7 on the 0.5B rerun (0.329). The audited 0.5B run is a conservative rerun under the shared code path, whereas the main-table 0.5B result (rank 16) is a different run whose stage logs were not retained. The two-stage 7B fragment evicts nothing, but GSM8K already falls from 0.850 to 0.790 within its first stage, which shows the same epoch-0 interference signature observed before any eviction occurs. The F1 repair row is reconstructed from that run's own per-stage configuration chain: the protected order evicts CommonsenseQA first and never evicts GSM8K.

Run	Complete exclusion order	Best
Qwen2.5-3B (main)	**GSM8K** → CSQA → ARC-E → **AQUA** → MedMCQA → SciQ → BoolQ → WinoGrande → HellaSwag	0.733
Qwen2.5-3B (early)	**GSM8K** → ARC-E → **AQUA** → MedMCQA → CSQA → SciQ → HellaSwag → OBQA → WinoGrande	0.730
Qwen2.5-3B (F1 repair)	CSQA → ARC-E → **AQUA** → MedMCQA → SciQ → BoolQ → OBQA → WinoGrande → HellaSwag (GSM8K retained)	0.736
OLMo2-1B (main)	**GSM8K** → SciQ → BoolQ → ARC-E → WinoGrande → MedMCQA → HellaSwag → OBQA → CSQA	0.484
Qwen2.5-0.5B (rerun)	**AQUA** → SciQ → HellaSwag → OBQA → WinoGrande → BoolQ → MedMCQA → ARC-E → CSQA	0.329
Qwen2.5-7B (fragment)	— (two stages, no exclusion)	—

Table 14: **Complete exclusion orders across all audited runs.** Mathematical tasks in bold. In every baseline run that evicts a task, the first eviction is a mathematical task. The interference-aware F1 repair instead evicts CommonsenseQA first, evicts AQUA only at its true post-epoch-0 peak (third), and retains GSM8K throughout.

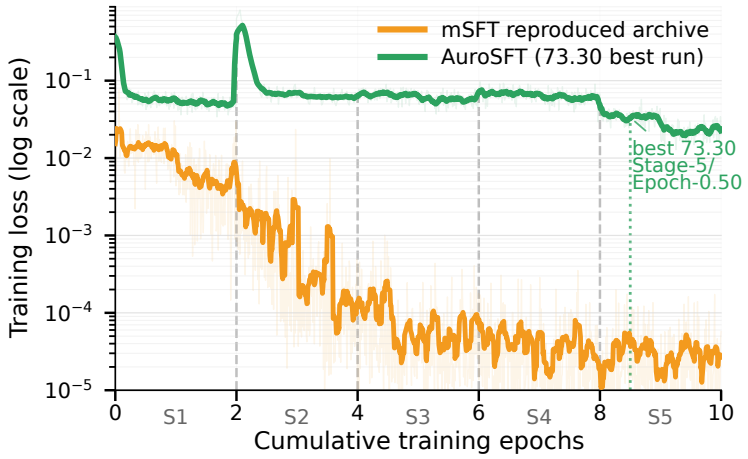


Figure 4: **Training-loss dynamics on Qwen2.5-3B.** The AuroSFT curve restarts from rolled-back adapter checkpoints at stage boundaries (S1–S5 marked), while reproduced MSFT tracks full-state training. The curves diagnose adapter-state continuation across stage boundaries in the matched run, and lower loss is not treated as proof of higher accuracy.

D Full Ten-Benchmark Results

Table 15 expands the aggregate main-table averages into per-benchmark accuracies for the best available AuroSFT checkpoints with complete task-level breakdowns in this appendix. The table highlights how the selected AuroSFT checkpoints perform across the ten benchmark families while preserving the headline five-backbone comparison used in the main manuscript.

Backbone	ARC	Hella.	BoolQ	Med.	GSM8K	CSQA	OBQA	Wino.	AQUA	SciQ	Overall
OLMo2-1B	78.50	61.50	71.50	33.00	20.50	62.00	63.00	48.00	17.00	29.00	48.40
Qwen2.5-0.5B	70.00	41.00	67.50	34.00	20.50	55.50	51.00	48.00	18.50	11.50	41.75
Qwen2.5-1.5B	91.00	81.50	82.00	52.50	56.50	79.50	80.00	58.50	38.50	38.50	65.85
Qwen2.5-3B	95.00	91.50	86.00	62.50	76.50	86.00	85.50	62.00	44.00	44.00	73.30

Table 15: **Ten-benchmark accuracy breakdown.** Rows show complete per-benchmark breakdowns available for supplemental analysis of selected AuroSFT checkpoints.

