MoEcraft: Trajectory-Based Sparse Upcycling of Instruction-Tuned Small Language Models

Anonymous authors
Paper under double-blind review

Abstract

We study sparse upcycling of instruction-tuned small language models into mixture-of-experts (MoE) models without returning to large-scale continual pre-training. MoEcraft reuses checkpoints from a single on-policy reinforcement-learning (RL) trajectory to initialize experts, followed by lightweight joint router–expert calibration. It combines all-linear low-rank adaptation, feed-forward adapters from eight checkpoints, and top-2 routing with calibration on 512 examples. In the canonical Qwen3-1.7B run, the chain scores 52.35 at the dense RL endpoint, 51.84 after sparse assembly, and 56.95 after calibration on Overall13, the mean over 13 benchmarks. Across three RL trajectories, calibration adds 3.60–5.11 points, and final performance averages 56.14 ± 0.72 (mean $\pm$ sample standard deviation). A single-run supervised fine-tuning trajectory control using the same sparse construction and calibration gains 2.50 points from assembly but loses 0.64 during calibration, driven by a summarization regression. The stage-wise pattern also recurs with a second RL optimizer and on Llama and Falcon models. The tested pipelines complete within one day per model on a single 96GB GPU. Under a fixed serving setup, the 1.7B-derived MoE achieves $1.54\times$/$1.22\times$ the output throughput of released dense Qwen3-4B at batch sizes 1/4 with 2.76B active parameters, but has 9.10B total parameters and a larger resident footprint (30 versus 20 GiB). These results characterize the utility and performance–footprint trade-offs of reusing RL trajectories for post-alignment sparse upcycling in the tested settings.

1 Introduction

Scaling laws have established that model capacity is a central driver of language-model performance (Kaplan et al., 2020). Mixture-of-Experts (MoE) architectures (Fedus et al., 2022; Jiang et al., 2024; Yang et al., 2025; DeepSeek-AI, 2024) are attractive because they decouple total capacity from per-token computation. However, the pre-training-oriented route to MoE is impractical for groups whose compute is limited to a single high-VRAM workstation GPU: building MoEs from scratch requires cluster-scale pre-training (Fedus et al., 2022; DeepSeek-AI, 2024). Sparse Upcycling (Komatsuzaki et al., 2023) initializes from a dense checkpoint but still requires substantial continual pre-training (CPT). Parameter-efficient adapter-MoE methods do not carry that cost and are the closer comparison; Section 2 differentiates from them.

A narrower but more practical question. These groups must build and refresh specialized models on a fixed hardware budget. We therefore ask a narrower question: given a *fully instruction-tuned* small language model (SLM), can we convert it into a sparse MoE *without* reverting to pre-training or performing large-scale CPT? We call this regime the *Post-Post-Training* setting: upcycling after post-training (instruction tuning) is complete, rather than from a pre-trained checkpoint. This setting—operating on an already aligned model—has two practical consequences. First, the base model already encodes aligned capabilities, so we test whether a small calibration set suffices rather than assuming a large one is needed (Overall13 shows diminishing returns beyond N_a=512, Section 4.5). Second, starting from an instruction-tuned checkpoint avoids returning the model to broad-corpus continual pre-training (Kirkpatrick et al., 2017; Luo et al., 2023).

Problem and approach. We study post-alignment sparse upcycling: converting an instruction-tuned small language model into a mixture-of-experts model without returning to large-scale continual

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.

pre-training. MoEcraft reuses states saved during a single on-policy RL run as expert initializations and jointly calibrates the assembled sparse model on a small dataset. All-linear low-rank adaptation produces the trajectory; feed-forward adapters from eight checkpoints initialize separate experts, while the attention adapters are combined in the shared attention module using the procedure specified in Section 3.

Contributions. We present a concrete post-alignment upcycling method and characterize its stage-wise behavior under a workstation-scale budget. Our central evidence combines a supervised-trajectory source control with three RL trajectories, cross-family evaluation, and measured training and serving trade-offs. These experiments show where the pipeline improves performance and where task-level and memory costs remain. Prior work already reuses saved checkpoints for expert construction (Sukhbaatar et al., 2024; Hui et al., 2024); our focus is the use of one task-verifiable on-policy RL trajectory after instruction tuning and its interaction with lightweight sparse calibration.

2 Related Work

Table 1 compares MoEcraft with representative upcycling strategies along six attributes that can be read off each paper: where it starts, where its experts come from, how they are constructed, what is trained after assembly, what that training updates, and what budget is reported.

Pre-training and large-scale MoE. Switch (Fedus et al., 2022), Mixtral (Jiang et al., 2024), DeepSeekMoE (Dai et al., 2024) and OLMoE (Muennighoff et al., 2024) build MoEs at pre-training scale, and MegaBlocks (Gale et al., 2023) addresses the training-system efficiency of doing so. These complement rather than substitute the workstation-level setting we study.

Dense-to-MoE upcycling and refactorization. Sparse Upcycling (Komatsuzaki et al., 2023) duplicates FFN modules and continues pre-training, so the backbone is updated at full capacity and adapter-subspace sizing (L1) does not apply. MoEfication (Zhang et al., 2022) partitions FFN neurons of a trained dense model statically. Branch-Train-MiX (Sukhbaatar et al., 2024) continues pre-training parallel branches on separate domain corpora, so expert diversity comes from manually specified domain boundaries. It reports 926 GPU-days (22,224 GPU-hours) of total training from its seed model over 533B tokens, of which 23% is the MoE-training stage. Our most expensive complete pipeline is 8.33 GPU-hours on one GPU (Section D.7). The two figures measure different things, continued pre-training from a seed model versus upcycling an already-aligned checkpoint, and we do not read their ratio as a speedup. Read-ME (Cai et al., 2024) refactorizes a dense LLM into a router-decoupled MoE with system co-design for inference efficiency, but operates before instruction alignment.

Parameter-efficient adapter-MoE and checkpoint reuse. LoRAMoE (Dou et al., 2023) and PESC (Wu et al., 2024) graft MoE-style adapters onto a dense backbone during instruction tuning. UpIT (Hui et al., 2024) is closest to our setting: it takes experts from intermediate instruction-tuning checkpoints, expands the expert set by merging the two most dissimilar experts, pre-optimizes a routing vector per expert on expert-specific data, and then fine-tunes the assembled MoE. It has both FFN-based and LoRA-based variants. UpIT starts from a dense pre-trained model and splits instruction tuning around the conversion. MoEcraft instead starts from an instruction-tuned model, draws its experts from a single on-policy RL trajectory with task-verifiable rewards, and spends its post-assembly budget on joint router–expert calibration over 512 examples. Model Soups (Wortsman et al., 2022) also reuses checkpoints, but averages them into one dense model instead of assigning them to separate experts of a sparse architecture.

Trajectory source. Our experts come from RL with task-verifiable rewards (Shao et al., 2024; Gao et al., 2025), where rewards are computed from task correctness rather than from a learned reward model as in standard RLHF (Ouyang et al., 2022). Offline preference optimization (e.g. DPO (Rafailov et al., 2023)) provides no rollout-level exploration. Section 4.4 reports the trajectory-source comparison, and Section C.1 the corresponding initialization controls.

Table 1: Objective attribute comparison against representative upcycling strategies. Entries state what each method does, not which performs better. Reported budgets are measured on different axes and are not directly comparable; "—" marks an attribute the source does not state in terms comparable to the others. The MoEcraft budget is 8,192 trajectory prompts plus 512 calibration examples; the 512 is the post-assembly part alone.

Method	Starting ckpt.	Expert source	Construction	Post-assembly training	Trainable scope	Reported budget
Sparse Upcycling	Pre-trained	Duplicated FFN	Identical copies	Continued pre-training	All parameters	—
MoEfication	Trained dense	FFN neuron clusters	Static partition	Selector fitting	Router	—
Branch-Train-MiX	Pre-trained seed	Domain-CPT branches	One branch per domain	MoE fine-tuning	All parameters	533B tokens
UpIT	Pre-trained	Intermediate SFT ckpts	Merge dissimilar pairs	MoE fine-tuning	FFN or LoRA variant	—
Model Soups	Fine-tuned ckpts	Final ckpts	Weight averaging	None	—	—
MoEcraft (ours)	**Instruction-tuned**	**On-policy RL trajectory**	**One FFN copy per ckpt.**	**Joint router–LoRA calib.**	**Router + LoRA**	**8,192 + 512**

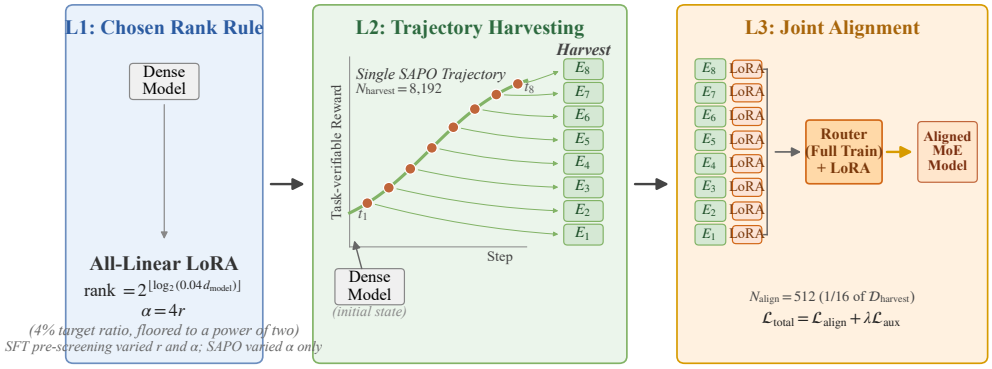

Figure 1: Pipeline grouped by conditions: L1 (subspace), L2 (SAPO trajectory + harvest), L3 (joint router–LoRA align). Backbone adaptation uses low-rank updates; post-assembly calibration jointly trains the router and expert adapters. Assembly itself does not perform gradient updates.

3 METHOD

MoEcraft comprises three workflow components: all-linear low-rank adaptation sized as a fraction of the model's hidden size rather than fixing one rank across scales (L1), trajectory-derived expert initialization (L2), and joint router–expert calibration after sparse assembly (L3). We evaluate their behavior through stage-wise measurements and the controls described below. Figure 1 shows the full pipeline; Figure 2 contrasts it with standard sparse upcycling.

3.1 PRELIMINARIES

An MoE layer with N experts $\{E_i\}$ and router $G(\cdot)$ produces $\mathbf{y} = \mathbf{x} + \sum_{i \in \mathcal{T}(\mathbf{x})} G(\mathbf{x})_i \, E_i(\mathbf{x})$, with $\mathcal{T}(\mathbf{x}) = \text{Top-k}(\text{Softmax}(\mathbf{x}\mathbf{W}_g))$. We use N=8, k=2. Each expert is a full copy of the feed-forward block, so a converted model carries N such blocks over one shared embedding and attention body and routes k of them per token. LoRA (Hu et al., 2022) constrains weight updates to $\mathbf{W}_0 + \frac{\alpha}{r}\mathbf{B}\mathbf{A}$ with $r \ll d$.

3.2 L1: CAPACITY-AWARE ADAPTIVE SUBSPACE (STAGE 1)

Prior work studies fine-tuning through low-dimensional reparameterizations (Aghajanyan et al., 2021); we use the following rank rule as an empirical budget choice. We apply LoRA to *all* linear projections with $r = 2^{\lfloor \log_2(0.04\, d_{\text{model}}) \rfloor}$ and $\alpha = 4r$, so the 4% ratio is a target that is discretized down to a power of two. Two properties motivate this, both validated in Section 4.3. First, restricting LoRA to the FFN leaves attention frozen under RL, and training then collapses in our setup (Section 4.3). Second, we searched at and below this ratio under two training paradigms and adopted the largest rank in the grid; nothing above it was tried, and Table 6 reports where that point ranks inside the grid. We treat this as an empirical default in the current setting rather than a universal law.

3.3 L2: TRAJECTORY-DERIVED EXPERT INITIALIZATION (STAGES 2 AND 3)

Checkpoint source. We obtain expert initializations from checkpoints of a single sampled on-policy RL run with task-verifiable rewards; what we test is whether the source changes the outcome. We use Soft Adaptive Policy Optimization (SAPO) (Gao et al., 2025) as the default and Group Relative Policy Optimization (GRPO) (Shao et al., 2024) as a second tested optimizer. Offline preference-optimization methods (e.g. DPO) optimize a static pairwise objective without rollout-level exploration. Section 4.4 compares the resulting stage-wise behavior with an SFT trajectory; the comparison does not establish a universal ordering of learning objectives.

Stage 2: Trajectory generation. A single LoRA adapter is trained on $\mathcal{D}_{\text{harvest}}$ ($N_h = 8{,}192$) under the SAPO objective of Gao et al. (2025). Rewards are computed from task correctness (no learned reward model): general tasks use exact-match $R_{\text{match}} \in \{0, 1\}$, programming tasks use an execution-based reward. Checkpoints are stored at eight sequential steps spanning the run.

Stage 3: Harvesting. Each stored checkpoint provides the FFN adapter for one expert, $E_i \leftarrow \theta_{\text{ffn}}^{(t_i)}$. The shared attention body takes the mean of the checkpoints' effective LoRA updates, $\Delta\mathbf{W}_{\text{attn}} = \frac{1}{N}\sum_i \Delta\mathbf{W}_{\text{attn}}^{(t_i)}$ with $\Delta\mathbf{W} = \frac{\alpha}{r}\mathbf{BA}$, rather than averaging the $\mathbf{A}$ and $\mathbf{B}$ factors separately; the two differ because $\text{mean}(\mathbf{B}_i\mathbf{A}_i) \neq \text{mean}(\mathbf{B}_i)\,\text{mean}(\mathbf{A}_i)$. We use sequential checkpoints spanning the run; additional selection and initialization controls are reported in Section C.1.

3.4 L3: SPARSE-AWARE JOINT CALIBRATION (STAGE 4)

Sparse conversion introduces a transition gap, measured in Section 4.2. We close it by jointly optimizing the router ϕ and LoRA parameters θ on $\mathcal{D}_{\text{align}}$, a $1/16$ downsample of $\mathcal{D}_{\text{harvest}}$ ($N_a = 512$). Stage 4 optimises the SAPO objective on this alignment set, combined with a standard load-balancing term $\lambda_{\text{aux}} \sum_i f_i P_i$, where f_i is the routed-token fraction and P_i the mean gate probability for expert i. This Stage-4 objective is held fixed across trajectory sources: the matched SFT-source run of Table 3 differs from the canonical pipeline only in the Stage-2 objective and the checkpoints it yields. The dense-copy controls of Section C.1 are the exception and are reported under their recorded alignment names. A small alignment set is empirically adequate in the tested range (diminishing returns beyond N_a=512, Section 4.5).

3.5 COMPLEXITY

Algorithm 1 states the pipeline in full. We bound training cost by measurement rather than by an analytic ratio. Our N_h+N_a=8,704 counts prompts, not tokens, and Stage 2 samples completions per prompt. We therefore do not convert it into a FLOP ratio against continued pre-training, and do not quantify the factor by which our budget is smaller. Restricting updates to the adapter subspace saves optimizer state and backbone weight gradients, but Stage 2 still rolls out and backpropagates through the full dense model, so cost is not proportional to the adapter fraction. The measured pipeline completes within a day on one 96GB workstation GPU, at 8.33 GPU-hours with a 92.9 GiB Stage-4 peak (Sections D.7 and D.8).

Table 2: Main results on Qwen3 (top) and the canonical Qwen3-1.7B full stage chain (bottom). Overall13 and category aggregates. Total/active parameters are explicit so the comparison is a footprint–efficiency trade-off rather than "smaller beats larger".

Model	Total	Active	General7	Math2	Code2	Summary2	Overall13
Qwen3-0.6B (Base)	0.60B	0.60B	38.4	37.7	57.8	19.9	38.4
MoEcraft (0.6B)	2.45B	0.86B	49.8	49.3	65.0	30.7	49.1
Qwen3-1.7B (Base)	1.70B	1.70B	46.9	40.6	59.7	19.7	43.7
Qwen3-1.7B (Dense SAPO)	1.70B	1.70B	52.7	63.3	62.2	30.4	52.3
Qwen3-1.7B Harvest-only	9.10B	2.76B	51.8	63.7	61.2	30.9	51.8
MoEcraft (1.7B)	9.10B	2.76B	**58.8**	**69.8**	**65.1**	**29.4**	**57.0**
Qwen3-4B (Base)	4.00B	4.00B	59.5	43.1	61.0	21.7	51.4
MoEcraft (4B)	22.83B	6.69B	70.3	79.1	65.9	34.9	65.5

4 EXPERIMENTS

4.1 SETUP

Benchmarks and aggregates. We use 13 benchmarks across four categories: General (7: ARC-C, ARC-E (Clark et al., 2018), GPQA-Main, GPQA-Diamond (Rein et al., 2023), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021a), MMLU-Pro (Wang et al., 2024)), Math (2: GSM8K (Cobbe et al., 2021), Math500 (Hendrycks et al., 2021b; Lightman et al., 2023)), Code (2: MBPP (Austin et al., 2021), CRUX-O (Gu et al., 2024)), and Summary (2: SAMSum (Gliwa et al., 2019), XSum (Narayan et al., 2018)). We report five aggregates: General7, Math2, Code2, Summary2, and **Overall13** (mean of all 13 scores, treating summary metrics on the same 0–100 scale as accuracy metrics by multiplying by 100). Unless otherwise stated, all reported scores in the main body use Overall13 as the primary summary.

Models. Base models are instruction-tuned checkpoints. We use Qwen3-0.6B, Qwen3-1.7B, Qwen3-4B, and Qwen3-32B (Yang et al., 2025), with Qwen3-1.7B as the canonical setting in which we run every ablation. For cross-family tests, we additionally use Llama-3.2-1B-Instruct, Llama-3.2-3B-Instruct (Grattafiori et al., 2024), Falcon3-1B-Instruct, and Falcon3-3B-Instruct (Falcon-LLM Team, 2024), with Llama-3.1-8B-Instruct and Falcon3-10B-Instruct as larger dense comparators.

Training configuration. Rank and α follow the rule of Section 3.2 (r=64, α=256 at d_{model}=2048), all-linear target, N=8 experts, k=2. Harvesting data $N_h = 8{,}192$; alignment data $N_a = 512$ ($1/16$ downsample of $\mathcal{D}_{\text{harvest}}$). All experiments use a single NVIDIA RTX PRO 6000 Blackwell Max-Q (96GB VRAM) workstation GPU.