Table 16 reports the manuscript-facing comparisons where both AuroSFT and pure MSFT values are available under the same backbone family. These rows are useful for interpreting the main

claim conservatively: AuroSFT is competitive with strict full-model reproduction while storing and carrying only adapter state.

Backbone	AuroSFT best	MSFT repro best	Δ
Qwen2.5-3B	73.30	73.00	+0.30
Qwen2.5-7B main selected run	77.50	77.45	+0.05

Table 16: **Matched manuscript-facing comparison summary.** These rows align with the formal comparison used by the main manuscript and highlight AuroSFT's compact adapter-state alternative to full-model reproduction.

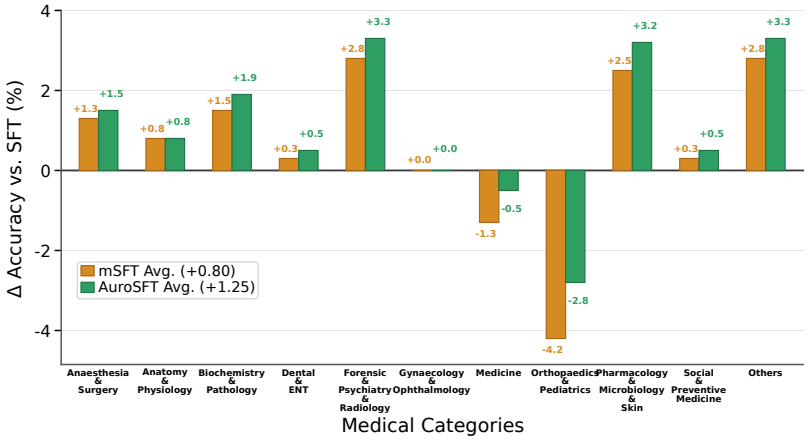


Figure 5: **Fine-grained MedMCQA results on Qwen2.5-3B.** Bars aggregate the underlying 21 MedMCQA subcategories into 11 groups following MSFT. The +1.25-point average is computed on the same grouping and is not the unweighted mean of the displayed bars. AuroSFT improves the reproduced average gain and reduces the largest negative category-level drops.

E Further Analysis

We further examine when the framework helps, what state it changes, and how its training dynamics differ from full-state multi-task SFT. The compact diagnostics are shown in Figures 6 and 7.

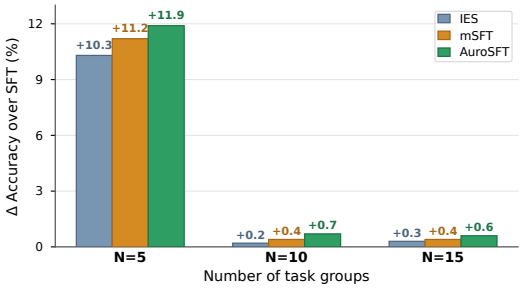


Figure 6: **Robustness across mixture scales on Qwen2.5-3B.** AuroSFT retains the largest gain over SFT at all evaluated mixture sizes under the matched setup.

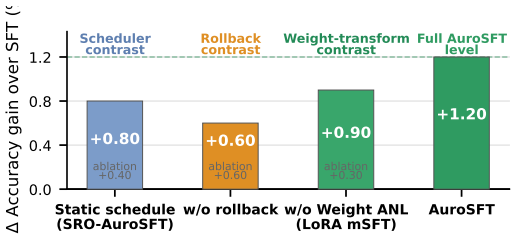


Figure 7: **Component sensitivity on Qwen2.5-3B.** Bars are component-removal *deficits* relative to full AuroSFT (e.g., the 0.90 no-ANL bar equals 73.30%−72.40% in Table 2), not gains over SFT.