4.2 MAIN RESULTS

Table 2 summarizes the main comparison across Qwen3 scales, with total and active parameter counts explicit, plus the full stage chain on the canonical Qwen3-1.7B setting. The harvest-only regression ($52.35 \rightarrow 51.84$) identifies the sparse transition gap that Stage 4 joint calibration (L3) is designed to close. Per-benchmark scores across all Qwen3 scales (0.6B–32B) are in Section B; cross-family results are reported separately in Section 4.6.

4.3 L1 VALIDATION: SUBSPACE CAPACITY

All-linear vs. FFN-only. Restricting LoRA to FFN layers destabilizes training under the RL pipeline: on Qwen3-0.6B, FFN-only collapses to Math 12.4 (unstable) while all-linear is stable at Math 45.2. Here "Math" is the legacy single-benchmark score of the L1 sweep, not the Math2 aggregate. We interpret this as indirect evidence that for small models the attention pathway is not effectively redundant under sparse-routing adaptation, so freezing it leaves the routing-adjacent signal ill-conditioned.

Table 3: Trajectory-source comparison on Qwen3-1.7B (Overall13). Sparse construction and post-assembly calibration are held fixed. One SFT trajectory and three RL trajectories are reported. ΔL3 is Final minus Harvest-only. All deltas are computed from unrounded scores and can therefore differ by 0.01 from the difference of the rounded columns. Dense endpoint quality and trajectory-generation cost are not matched.

Source	Seed	Dense endpoint	Harvest-only	Final	ΔL3
SFT	2025	43.92	46.42	45.78	−0.64
SAPO (canonical)	2025	52.35	51.84	56.95	+5.11
SAPO	2026	52.68	51.79	55.91	+4.12
SAPO	2027	52.58	51.97	55.58	+3.60

Rank ratio. We set the target ratio to γ=0.04 and searched at and below the resulting rank under two training paradigms; Table 6 gives the grid and what each paradigm varied, and Section A.1 the hyperparameters.

4.4 Trajectory Source and Stage-wise Calibration

We test whether the source of expert initialization changes the response to the same sparse-construction and post-assembly calibration protocol. Motivated by prior comparisons of SFT and RL generalization (Chu et al., 2025), we compare an SFT trajectory with three SAPO trajectories. Their dense endpoints and trajectory-generation costs differ, so the comparison characterizes source-dependent behavior rather than isolating the causal effect of the RL objective.

Calibration improves Overall13 on all three RL trajectories, yielding a final mean of 56.14 ± 0.72 (mean $\pm$ sample standard deviation). The single SFT trajectory gains 2.50 points from harvesting but loses 0.64 during calibration. This aggregate decline combines improvements in General7, Math2 and Code2 (+0.42, +3.55 and +0.69) with a Summary2 regression (−9.89), rather than a uniform loss of capability. The separately tuned 150k-sample SFT outcome baseline reaches 51.40 and is distinct from this 8,192-prompt stage-wise control.

RL algorithm generality. Replacing SAPO with GRPO reproduces the same dense-to-harvest-to-final pattern ($52.05 \rightarrow 51.31 \rightarrow 56.66$); the full breakdown is in Table 13. This is evidence from a second optimizer, not a universal ranking of RL and SFT algorithms. Adding an explicit negative-reward term does not improve the final sparse model (55.57 against 56.95; Section C.3).

Initialization and selection. Additional comparisons of expert initialization and checkpoint selection are reported in Section C.1. A single-terminal-adapter interpolation variant also performs well, so these results do not establish that multiple distinct checkpoints are necessary.

4.5 L3 Validation: Sparse-Aware Joint Calibration

Router-only vs. joint calibration. Running only the router during Stage 4 does not close the transition gap introduced by sparse conversion. In our Qwen3-1.7B setup, router-only alignment reaches Math2 63.4, Code2 61.5, General7 52.1, whereas joint router–LoRA alignment (MoEcraft) reaches Math2 69.8, Code2 65.1, General7 58.8 (+6.4/+3.6/+6.7), indicating that the expert subspace benefits from co-adapting with the routing policy.

Alignment-size sensitivity. Stage-4 scores at 256, 512 and 1,024 examples are 55.65, 56.95 and 57.31 on Overall13. The gain decreases from 1.30 to 0.36 across the two doublings, motivating 512 as a compute-aware operating point. This single-seed sweep does not distinguish capability recovery from acquisition, or exclude optimization and evaluation limits as explanations for the diminishing return; we state the summarization trade-off in Section 6. Full per-aggregate breakdown in Section C.4.

Table 4: Cross-family full chains (Overall13) with MoE total / active parameter footprint (N=8, top-2). The Dense Base → Dense SAPO → Harvest-only → Final pattern appears on both Llama-3.2 and Falcon3 families. For the larger dense comparator rows, the Base column is the released Instruct checkpoint and the Dense SAPO column is the same dense model after Stage 2; Harvest-only and Final are not run.

Model	Base	Dense SAPO	Harvest-only	Final	Total / Active (B)
Llama-3.2-1B	32.01	41.10	39.52	42.60	6.64 / 1.81
Llama-3.2-3B	45.20	50.40	49.70	53.02	17.80 / 5.11
Falcon3-1B	32.06	39.23	35.74	40.32	7.34 / 1.91
Falcon3-3B	48.83	54.92	54.19	58.01	16.08 / 4.87
Llama-3.1-8B dense comparator	55.08	57.68	–	–	8.0 / 8.0
Falcon3-10B dense comparator	55.60	60.26	–	–	10.0 / 10.0

Stage-4 seed robustness. Repeating only Stage 4 with three seeds (same trajectory and harvested experts) gives Overall13 56.97 ± 0.04 (Final) vs. 50.96 ± 0.08 (final-checkpoint copy + SAPO align), so the final-checkpoint-copy gap is not a Stage-4 seed artifact (Section D.5).

Top-k at inference. Beyond the L3 mechanism, inference-time top-k shows diminishing returns: at N=8, k=2/3/4 give 57.0/57.8/58.1 while active parameters scale roughly linearly with k. k=2 is our default (Section C.5).

4.6 Generalization Beyond Qwen

To test whether the stage pattern reproduces outside Qwen, we run the full MoEcraft chain on two additional instruction-tuned families and compare to larger dense comparators from the same families. Table 4 reports the full chain, and Figure 3 plots it against the canonical Qwen3-1.7B chain for all five models.

Interpretation. Falcon3-3B Final (58.01) surpasses Falcon3-10B Instruct (55.60) with 4.87B vs. 10B active. The same 10B model after Stage 2 reaches 60.26 in the same table, so the comparison is against the released instruct checkpoint rather than the strongest dense variant we ran. Llama-3.2-3B Final (53.02) trails Llama-3.1-8B Instruct (55.08) on Overall13 but is close and exceeds it on General7. The two families therefore point in different directions, and we read the pair as a footprint–efficiency trade-off rather than a claim that the upcycled models beat larger dense ones.

4.7 Measured Training and Serving Trade-offs

For the instrumented Qwen3-1.7B RL runs, Stages 2–4 require 5.46 and 7.52 GPU-hours; including the recorded stage evaluations and routing measurements gives 6.16 and 8.33 GPU-hours. These totals exclude the separately reported L1 attachment. Under the controlled serving setup, the canonical MoE reaches $1.54\times$/$1.22\times$ the output throughput of released dense Qwen3-4B at batch sizes 1/4, but its resident footprint is 30.0 versus 20.0 GiB. The comparison is therefore a performance–footprint trade-off rather than a memory saving. Detailed protocols and costs are in Sections D.4 and D.7; the earlier unlogged harness is retained in Section D.3 as a per-token compute proxy.

The released Qwen3-4B accuracy comparator scores 51.41 on Overall13, whereas the separately evaluated dense Qwen3-4B after SAPO reaches 61.40. The 1.7B-derived MoE therefore does not outperform every trained 4B comparator. We keep serving measurements tied to the checkpoints actually measured rather than assigning their runtime to the separately trained comparator. Routing diagnostics for the assembled models, under the two aggregation orders, are reported in Section D.2.

5 Discussion

The stage-wise results show that sparse assembly and post-assembly calibration play different roles: assembly slightly lowers the canonical dense RL score, while calibration produces the final gain.

Across the tested sources, an on-policy RL run and an SFT run responded differently to the same calibration, giving different task-level responses. This supports the practical use of RL-trajectory reuse in the evaluated pipeline, not a claim that every source or initialization requires multiple distinct checkpoints.

The ablations distinguish adaptation scope, checkpoint source, and post-assembly training. The rank setting is an empirical operating point; we did not compare converted models across ranks. Both tested RL optimizers reproduce the stage-wise pattern (Section 4.4). Joint calibration improves on router-only training in the tested configuration (Section 4.5), but the score changes and routing statistics do not by themselves identify whether the updates recover, acquire, or redistribute capabilities.

The supplementary initialization and selection controls include a competitive single-terminal-adapter interpolation variant (Section C.1). Together with the single-run SFT control and the absence of fully reproduced external pipelines, this limits causal and general comparative claims. Routing entropy and divergence describe expert utilization after assembly; they do not directly measure functional modes visited during training or establish that routing diversity causes the performance gain.