(I) Robustness across mixture scales. Following the scale-stress analysis in MSFT, we vary the number of task groups on Qwen2.5-3B and measure improvements over a fixed-mixture SFT baseline. As shown in Figure 6, for $N \in \{5, 10, 15\}$ groups, AuroSFT obtains gains of +11.9, +0.7, and

$+0.6$ points, respectively, compared with $+11.2$, $+0.4$, and $+0.4$ for reproduced MSFT and $+10.3$, $+0.2$, and $+0.3$ for IES. The ordering is stable across all three settings.

(II) Fine-grained task granularity. On the MedMCQA granular decomposition used by MSFT (21 subcategories grouped into 11 categories), AuroSFT improves over SFT by $+1.25$ points on average versus $+0.80$ for reproduced MSFT. It is no worse on 10 of 11 groups, strictly better on 8, and shows less severe negative categories, which suggests reduced negative transfer after exclusion (Figure 5, Appendix D).

(III) Component sensitivity. The ablations in Table 2 and Figure 7 separate three effects that otherwise appear entangled: the schedule, the rollback operation, and the nonlinear weight transform. Single-roll-out and soft variants underperform the full method, so a static estimate of task peaks is not a sufficient substitute for iterative rollback. Removing rollback while keeping the adapter form also degrades accuracy, which shows that the checkpointed state is an active part of the algorithm. The remaining contrast between AuroSFT and the scheduler-matched MSFT+LoRA baseline isolates the nonlinear weight parameterization.

(IV) Overfitting prevention and adapter-state forgetting. Let $M_i(\phi)$ be task i's validation metric under adapter state ϕ (precise definitions in Appendix G, Prop. 3). If $\phi_i^\star$ is the adapter checkpoint at task i's validation peak and ϕ_{final} is the final adapter state, the final gain over SFT decomposes into a peak-gain bracket (overfitting prevention) plus a peak-to-final bracket (forgetting or transfer after exclusion). AuroSFT does not change this identity. It changes the state in which the identity is realized, namely a compact, mergeable adapter state instead of a full-model checkpoint.

(V) Optimization dynamics. Figure 4 (Appendix C.1) plots Qwen2.5-3B training-loss traces: the AuroSFT curve restarts from rolled-back adapter checkpoints at stage boundaries, while reproduced MSFT tracks full-state training.

F Resource and Checkpoint Footprint

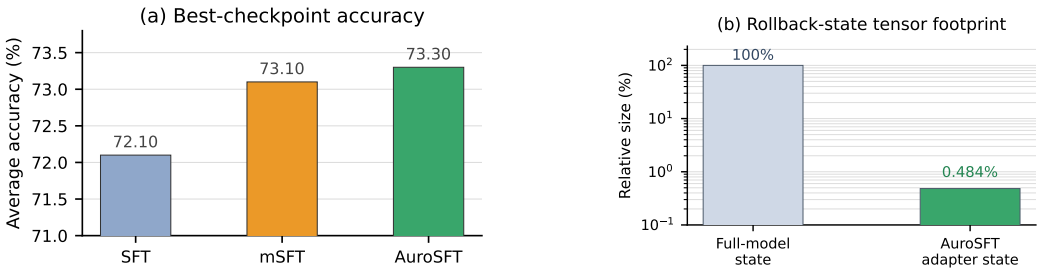


Figure 8: **Accuracy and checkpoint-state efficiency on Qwen2.5-3B.** (**a**) Best-checkpoint accuracy: AuroSFT keeps the best checkpoint accuracy. (**b**) Rollback-state tensor footprint (log scale): the adapter payload is 0.484% of the full-model tensor state.

Table 17 records the resource-facing interpretation used in the paper. The central advantage is in trainable state and checkpoint footprint, because AuroSFT stores adapter tensors and metadata rather than full-model snapshots.

Resource item	Audited interpretation
Backbone parameters	Frozen during AuroSFT training.
Trainable state	Low-rank adapter factors and ANL parameters.
Checkpoint payload	Adapter tensors plus `aurora_metadata.json`.
3B adapter footprint	Approx. 0.484% of full-model tensor state in the bf16 tensor-payload estimate. Adapter tensors are counted, and duplicated frozen backbone tensors are excluded.
7B trainable ratio	About 0.53% of total parameters for the rank16 adapter branch, counting trainable adapter/ANL parameters against the frozen-backbone total.
GPU memory	Profiling complements the primary checkpoint-footprint analysis.
Public repository	Excludes base model weights, checkpoints, raw logs, predictions, and PID files.