6 LIMITATIONS

Scope of empirical claims. The 4%-target all-linear rule is an empirical default at the tested scales rather than a universal law, and the 32B result is Dense-SAPO only because full MoE upcycling at 32B was not run under the single-GPU setup. Starting from an instruction-tuned checkpoint does not, by itself, guarantee preservation of alignment or prevent forgetting; we provide no dedicated safety or alignment-retention measurements. Expert diversity depends on the on-policy trajectory, and we measure it only after harvesting (routing entropy and pairwise divergence, Section D.2); we have no criterion that predicts from the training run alone whether a given trajectory will yield non-collapsed experts. The rank search of Table 6 was run on the dense backbone only, and the conversion itself was run at a single rank per model. We therefore cannot say what the chain would do at a smaller subspace: the table bounds the searched range but does not isolate the effect of rank after conversion. We did not re-run any prior pipeline, so Table 1 compares stated designs rather than measured outcomes. Although we validate the canonical chain on Llama and Falcon in addition to Qwen, broader architectural families (e.g. Mistral, Gemma) and non-Qwen rank-ratio sweeps remain future work.

Compute, efficiency, and robustness. Active parameters are a per-token compute proxy. Total parameters grow with N and all experts stay resident at inference, so the measured footprint is 30.0 GiB against 20.0 GiB for the dense 4B baseline (Section D.4); we therefore make no memory-efficiency claim. Throughput is reported at two batch sizes under one fixed serving setup and does not constitute a full deployment latency model (Sections D.3 and D.4); sharding, quantization, and offloading remain engineering directions. The trade-off is two-axis rather than a ranking: Section D.8 reports the per-stage cost, which variant to deploy under which constraint, and the conversion memory requirement separately from the serving one. We provide Stage-4 alignment seed robustness (Section D.5) and full-pipeline multi-seed over three SAPO trajectory seeds (56.14 ± 0.72 Overall13; Section D.6), but only at the Qwen3-1.7B scale. Cross-family variance therefore remains untested, as does the summarization trade-off we observe on Qwen3-1.7B (29.41 vs. Dense-SAPO 30.37). Summary2 is likewise lowest at our default N_a=512 (29.41) and higher at 256 (30.88) and 1024 (30.46). The 512 default is therefore selected on Overall13 at a measured cost on summarization rather than because it is uniformly best (Section C.4).

7 CONCLUSION

MoEcraft reuses checkpoints from one on-policy RL trajectory to initialize sparse experts after instruction tuning and jointly calibrates the assembled model on 512 examples. On Qwen3-1.7B the canonical chain runs 52.35 at the dense RL endpoint, 51.84 after sparse assembly and 56.95 on Overall13 after calibration, with a three-trajectory final mean of 56.14 ± 0.72, within a day per (family, scale) pair on one 96GB workstation GPU. The stage-wise source comparison, the cross-family evaluations, and the measured training and serving costs characterize the utility of this workstation-scale post-alignment pipeline and its task-level and memory trade-offs. The controls

support an empirical dependence on expert initialization and joint calibration in the tested settings; they do not establish that measured routing diversity uniquely causes the gain, or that the same relationship holds for every SFT or RL trajectory. These results support RL-trajectory reuse in the tested setting without establishing universal necessity of its individual design choices. We frame the outcome as a footprint–efficiency trade-off rather than "smaller beats larger," and view the framework as one path to making sparse-architecture research feasible for compute-constrained groups.

AI USE STATEMENT

Among the tasks for which disclosure is required, we used generative AI tools for data processing. Experimental results were aggregated, and later re-aggregated and cross-checked against the raw measurement records, with LLM assistance, and every number reported in this paper was verified by the authors against those records. The research question, the method, the pipeline, and the training and evaluation code are the authors' own work, and generative AI tools were not used to implement the method. The authors designed every experiment reported here, including the additional controls added during revision, and ran them. The authors also chose which experimental comparisons to report and set the interpretation and scope of the resulting claims. Generating synthetic data sets, proving theorems, formulating mathematical claims, and translation are not applicable to this work.

Among the tasks for which disclosure is recommended, we used generative AI tools for drafting and editing, for retrieval and discovery, and for reviewing the manuscript. The abstract and selected explanatory passages were drafted and edited with LLM assistance, as were reference formatting and the layout of the figures; the data underlying every figure are the authors' own. The tools were used to survey related work and bibliographic records, to check reported numbers against the measurement records and the released files, to check wording and readability, and to flag statements that the cited evidence did not support. The authors decided every resulting change. Auxiliary scripts used only to audit this manuscript—recomputing reported table values from the released measurement files and re-verifying bibliographic metadata—were also LLM-assisted.

We have reviewed all AI-assisted work. Bibliographic metadata for every reference was re-verified against the publisher or arXiv record of the version cited, and the numerical claims in this paper were re-derived from the released measurement files before being reported. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The reported adaptation settings and dataset sizes are given in Sections A.1 and 4.1; the run-level training specification of each stage is not part of this release. Hardware, software versions and the per-(family, scale) wall-clock budget are given in Section A.2, with per-stage cost and training-time peak memory in Section D.7. The inference measurement is reported under one fully specified serving configuration in Section D.4, and the earlier unlogged harness is retained separately in Section D.3 so the two are not conflated. Routing statistics are reported under two aggregation orders that Section D.2 states and keeps separate. We release manifest-level experiment descriptors, the per-benchmark results underlying every table reported at that granularity, and the raw per-stage cost ledger as supplementary material, together with scripts that recompute the reported values from them. Nine auxiliary ablation rows are reported only in aggregate and are marked as such in the released tables; for four of them the per-benchmark scores are released separately. Rows measured for configurations the paper no longer reports are retained with a note saying so rather than deleted. The supplementary material covers verification of the reported values; it does not contain the training repository, and exact command-line configurations will accompany a later code release.

ETHICS STATEMENT

This work converts publicly released instruction-tuned language models into sparse Mixture-of-Experts models and evaluates them on public benchmarks. It involves no human subjects, no personally identifying data, and no new data collection. The upcycled models inherit the capabilities and the failure modes of their base checkpoints, including any biases present in them; we did not conduct a safety or bias evaluation, and Section 6 states the evaluation scope. We release experiment descriptors and per-row results rather than model weights.

ACKNOWLEDGMENTS

Anonymous.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7319–7328, 2021.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Ruisi Cai, Yeonju Ro, Geon-Woo Kim, Peihao Wang, Babak Ehteshami Bejnordi, Aditya Akella, and Zhangyang Wang. Read-ME: Refactorizing LLMs as router-decoupled mixture of experts with system co-design. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.

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. *arXiv preprint arXiv:1803.05457*, 2018.

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. *arXiv preprint arXiv:2110.14168*, 2021.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, et al. Loramoe: Alleviate world knowledge forgetting in large language models via moe-style plugin. *arXiv preprint arXiv:2312.09979*, 2023.

Falcon-LLM Team. The Falcon 3 family of open models, 2024. URL `https://huggingface.co/blog/falcon3`.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient sparse training with mixture-of-experts. *Proceedings of Machine Learning and Systems (MLSys)*, 5, 2023.

Chang Gao, Chujie Zheng, Xiong-Hui Chen, Kai Dang, Shixuan Liu, Bowen Yu, An Yang, Shuai Bai, Jingren Zhou, and Junyang Lin. Soft Adaptive Policy Optimization. *arXiv preprint arXiv:2511.20347*, 2025.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79, 2019.

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

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2021b.

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.

Tingfeng Hui, Zhenyu Zhang, Shuohuan Wang, Yu Sun, Hua Wu, and Sen Su. Upcycling instruction tuning from dense to mixture-of-experts via parameter merging. *arXiv preprint arXiv:2410.01610*, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114(13):3521–3526, 2017.

Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints. In *International Conference on Learning Representations (ICLR)*, 2023.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. OLMoE: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1797–1807, 2018.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. GPQA: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Sainbayar Sukhbaatar, Olga Golovneva, Vasu Sharma, Hu Xu, Xi Victoria Lin, Baptiste Rozière, Jacob Kahn, Shang-Wen Li, Wen-tau Yih, Jason Weston, and Xian Li. Branch-train-MiX: Mixing expert LLMs into a mixture-of-experts LLM. In *Conference on Language Modeling (COLM)*, 2024.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024.

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 *International Conference on Machine Learning (ICML)*, volume 162, pp. 23965–23998. PMLR, 2022.

Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. Parameter-efficient sparsity crafting from dense to mixture-of-experts for instruction tuning on general tasks. *arXiv preprint arXiv:2401.02731*, 2024.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 4791–4800, 2019.

Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. MoEfication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 877–890, 2022.

A Experimental Setup and Compute

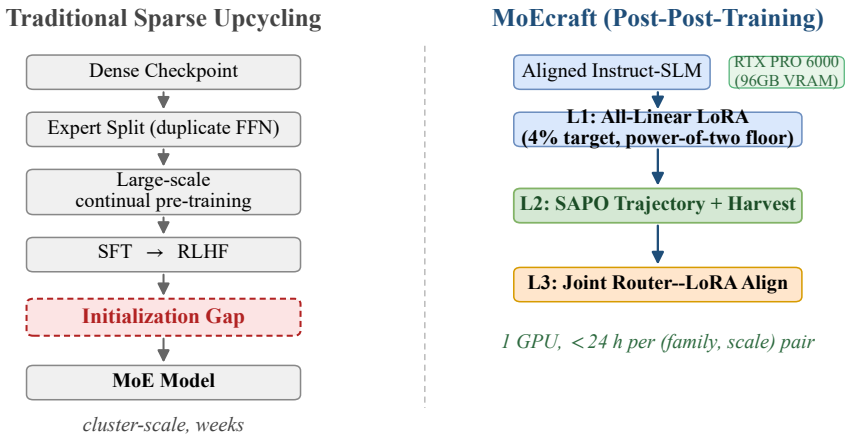

Figure 2: Standard sparse upcycling (left) vs. MoEcraft (right). MoEcraft satisfies L1/L2/L3 in the Post-Post-Training regime; the hardware and wall-clock budget is specified in Section 4.1. Parameter counts and scores for the converted models are in Table 2; the upcycled models carry a larger total footprint than the dense baselines they are compared against, so the comparison is a footprint–efficiency trade-off rather than "smaller beats larger".