Table 17: **Resource and checkpoint footprint details.** The reproducible storage claim is tied to adapter-state checkpointing rather than full-model checkpointing.

G Complete Mathematical Derivation of AuroSFT

This section expands the algebraic reasoning that is compressed in the main paper. The following derivation establishes structural properties of the adapter parameterization and the stage-transfer state.

Setup and dimensions. Consider one frozen Transformer projection with base weight

$$W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, \qquad x \in \mathbb{R}^{d_{\text{in}}}. \tag{5}$$

The unadapted projection returns $h_0 = W_0 x$. AuroSFT attaches two trainable low-rank factors,

$$A \in \mathbb{R}^{r \times d_{\text{in}}}, \qquad B \in \mathbb{R}^{d_{\text{out}} \times r}, \tag{6}$$

where $r \ll \min(d_{\text{in}}, d_{\text{out}})$ in the parameter-efficient regime. The adaptive nonlinear layer is applied to the weight factor rather than the sample representation:

$$\bar{A} = \sigma(A^\top)^\top, \qquad \bar{A} \in \mathbb{R}^{r \times d_{\text{in}}}. \tag{7}$$

The adapter update is therefore

$$\Delta W(\phi) = \frac{\alpha}{r} B\bar{A} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}, \qquad \phi = \{A, B, \sigma\}, \tag{8}$$

and the adapted projection is

$$h = W_0 x + \Delta W(\phi) x. \tag{9}$$

Lemma 1: input-linearity after weight transformation. For a fixed adapter state ϕ, the map $x \mapsto W_0 x + \Delta W(\phi) x$ is linear in the input x.

Proof sketch. Once the adapter state ϕ is fixed, $\bar{A} = \sigma(A^\top)^\top$ is a fixed matrix and $\Delta W(\phi) = (\alpha/r) B\bar{A}$ is also a fixed matrix. Therefore

$$W_0 x + \Delta W(\phi) x = (W_0 + \Delta W(\phi))\, x. \tag{10}$$

For any scalars λ, μ and inputs x_1, x_2,

$$\begin{aligned} &(W_0 + \Delta W(\phi))\,(\lambda x_1 + \mu x_2) \\ &\quad = \lambda\,(W_0 + \Delta W(\phi))\, x_1 + \mu\,(W_0 + \Delta W(\phi))\, x_2. \end{aligned} \tag{11}$$

Thus the nonlinear map changes the learned weight state, not the input-side functional form of the deployed projection.

Lemma 2: rank boundedness. The AuroSFT update satisfies

$$\operatorname{rank}(\Delta W(\phi)) \le r. \tag{12}$$

Proof sketch. By the standard matrix rank inequality,

$$\operatorname{rank}(MN) \le \min\{\operatorname{rank}(M), \operatorname{rank}(N)\}. \tag{13}$$

Substituting $M = B \in \mathbb{R}^{d_{\text{out}} \times r}$ and $N = \bar{A} \in \mathbb{R}^{r \times d_{\text{in}}}$ gives

$$\begin{aligned} \operatorname{rank}(\Delta W(\phi)) &= \operatorname{rank}\Big(\frac{\alpha}{r} B\bar{A}\Big) \\ &\le \min\{\operatorname{rank}(B), \operatorname{rank}(\bar{A})\} \\ &\le r. \end{aligned} \tag{14}$$

The scalar α/r does not increase rank. Thus the nonlinear transformation may alter the entries of the low-rank factor, but it cannot make the final update exceed the rank bottleneck induced by the factorization.

Proposition 1: exact mergeability. For a fixed trained adapter state $\hat{\phi}$, the adapter-augmented projection is exactly equivalent to a single merged projection

$$W_{\text{merge}} = W_0 + \Delta W(\hat{\phi}). \tag{15}$$

Proof sketch. For every input x,

$$W_0 x + \Delta W(\hat{\phi}) x = \Big(W_0 + \Delta W(\hat{\phi})\Big)\, x = W_{\text{merge}} x. \tag{16}$$

The equality holds pointwise for all inputs because $\Delta W(\hat{\phi})$ is a weight-space matrix at inference time. This is the reason the method can use a nonlinear adapter state during training while preserving a standard linear projection after merging.