Algorithm 1 MoEcraft Upcycling Pipeline

Input: Dense checkpoint θ_{dense} with hidden size d_{model}, dataset $\mathcal{D}_{\text{harvest}}$, experts $N{=}8$, target rank ratio $\gamma{=}0.04$
{L1: Subspace configuration}
$r \leftarrow 2^{\lfloor \log_2(\gamma\, d_{\text{model}}) \rfloor}$, $\alpha \leftarrow 4r$; attach LoRA to all linear projections
{L2: Trajectory generation and harvesting}
$\mathcal{T}_{\text{traj}} \leftarrow \textsc{Sapo_Train}(\theta_{\text{dense}}, \mathcal{D}_{\text{harvest}})$ with sequential checkpoint storage
For $i = 1, \ldots, N$: $\theta^{(i)} \leftarrow \textsc{LoadCheckpoint}(\mathcal{T}_{\text{traj}}, t_i)$
{L3: Sparse construction and joint alignment}
$\theta_{\text{attn}} \leftarrow \frac{1}{N}\sum_i \theta^{(i)}_{\text{attn}}$, $\quad \mathcal{E} \leftarrow \{\theta^{(i)}_{\text{ffn}}\}$
$\theta_{\text{MoE}} \leftarrow \textsc{InitMoE}(\theta_{\text{dense}}, \theta_{\text{attn}}, \mathcal{E}, G_\phi)$
Joint Align on $\mathcal{D}_{\text{align}}$: $(\phi, \theta) \leftarrow (\phi, \theta) - \eta\, \nabla \mathcal{L}_{\text{total}}$
Return: Sparse MoE θ_{MoE}

A.1 Hyperparameters and Data Sizes

Hyperparameter and dataset sizes are summarized in Table 5. Phase 1 (rank search) used an SFT-style pre-screening with 150,000 samples, which varied both r and α, and the SAPO pipeline with 8,192 samples, which varied α at the rank the pre-screening had selected. The adopted operating point is the largest searched rank with $\alpha{=}4r$.

Table 5: Hyperparameters (left) and dataset sizes (right).

Hyperparameter	Value	Phase	Size
Rank r	$2^{\lfloor \log_2(0.04\, d_{\text{model}}) \rfloor}$	Ph.1 Search (SFT)	150,000
α	$4r$	Ph.1 Search (RL)	8,192
Target	All linear	Ph.2 SAPO	8,192
Experts N	8	Ph.4 Align	512

Table 6 reports that search. Two things are visible. First, the adopted point is the ceiling of each grid rather than an interior optimum, and nothing above γ=0.04 was tried. Second, at the dense stage the adopted cell is not the best one for two of the three models: it ranks eighth of nine on both Qwen3-0.6B and Qwen3-1.7B, and first on Qwen3-4B. We report this rather than omit it. The dense stage measures an adapted dense model, whereas L1 concerns the subspace a set of experts is later carved from, so the two are not the same quantity. We did not run the conversion at any other rank, so this table cannot be read as evidence that the larger subspace helps after conversion either (Section 6).

Table 6: Phase-1 search. Overall13 after Phase-1 training on the dense backbone—not MoEcraft—so these numbers measure the adapted dense model, not expert specialization. The SFT pre-screening varied both r and α; the SAPO pipeline varied α only, at the rank the pre-screening had already selected, so its columns are populated on one row per model. † marks the adopted operating point, which is the largest rank in each grid; nothing above γ=0.04 was searched.

			SFT pre-screening (150k)			SAPO (8,192)		
Model	r	r/d_{model}	$\alpha{=}r$	$2r$	$4r$	$\alpha{=}r$	$2r$	$4r$
Qwen3-0.6B	dense base 38.41							
	8	0.781%	35.42	36.23	36.82	—	—	—
	16	1.562%	36.14	36.63	37.19	—	—	—
	32†	3.125%	36.78	37.02	36.03	43.16	44.46	45.36
Qwen3-1.7B	dense base 43.74							
	16	0.781%	43.01	43.96	45.05	—	—	—
	32	1.562%	44.22	45.03	45.12	—	—	—
	64†	3.125%	44.75	44.29	43.92	51.87	52.48	52.35
Qwen3-4B	dense base 51.41							
	16	0.625%	52.61	53.20	54.64	—	—	—
	32	1.25%	53.22	54.61	55.21	—	—	—
	64†	2.5%	53.92	55.47	55.66	59.49	60.17	61.40

A.2 Compute and Reproducibility

All experiments were run on a single NVIDIA RTX PRO 6000 Blackwell Max-Q (96GB VRAM) workstation GPU. The full pipeline (Stage 2 SAPO training, Stage 3 harvesting, Stage 4 alignment, and 13-benchmark evaluation) completes within a day per (family, scale) pair; per-stage wall-clock cost and training-time peak GPU memory were instrumented for Qwen3-1.7B and are in Section D.7. We release manifest-level experiment descriptors, the per-benchmark results underlying every table reported at that granularity, and the raw per-stage cost ledger in the supplementary material, together with scripts that recompute the reported values from them; nine auxiliary ablation rows are reported only in aggregate and marked as such, four of them with per-benchmark scores released separately. Exact command-line configurations will accompany the code release. Experiment identifiers (exp01–exp14) used internally correspond to: exp01 harvest-only, exp02 align-256, exp03 align-1024, exp04–exp06 the three final-checkpoint-copy controls, recorded under their original run names `dense_copy_only`, `dense_copy_sft_align` and `dense_copy_sapo_align`, exp07 harvest=last×8, exp08 harvest=4×2, exp09 harvest=best-8, exp10 negative reward, exp11 Llama-3.2-1B, exp12 Llama-3.2-3B, exp13 Falcon3-3B, exp14 Falcon3-1B.

B Full Benchmark Breakdown

Tables 7 and 8 report the per-benchmark scores for all Qwen3 scales from 0.6B to 32B. For 32B we report only the Dense (Base) and Dense (SAPO) settings, because the full MoE stage was not run at this scale under our single-GPU setup.

Table 7: Part 1: General-knowledge benchmarks across scales.

Scale	Configuration	ARC-C	ARC-E	GPQA-D	GPQA-M	HellaSwag	MMLU	MMLU-P
0.6B	Dense (Base)	49.3	65.5	29.8	25.5	41.1	40.8	16.5
0.6B	Dense (SAPO)	62.2	77.3	28.3	30.1	46.4	47.7	23.0
0.6B	**MoEcraft**	**68.9**	**82.8**	**33.1**	**30.9**	**52.2**	**52.8**	**28.1**
1.7B	Dense (Base)	72.5	82.3	19.7	25.0	54.5	53.9	20.7
1.7B	Dense (SAPO)	73.6	85.2	25.8	26.6	64.0	58.5	35.3
1.7B	Harvest-only	74.2	84.6	25.8	26.3	63.1	57.2	31.2
1.7B	**MoEcraft**	**84.0**	**93.1**	**31.1**	**28.9**	**69.6**	**65.5**	**39.7**
4B	Dense (Base)	88.3	91.7	22.7	27.2	79.3	68.3	39.3
4B	Dense (SAPO)	86.9	91.2	46.0	37.7	80.7	68.1	43.4
4B	**MoEcraft**	**92.0**	**96.0**	**51.8**	**44.5**	**85.9**	**73.4**	**48.2**
32B	Dense (Base)	89.8	92.1	20.7	25.9	88.9	67.5	23.9
32B	Dense (SAPO)	**93.8**	**95.9**	**43.9**	**45.3**	**91.2**	**80.8**	**57.5**

Table 8: Part 2: Reasoning and summary benchmarks across scales.

Scale	Configuration	GSM8K	Math500	MBPP	CRUX-O	SAMSum	XSum
0.6B	Dense (Base)	46.2	29.2	90.4	25.3	24.7	15.1
0.6B	Dense (SAPO)	54.1	34.6	99.2	25.5	39.5	21.7
0.6B	**MoEcraft**	**57.4**	**41.2**	**99.8**	**30.2**	**39.0**	**22.5**
1.7B	Dense (Base)	52.2	29.0	93.6	25.9	22.5	16.9
1.7B	Dense (SAPO)	75.1	51.4	98.2	26.1	40.5	20.2
1.7B	Harvest-only	76.9	50.6	96.4	26.0	40.3	21.4
1.7B	**MoEcraft**	**82.6**	**57.0**	**99.0**	**31.2**	**39.6**	**19.2**
4B	Dense (Base)	48.8	37.4	95.2	26.8	25.2	18.1
4B	Dense (SAPO)	86.4	61.6	99.6	26.8	41.5	28.5
4B	**MoEcraft**	**91.8**	**66.4**	**99.8**	**31.9**	**41.5**	**28.4**
32B	Dense (Base)	85.0	61.8	96.8	27.8	25.2	19.8
32B	Dense (SAPO)	**88.5**	**64.0**	**99.2**	**27.6**	**38.4**	**29.0**

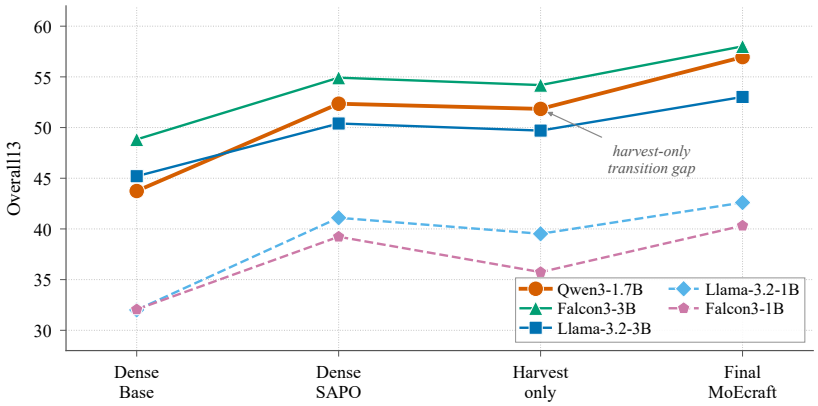

Figure 3: Stage-wise Overall13 (mean of 13 benchmark scores; higher is better) across families. The Harvest-only dip and the Final-stage recovery jointly support the role of sparse-aware joint calibration (L3); the chain Base → SAPO → Harvest-only → Final reproduces across Qwen, Llama, and Falcon.