Proposition 2: adapter-state sufficiency for rollback. Assume the backbone W_0 is frozen during all stages. Then restoring the stage state only requires restoring $\phi_s = \{A_s, B_s, \sigma_s\}$ and the corresponding adapter metadata, so a full checkpoint of W_0 is not required.

Proof sketch. At any stage s, the deployed trainable projection is determined by

$$F_s(x) = (W_0 + \Delta W(\phi_s))\, x. \tag{17}$$

Since W_0 is constant across stages, the only stage-dependent term is $\Delta W(\phi_s)$, which is completely determined by ϕ_s. For two candidate stages s and t,

$$F_s(x) - F_t(x) = (\Delta W(\phi_s) - \Delta W(\phi_t))\, x. \tag{18}$$

Therefore stage rollback is a rollback in adapter-state space. Restoring ϕ_t reconstructs the same adapted projection that would have been produced at stage t under the frozen-backbone assumption.

Proposition 3: task-level peak and stage-transfer accounting identity. Let

$$M_i(\phi) = \text{Metric}\big(W_0 + \Delta W(\phi); \mathcal{D}_i^{\text{eval}}\big) \tag{19}$$

be the validation metric for task i under adapter state ϕ. If $\phi_i^\star$ is the adapter state at task i's validation peak and ϕ_{final} is the final adapter state, then

$$\begin{aligned} M_i(\phi_{\text{final}}) - M_i^{\text{SFT}} = &\big[M_i(\phi_i^\star) - M_i^{\text{SFT}}\big] \\ &+ \big[M_i(\phi_{\text{final}}) - M_i(\phi_i^\star)\big]. \end{aligned} \tag{20}$$

Proof sketch. Add and subtract $M_i(\phi_i^\star)$:

$$\begin{aligned} M_i(\phi_{\text{final}}) - M_i^{\text{SFT}} = &M_i(\phi_{\text{final}}) - M_i(\phi_i^\star) \\ &+ M_i(\phi_i^\star) - M_i^{\text{SFT}}. \end{aligned} \tag{21}$$

Reordering terms gives Eq. 20. The first bracket in Eq. 20 measures the gain available at the task's own peak relative to the SFT reference. The second bracket records the contribution of subsequent stage transfer to the same task metric.

Proposition 4: compact checkpoint footprint. For one projection, the trainable adapter tensor footprint scales as

$$|\phi|_{\text{tensor}} = rd_{\text{in}} + d_{\text{out}}r + |\sigma|, \tag{22}$$

where $|\sigma|$ denotes the ANL parameter count. A full projection checkpoint scales as

$$|W_0|_{\text{tensor}} = d_{\text{out}}d_{\text{in}}. \tag{23}$$

When $r \ll \min(d_{\text{in}}, d_{\text{out}})$ and $|\sigma|$ is small relative to $d_{\text{out}}d_{\text{in}}$, adapter-state rollback stores substantially less tensor state than full-model rollback.

Proof sketch. The factor A has rd_{in} parameters, B has $d_{\text{out}}r$ parameters, and ANL contributes $|\sigma|$ parameters. Since the frozen base projection is not stage-dependent, it does not need to be duplicated across rollback checkpoints. Summing this expression across adapted projections gives the model-level adapter checkpoint size, plus small metadata needed to identify target modules, tensor shapes, and the ANL configuration.

Mathematical ingredient	How it is used in the derivation
Low-rank factorization	Represents the stage-dependent update as $B\bar{A}$ rather than as a full dense matrix.
Rank inequality	Establishes $\text{rank}(B\bar{A}) \leq r$.
Linear map equivalence	Shows $W_0x + \Delta Wx = (W_0 + \Delta W)x$ for fixed adapter state.
Weight-space nonlinear transform	Allows σ to enrich the adapter factor while preserving input-linearity after the transformed weight is formed.
Checkpoint state sufficiency	Uses frozen W_0 to show that restoring ϕ reconstructs the stage-dependent projection.
Telescoping identity	Adds and subtracts $M_i(\phi_i^\star)$ to account for peak gain and subsequent stage transfer.
Parameter-count bound	Compares $rd_{\text{in}} + d_{\text{out}}r + \|\sigma\|$ with $d_{\text{out}}d_{\text{in}}$ to explain compact rollback storage.

Table 18: **Mathematical ingredients used in the derivation.** The proof chain supports the structural properties used by AuroSFT.

H Symbol-to-Code Mapping

Table 19 maps the notation used in the main paper to implementation artifacts in the anonymous code release. The mapping is included to make it easier for reviewers to audit whether the equations describe the code path that produced the reported results.

Paper symbol or concept	Implementation artifact	Audit note
A, B	`project/sft/adapters/aurora.py`	Low-rank adapter factors attached to frozen projections.
$\bar{A} = \sigma(A^\top)^\top$	`project/sft/adapters/aurora.py`	Weight-level nonlinear transformation, not activation-level nonlinear mixing.
$\Delta W = (\alpha/r)B\bar{A}$	`project/sft/adapters/aurora.py`	Mergeable linear update applied to the frozen projection.
$\phi = \{A, B, \text{ANL}\}$	`aurora_adapter.bin`, `aurora_metadata.json`	Adapter-state checkpoint payload.
Rollback	`project/sft/run_mSFT.sh`	Stage transition restores adapter checkpoint rather than full backbone.
Peak analysis	`project/sft/analyze_history.py`	Reads validation histories and selects stage transition points.
Training and evaluation	`project/sft/train_eval.py`	Loads configs, trains adapters, evaluates benchmark accuracy.
Configuration schema	`project/sft/settings/configs.py`	Stores adapter and scheduler fields used by run configs.
Checkpoint pruning	`project/sft/scripts/prune_low_score_checkpoints.py`	Removes low-score or incomplete checkpoints while preserving selected states.

Table 19: **Symbol-to-code mapping.** This table links the manuscript equations and stage operations to the released implementation.

I Reproduction Commands and Repository Manifest

The anonymous release is intended to provide the code and configuration surface needed to reproduce the experiments once users supply local base-model weights and datasets. Table 20 gives representative command templates. Reviewers should edit only local model paths when reproducing on a different machine.

The minimal reproduction path is config-driven. After installing the listed environment, a reviewer sets local base-model and dataset paths in the selected config, launches the corresponding script,

and reads the generated stage summaries, validation histories, and benchmark accuracy files from the experiment output directory. Table 12 is recomputed from the selected best checkpoints, and Table 15 from the per-benchmark accuracy records.

Experiment	Config name	Launch command
3B conservative AuroSFT	`opt_scheduler_conservative_2epoch`	`bash scripts/run_experiment.sh opt_scheduler_conservative_2epoch`
0.5B rank16 AuroSFT	`opt_qwen25_05b_constant5_rank16`	`bash scripts/run_experiment.sh opt_qwen25_05b_constant5_rank16`
1.5B rank16 AuroSFT	`opt_qwen25_15b_constant5_rank16`	`bash scripts/run_experiment.sh opt_qwen25_15b_constant5_rank16`
3B pure MSFT repro	`msft-full-qwen25-local.json`	`CONFIG_FILE=configs/msft-full-qwen25-local.json bash run_mSFT.sh`
7B pure MSFT repro	`msft-full-qwen25-7b-local.json`	`CONFIG_FILE=configs/msft-full-qwen25-7b-local.json bash run_mSFT.sh`

Table 20: **Representative reproduction commands.** The public release uses config-driven launches, and local model paths must be set before running.

Repository component	Release policy
Core code	Included: adapters, training/evaluation entry point, config schema, and stage runner.
Experiment configs	Included: representative AuroSFT and mSFT reproduction configs.
Environment manifests	Included: requirements and inspected package versions.
Documentation	Included: README, experiment matrix, setup notes, and artifact policy.
Model weights	Excluded. Users download base models separately.
Checkpoints	Excluded. Large generated artifacts are not committed.
Raw logs and predictions	Excluded from the repository. Summarized evidence is reported here.
Secrets and host paths	Excluded. No credentials or private absolute paths are released.

Table 21: **Repository manifest.** The anonymous repository is designed for code review and reproducibility without shipping large or private artifacts.