C Ablation Details

C.1 Initialization and Checkpoint-Selection Controls

These supplementary comparisons examine the choice of expert initialization and checkpoint selection. They complement the trajectory-source analysis in Section 4.4 rather than define a separate contribution.

Identical copying and terminal interpolation. The dense-copy controls duplicate the FFN weights of the terminal dense SAPO checkpoint, C_8, identically across all eight expert slots. Last $\times$ 8 instead uses the LoRA FFN update from the same C_8 with expert-specific merge coefficients from 0.65 to 1.00 in increments of 0.05. It is therefore a single-checkpoint interpolation variant, not eight identical copies of the terminal dense model. Sequential-8 uses FFN adapters from eight distinct trajectory checkpoints, $C_1, \ldots, C_8$. Best-8 also draws eight distinct checkpoints from that trajectory, but the criterion by which they were selected is not recorded in the released configuration, so we do not state one. For an FFN projection, let $\mathbf{W}_0$ denote the starting backbone weight and $\Delta\mathbf{W}_8$ its effective terminal LoRA update, including the LoRA scaling. Then

$$\mathbf{W}_8 = \mathbf{W}_0 + \Delta\mathbf{W}_8, \qquad \mathbf{W}_e^{\text{copy}} = \mathbf{W}_8, \qquad \mathbf{W}_e^{\text{interp}} = \mathbf{W}_0 + \beta_e \Delta\mathbf{W}_8,$$

where $\{\beta_e\}_{e=1}^{8} = \{0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 1.00\}$. The merge coefficient β_e is separate from the LoRA hyperparameter α and scales the FFN update, not the entire model. The shared attention body averages the effective LoRA updates of the contributing checkpoints, as in Section 3.3; for Last $\times$ 8 that set contains only C_8, whose attention adapter is merged at weight 1.00, so the β_e interpolation applies to the FFN experts alone. Post-assembly training is a separate part of the run specification.

Observed comparisons. All three remain well below the trajectory-harvested final: in the settings tested, the sparse-conversion plus joint-alignment pipeline was not sufficient on its own. The three controls differ in the Stage-4 alignment that was run and are reported under their recorded run names: dense-copy only, dense-copy with SFT alignment and dense-copy with SAPO alignment reach 50.97, 48.54 and 50.94. Last $\times$ 8 terminal interpolation reaches 55.92, while Sequential-8 reaches 56.95; Best-8 and the four-unique-checkpoint variant reach 56.69 and 55.98. All original result rows and their category breakdowns are retained in Tables 9 and 10. Sequential-8 exceeds Last $\times$ 8 by 1.03 points and identical copying with SAPO alignment by 6.01 points. These are differences between the documented configurations. Last $\times$ 8 already introduces expert-weight differences through interpolation, so its contrast with Sequential-8 is not diversity versus no diversity. The results support the chosen default among the tested configurations, but do not establish that eight distinct checkpoints are necessary, isolate checkpoint count alone, or constitute a multi-seed estimate of the selection effect.

Table 9: L2 evidence (Qwen3-1.7B): checkpoint-selection and expert-construction variants, all assembling N=8 experts from the same trajectory. Sequential-8 and Best-8 use eight distinct checkpoints and 4 unique $\times$ 2 uses four. Last $\times$ 8 (terminal-adapter interpolation) uses only the terminal checkpoint C_8, but applies expert-specific FFN-update coefficients from 0.65 to 1.00, so its eight experts are not identical copies.

Strategy	Overall13	General7	Math2	Code2	Summary2
Sequential-8 (default)	56.95	58.83	69.80	65.09	29.41
Best-8	56.69	58.12	69.18	65.12	30.74
4 unique $\times$ 2	55.98	56.97	68.89	64.30	31.25
Last $\times$ 8	55.92	57.24	69.00	64.15	29.96

This appendix collects the per-aggregate breakdowns for the ablations whose Overall13 summaries appear in the main body (Sections 4.4 and 4.5), plus the top-k inference ablation.

C.2 Dense-Copy Controls (L2)

Table 10: Dense-copy control (Qwen3-1.7B): per-aggregate breakdown of the three controls versus the trajectory-harvested final. Each control duplicates the FFN weights of the terminal dense SAPO checkpoint C_8 across all eight experts; the rows differ in the Stage-4 alignment that was run. Last $\times$ 8 in Table 9 instead interpolates the FFN update from that same C_8 with expert-specific coefficients, so the two are different constructions.

Setting	Overall13	General7	Math2	Code2	Summary2
Dense-copy only	50.97	50.24	63.02	60.36	32.09
Dense-copy + SFT align	48.54	49.55	60.81	61.05	20.22
Dense-copy + SAPO align	50.94	50.22	62.76	60.52	32.04
Trajectory-harvested Final	56.95	58.83	69.80	65.09	29.41

Figure 4 plots the Overall13 column of Table 10.

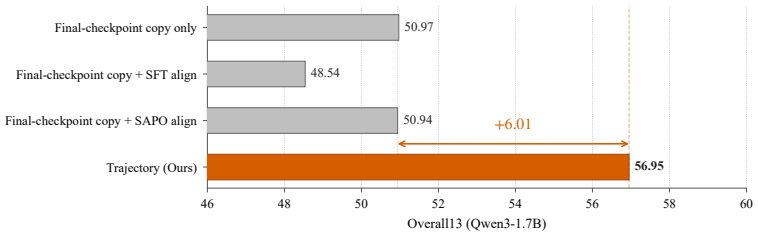

Figure 4: Dense-copy controls vs. trajectory-harvested final on Qwen3-1.7B (Overall13). The sparse architecture plus Stage-4 alignment alone does not reach the trajectory-harvested final, supporting L2.

C.3 Reward Design and RL Algorithm (L2)

Table 11 gives the full stage chain under both reward designs and Table 12 the per-aggregate difference between them.

Table 11: Negative-reward vs. positive-only full chain (Qwen3-1.7B).

Setting	Overall13	General7	Math2	Code2	Summary2
Positive Dense SAPO	52.35	52.70	63.26	62.16	30.37
Positive Harvest-only	51.84	51.76	63.74	61.20	30.88
Positive Final	56.95	58.83	69.80	65.09	29.41
Negative Dense SAPO	52.46	52.91	63.43	61.36	31.03
Negative Harvest-only	51.44	51.28	63.03	60.62	31.20
Negative Final	55.57	56.96	66.23	64.42	31.16

Table 12: Per-aggregate delta of negative-augmented minus positive-only rewards (Qwen3-1.7B). Negatives slightly help the dense SAPO stage on Overall13 but reduce the final aligned sparse model, especially on Math2 (-3.57).

Stage	Δ Overall13	Δ General7	Δ Math2	Δ Code2	Δ Summary2
Dense SAPO	$+0.11$	$+0.21$	$+0.17$	-0.80	$+0.66$
Harvest-only	-0.40	-0.48	-0.71	-0.58	$+0.32$
Final	-1.38	-1.87	-3.57	-0.67	$+1.75$

Table 13: Tested RL configurations on Qwen3-1.7B and the separately tuned SFT outcome baseline (final stage only). SAPO and GRPO both reproduce the dense–harvest–final pattern and their final scores exceed this SFT baseline in the reported setting. This comparison does not establish a general ranking of RL and SFT.

Paradigm	Dense RL	Harvest-only	Final	General7	Math2
SAPO	52.35	51.84	56.95	58.83	69.80
GRPO	52.05	51.31	56.66	58.87	68.19
SFT (best baseline)	–	–	51.40	–	–

C.4 Alignment-Size Sensitivity (L3)

Table 14 breaks the alignment-size sweep down by aggregate.

Table 14: Alignment-size sensitivity (Qwen3-1.7B): per-aggregate breakdown of the diminishing returns beyond N_a=512.

$\lvert\mathcal{D}_{\text{align}}\rvert$	Overall13	General7	Math2	Code2	Summary2
256	55.65	57.14	67.05	63.83	30.88
512	56.95	58.83	69.80	65.09	29.41
1024	57.31	59.18	69.78	65.12	30.46

C.5 Top-k at Inference

Table 15: Top-k at inference (Qwen3-1.7B). Gains are small relative to the active-parameter cost; k=2 is our default.

Config	Total	Active	Overall13	Δ
N=8, k=2 (default)	9.10B	2.76B	57.0	–
N=8, k=3	9.10B	3.82B	57.8	+0.8
N=8, k=4	9.10B	4.88B	58.1	+1.1

D Supplementary Analyses

D.1 Expert-Router Synergy (SAPO/SAPO vs. SFT/SFT)

Table 16 shows the full 2×2 table for expert source and router-alignment choice on the 0.6B scale, with the expert construction held fixed across all four cells. The SAPO/SAPO pipeline is strongest, but the remaining three cells are not ordered monotonically: switching the router-alignment objective to SAPO helps when the experts also come from the RL trajectory ($62.2 \rightarrow 70.0$) and hurts when they come from SFT ($63.0 \rightarrow 56.3$), with one run per cell. The better alignment objective therefore depends on the expert source in this comparison. This table predates the aggregate definitions of Section 4: its Math/Code/General columns are the single-benchmark GSM8K/MBPP/MMLU scores and Overall is their mean, so the values are not comparable to Math2/Code2/General7/Overall13 elsewhere.

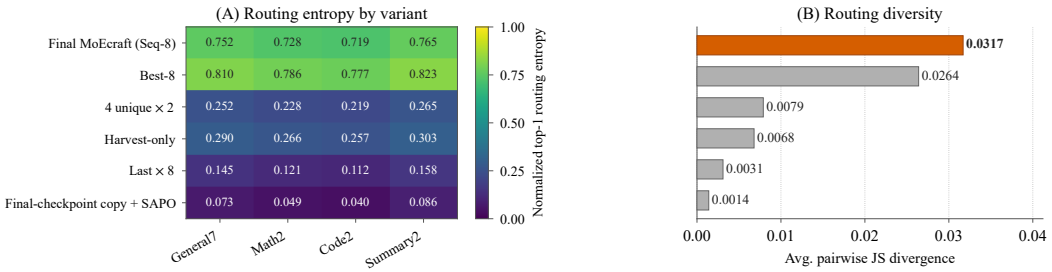

Figure 5: Measured routing on Qwen3-1.7B across six upcycling variants (256 evaluation samples per task group); both panels report the same variants in the same order. **(A)** Normalized top-1 routing entropy per task group (Table 17): trajectory harvests stay non-collapsed (0.72–0.82) while the duplicated, terminal-interpolated, and terminal-copy controls approach single-expert collapse (0.04–0.30). **(B)** Average pairwise JS divergence (Table 18): Sequential-8 (Final MoEcraft) most differentiated, final-checkpoint copy + SAPO align near zero. The two panels rank differently — Best-8 is more balanced but less differentiated — so entropy alone does not identify the effective configuration; the JS ranking is what tracks Overall13 (Table 9).

Table 16: Expert source × router alignment on the 0.6B scale. All four cells use the same expert construction, so the table varies only the two stated axes. Per-benchmark scores are in the supplementary material.

Expert Source	Router Align	Math	Code	General	Overall
SFT (Independent)	SFT	44.2	95.0	49.9	63.0
SFT (Independent)	SAPO	35.6	89.0	44.3	56.3
SAPO (Trajectory)	SFT	38.1	98.6	49.9	62.2
SAPO (Trajectory)	SAPO (ours)	57.4	99.8	52.8	70.0

D.2 Routing Diversity Detail (E2/E3)

Measured at 256 evaluation samples per task group on Qwen3-1.7B. Table 17 shows that Final MoEcraft and Best-8 maintain non-collapsed routing across all four task groups, while reduced-diversity variants and the final-checkpoint copy + SAPO align control are nearly collapsed. Table 18 reports the full pairwise JS divergence across all $\binom{8}{2} = 28$ expert pairs under both top-1 and top-k scopes; the ranking matches the harvest-strategy performance ranking in Table 9. Table 17 supplies the values plotted in Figure 5A, and Table 18 those in Figure 5B, so both panels are reproducible from the tables alone. Entropy is normalized by $\log N$ with $N = 8$. These statistics admit two aggregation orders, and the two differ systematically because entropy is concave while dominant share and JS are convex. Table 22 pools the routed-token counts within each task group and computes the statistic once; the released per-expert usage shares reproduce those values exactly. Table 17 and Table 18 instead average the finer-grained values recorded during evaluation, and are reported as recorded because those inputs are not part of the release. On the canonical run the same measurement gives 0.752 for General7 under the averaging order and 0.857 under the pooling order, so we never compare a value taken under one order against a value taken under the other. Task groups combine into the overall figures quoted below with each of the thirteen benchmarks weighted equally. The released routing files record one value per (variant, task group, scope); the layer scope of the underlying measurement is not recorded, so these statistics are model-level summaries and we do not read them as per-layer quantities.

Table 17: Top-1 routing entropy by task group (Qwen3-1.7B variants); higher = more dispersed routing. Dense-copy + SAPO align is near-zero (single-expert collapse) on every task group.

Variant	General7	Math2	Code2	Summary2
Final MoEcraft (Sequential-8)	0.752	0.728	0.719	0.765
Best-8	0.810	0.786	0.777	0.823
4 unique $\times$ 2	0.252	0.228	0.219	0.265
Harvest-only	0.290	0.266	0.257	0.303
Last $\times$ 8	0.145	0.121	0.112	0.158
Dense-copy + SAPO align	0.073	0.049	0.040	0.086

Table 18: Average pairwise JS divergence between expert usage distributions (Qwen3-1.7B variants); $\binom{8}{2}$=28 pairs per model. Higher = more differentiated experts.

Variant	Top-1 (mean $\pm$ std)	Top-k (mean $\pm$ std)
Final MoEcraft (Sequential-8)	0.0317 ± 0.0142	0.0148 ± 0.0061
Best-8	0.0264 ± 0.0113	0.0129 ± 0.0054
4 unique $\times$ 2	0.0079 ± 0.0037	0.0048 ± 0.0022
Harvest-only	0.0068 ± 0.0030	0.0042 ± 0.0020
Last $\times$ 8	0.0031 ± 0.0016	0.0018 ± 0.0009
Dense-copy + SAPO align	0.0014 ± 0.0008	0.0009 ± 0.0005

D.3 Throughput Measurement Detail

Table 19 reports the throughput-only measurement referenced in Section 4.7. All three rows were measured on the same single NVIDIA RTX PRO 6000 Blackwell Max-Q (96GB VRAM) workstation GPU under one fixed setup so the numbers are hardware-matched and comparable as a per-token compute proxy. The setting metadata of this early harness (batch size, inference backend, KV-cache configuration) was not logged at measurement time, so we re-measured all three models under one fully specified setup; Section D.4 reports that measurement together with resident GPU footprint, and the numbers below are retained only as the earlier per-token compute proxy. Training-time peak GPU memory *was* logged, separately and per stage, in Section D.7. The numbers below should not be interpreted as a complete deployment latency or memory model; they only support the L2 routing-level efficiency claim.

Table 19: Throughput-only measurement on Qwen3-1.7B and Qwen3-4B variants. Earlier harness with unlogged batch size and backend, retained as a per-token compute proxy; the controlled replacement measurement with resident GPU footprint is Table 20.

Model	Total (B)	Active (B)	tok/s
Qwen3-1.7B dense	1.70	1.70	1046.95
Qwen3-4B dense	4.00	4.00	513.26
Qwen3-1.7B MoEcraft	9.10	2.76	706.09

D.4 Controlled Inference Measurement

Table 20 replaces the unlogged harness of Section D.3 with a fully specified serving measurement. All three models were served on the same single NVIDIA RTX PRO 6000 Blackwell Max-Q (96GB VRAM) workstation GPU under one configuration: vLLM 0.19.1, BF16, 512-token prompts, 128-token greedy generations, 128 requests, a fixed 8 GiB KV cache, and batch sizes 1 and 4. These are the single fully logged run under that protocol; no repeated-run or best-of-run selection was performed. Resident footprint is the device memory held after model load with the KV cache allocated; the absolute device peak exceeded it by at most 2 MiB, so the two definitions agree to within 0.01% and we report the resident value. Tail latency preserves the same ordering: at batch 4 the p95 TTFT of MoEcraft is 68.3 ms against 88.2 ms for the dense 4B baseline.

Two readings follow, and they point in opposite directions. Against the dense 4B baseline, MoEcraft is faster at both batch sizes (1.54× and 1.22× output throughput) and has lower time-to-first-token, consistent with its smaller active parameter count. Against the dense 1.7B model it is both slower and larger. Because all 9.10B parameters remain resident, the footprint is 30.0 GiB versus 20.0 GiB for dense 4B, so this measurement supports a performance and active-parameter trade-off and does *not* support a memory-efficiency claim.

Table 20: Controlled inference measurement (vLLM 0.19.1, BF16, 512-token prompt, 128-token greedy generation, 128 requests, fixed 8 GiB KV cache; single 96GB workstation GPU). Resident footprint is batch-independent at these sizes because the KV cache is fixed.

Model	Total (B)	Active (B)	Output tok/s B=1 / B=4	Mean TTFT (ms) B=1 / B=4	p95 TTFT (ms) B=1 / B=4	Resident (GiB)
Qwen3-1.7B dense	1.70	1.70	299.7 / 926.5	10.9 / 28.3	11.4 / 35.1	15.0
Qwen3-1.7B MoEcraft	9.10	2.76	197.4 / 611.9	22.5 / 55.2	23.2 / 68.3	30.0
Qwen3-4B dense	4.00	4.00	128.0 / 501.2	24.4 / 69.7	24.8 / 88.2	20.0

D.5 Stage-4 Seed Robustness

We repeat *only* Stage 4 with three seeds while reusing the same dense SAPO trajectory and the same harvested expert set, isolating Stage-4 randomness from full-pipeline variance. Table 21 reports per-seed Overall13 and the mean/std for the canonical Qwen3-1.7B Final MoEcraft and the matched final-checkpoint copy + SAPO align control. Both settings are stable across seeds and the gap of ≈ 6 points persists, so the final-checkpoint-copy gap reported in Section 4.4 is not a Stage-4 seed artifact. Full-pipeline multi-seed—re-running Stage 2 and Stage 3 under independent seeds—is reported separately in Section D.6: across the canonical run and two re-seeded SAPO trajectories, Final Overall13 is 56.14 ± 0.72 with a range of 1.37, an order of magnitude more variable than Stage-4 reseeding alone. The larger full-pipeline spread is consistent with additional variability across trajectories and their resulting expert sets; we do not estimate separate variance components. These experiments do not further separate trajectory-generation variability from the resulting expert-set variability, and we report 56.14 ± 0.72 as the observed spread over three runs, not as a distributional or worst-case bound.

Table 21: Stage-4 seed robustness on Qwen3-1.7B (Overall13). Same dense SAPO trajectory and harvested experts reused; only Stage 4 alignment is re-seeded.

Setting	Seed 0	Seed 1	Seed 2	Mean	Std
Dense-copy + SAPO align	50.94	51.05	50.90	50.96	0.08
Final MoEcraft	56.95	56.94	57.02	56.97	0.04

D.6 SFT-Trajectory Harvesting Control (L2)

The primary trajectory-source analysis and stage-wise scores are reported in Section 4.4 and Table 3. This appendix retains the corresponding detailed measurements and routing summaries. The source comparison holds the sparse-construction and calibration protocol fixed but does not match the learned dense endpoints or trajectory-generation cost. It is not a reproduction of UpIT or of any other published pipeline: it varies the trajectory source inside our own chain and leaves every other stage fixed. Routing summaries are descriptive diagnostics under the aggregation conventions stated in Section D.2.

Across the same three trajectories the two pre-calibration stages are tightly reproduced: Overall13 is 52.54 ± 0.17 at the dense endpoint and 51.87 ± 0.10 after harvesting, with ranges of 0.33 and 0.19. The trajectories therefore reach similar aggregate performance before calibration and differ mainly in how much Stage 4 then adds (+5.11, +4.12, +3.60). Every trajectory also ends above its own dense endpoint. This is the pattern L2 predicts: the Stage-4 gain tracks the trajectory that supplied the experts, not the calibration stage on its own.

Table 22 reports routing for the same four L3 outputs; these measurements provide complementary evidence about expert utilization and show the same split. The SFT final concentrates 79.4–87.8% of top-1 tokens on one expert in every task group, at normalized entropy 0.274–0.410, whereas all three RL finals stay at 21.2–35.9% dominant share and 0.833–0.914 entropy, so the same sparse architecture and the same Stage 4 do not produce the same routing behavior from the two trajectory sources.

Table 22: Top-1 routing of the L3 outputs in Table 3 (Qwen3-1.7B, 256 evaluation samples per task group). Normalized entropy uses $\log N$ with $N{=}8$; dominant share is the fraction of top-1 tokens going to the most-used expert. Pool-first within each task group (Section D.2); these values are recomputable from the released per-expert usage shares.

		General7	Math2	Code2	Summary2
SFT 2025	Norm. entropy	0.396	0.357	0.274	0.410
	Dominant share	80.4%	82.4%	87.8%	79.4%
SAPO 2025 (canonical)	Norm. entropy	0.857	0.841	0.833	0.859
	Dominant share	34.7%	34.4%	35.4%	35.9%
SAPO 2026	Norm. entropy	0.914	0.907	0.903	0.914
	Dominant share	21.2%	24.4%	22.4%	21.8%
SAPO 2027	Norm. entropy	0.881	0.861	0.867	0.876
	Dominant share	24.5%	25.6%	25.6%	26.4%

D.7 Per-Stage Cost and Training-Time Peak Memory

Table 23 reports wall-clock cost and peak GPU memory per pipeline stage, instrumented for the three re-run trajectories of Section D.6. Because every stage occupies the single GPU exclusively, GPU-hours equal wall-clock hours, so we report one column rather than presenting the same measurement twice. The canonical Qwen3-1.7B run predates this instrumentation and its per-stage cost was never logged; we mark it `not logged` rather than estimating it. Peak memory is the absolute device peak during the stage, of which at most 15 MiB was resident before the stage began, so the incremental and absolute definitions agree to within 0.02%.

Three observations follow. First, the most expensive of the three instrumented Qwen3-1.7B pipelines costs 8.33 GPU-hours, which quantifies the "within a day" claim for this model: the figure is wall-clock measured, not extrapolated. The other (family, scale) pairs were run but not instrumented at this granularity. For reference, Branch-Train-MiX reports 926 GPU-days of total training compute from its seed model, i.e. 22,224 GPU-hours, of which 23% is the MoE-training stage. Second, Stage 4 is the memory-critical stage at 92.9 GiB on a 96 GB device, so the single-workstation constraint is binding rather than comfortable, and the restricted adapter subspace contributes to keeping Stage 4 within that single-device memory budget. Third, the two trajectory sources cost less than a factor of three apart: the matched SFT trajectory takes 2.67 GPU-hours against 4.75–6.73 for SAPO, since SAPO additionally rolls out sampled completions for each of the same N_h prompts. The trajectory-source comparison in Section D.6 is therefore not confounded by an order-of-magnitude budget difference: with a comparable budget, the Stage-4 response is negative on the SFT trajectory (-0.64) and positive on all three SAPO trajectories ($+3.60$ to $+5.11$), measured with every downstream stage held fixed.

Table 23: Per-stage wall-clock cost (GPU-hours; single GPU, so GPU-hours = wall-clock hours) and peak GPU memory for the three re-run trajectories. Successful stages only (146 of 151 ledger rows; the 5 failed retries total 0.04 GPU-hours and are excluded). Peak memory is the maximum process footprint observed across the three runs for that stage, except for Stage 2 where the three sources differ and all three are given. Core conversion covers Stages 2–4; the total additionally includes the three stage evaluations and the routing measurement. Both are computed from unrounded per-stage times, so they can differ from the sum of the rounded entries above by up to 0.01. Both figures exclude the separate one-time L1 attachment (46.7 s, no gradient step; Table 24).

Stage	SFT 2025	SAPO 2026	SAPO 2027	Peak mem.
Stage 2 trajectory	2.67	4.75	6.73	24.7/83.3/80.2 GiB
Stage 3 harvest / merge	0.02	0.03	0.02	34.5 GiB
Stage 4 joint alignment	0.73	0.69	0.76	**92.9 GiB**
Three 13-benchmark stage evals	0.71	0.67	0.79	92.1 GiB
Routing measurement	0.02	0.02	0.02	91.8 GiB
Core conversion (Stages 2–4)	3.41	5.46	7.52	—
Total (Stage 2 to analysis)	**4.15**	**6.16**	**8.33**	—
Canonical run (2025)		`not logged`		

D.8 Per-Stage Complexity and Deployment Guidance

Table 24 reports the measured cost of each pipeline stage. Two stages carry no gradient step and are therefore bounded by allocation and copying rather than by compute; Stage 2 is the largest single entry for every trajectory, which is why trajectory collection dominates the total of Table 23.

Table 24: Per-stage measured cost on Qwen3-1.7B. Rows are the successful ledger entries underlying Table 23.

Stage	Measured
L1 attach subspace	46.7 s, once for this model
Stage 2 trajectory	2.67 h (SFT) / 4.75–6.73 h (SAPO)
Stage 3 harvest + merge	12–35 s per operation
Stage 4 joint alignment	0.69–0.76 h
Inference	+1.06B active per routed expert

Which variant to deploy. The measurements of Section D.4 and Table 15 make this a two-axis decision rather than a recommendation, because all 9.10B parameters stay resident and MoEcraft therefore replaces neither dense model on footprint. Dense Qwen3-1.7B is the choice when resident memory, latency or serving cost is the binding constraint: it wins on all three (15.0 GiB, 10.9 ms TTFT, 299.7 tok/s at batch 1) and loses on accuracy. MoEcraft at N=8/k=2 is the choice when a high-memory GPU is available and the gain over the dense 4B baseline is worth 30.0 GiB resident: higher Overall13 at 2.76B active parameters, 1.54×/1.22× the output throughput at batch 1/4, and lower TTFT at both batch sizes. Dense Qwen3-4B is the choice when a smaller footprint (20.0 GiB) and a plain dense-serving stack matter more than the measured throughput difference. Raising k to 3 or 4 buys 0.8 and 1.1 Overall13 points for 38% and 77% more active parameters (Table 15), so k=2 is our default.

Conversion and serving have different memory requirements. The two should not be conflated. Conversion is bounded by Stage 4 at 92.9 GiB and therefore needs a 96 GB-class device (Table 23); the measured serving footprint of the converted 1.7B-derived model is 30.0 GiB under the controlled setup of Section D.4. A practitioner without the former can still deploy a model produced elsewhere, and our low-resource claim is relative to cluster-scale CPT rather than to modest hardware.