LEARN FROM YOUR BETTER SELF: ON-POLICY RECOVERY FOR SPARSE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse attention reduces inference cost but can induce *sparse reasoning drift*, where early token deviations compound through autoregressive feedback, degrading reasoning and prolonging generation. We introduce sparse on-policy distillation (**SparseOPD**), which uses the original checkpoint's dense execution as a privileged teacher to supervise next-token distributions on trajectories generated by the evolving sparse student. Across Qwen3-4B, Qwen3-14B, and Ministral-3-14B, SparseOPD improves mathematical reasoning and code generation over cross-entropy recovery and fixed-trajectory distillation. On AIME 2026, it nearly restores dense accuracy on Qwen3-4B and exceeds it on Qwen3-14B. It also curbs excessive generation and retains a $1.73\times$ request-level speedup over dense inference on Qwen3-4B HMMT. Our analyses localize on-policy gains to later sparse states and reveal that recovery signals concentrate at low-probability token positions. This motivates **LowP-40**, which supervises only the lowest-probability 40% of tokens. On Qwen3-4B, LowP-40 maintains recovery quality comparable to full-token supervision while accelerating end-to-end training by $1.19\times$.

1 INTRODUCTION

Sparse attention is increasingly used in foundation models to reduce long-context training and inference costs (Liu et al., 2025; Qiu et al., 2026b). Native sparse pretraining has preserved model quality at scale, but each candidate sparse design must be trained and validated at pretraining scale, making alternatives expensive to explore. Dense-to-sparse conversion offers a more flexible route: it reuses a pretrained dense checkpoint and introduces sparse attention afterward. Existing recovery uses continued language modeling with cross-entropy (CE), alignment of attention selection or intermediate representations, or both (Lu et al., 2026; Gao et al., 2024; Xiao et al., 2025; Shen et al., 2025). Across these objectives, recovery operates on fixed training sequences. We therefore ask: *what should recovery supervise, and on which states?*

Sparse execution changes the states visited during autoregressive generation: once sparse decode changes a token, that token enters the prefix and conditions every subsequent prediction. Our phase-controlled analysis (Section 2.2) shows that sparse prefill primarily affects long-context retrieval, whereas sparse decode drives reasoning degradation and failed termination. Among failed sparse generations, disagreement with the dense model grows after divergence even as entropy falls. Figure 1 illustrates this compounding drift and the resulting generation-length tail, which offsets part of the per-token acceleration from sparse attention.

These observations identify two requirements for effective sparse recovery. First, recovery should preserve the dense next-token distribution. CE exposes one token sampled from a dense trajectory and discards the probabilities assigned to alternative continuations (Hinton et al., 2015). Attention or representation alignment transfers internal structure; matching the dense next-token distribution directly constrains predictive behavior. Our gradient and held-out analyses identify sampled-target updates that move the sparse model away from the dense distribution. Second, supervision must cover the states visited by the sparse student. Fixed-sequence supervision does not track the evolving sparse policy after divergence, and cross-state replay shows that the benefit of sparse-state supervision grows along these trajectories, consistent with student-generated distillation (Agarwal et al., 2024).

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.

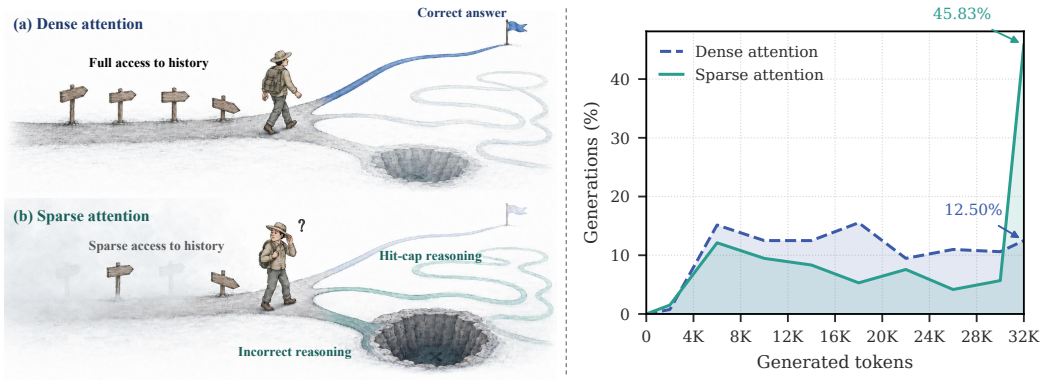


Figure 1: Sparse attention changes both the reasoning trajectory and generation cost. **Left:** dense execution retains access to the full history, whereas sparse execution can omit relevant context and enter a different reasoning branch. **Right:** on HMMT, sparse execution produces a heavier generation-length tail and more outputs at the 32K cap. The additional generated tokens offset part of its per-token speedup.

Fixed-sequence distillation supplies richer targets but preserves this state-coverage gap. *Recovery therefore requires dense distribution supervision on states visited by the sparse student.*

These requirements lead directly to sparse on-policy distillation (**SparseOPD**). Dense-to-sparse conversion provides a naturally aligned teacher–student pair: the original dense checkpoint is frozen as the teacher, and a sparse copy initialized from it becomes the student. They share the tokenizer and initial weights. In each recovery training round, the current sparse student generates trajectories, the dense teacher evaluates its next-token distribution on the resulting sparse prefixes, and the student matches this distribution using forward or reverse KL. The updated student then generates the trajectories for the next round. Dense distribution matching supplies the supervision signal, and trajectory refresh tracks the evolving sparse states. Dense attention serves as privileged information during recovery.

Across Qwen3-4B, Qwen3-14B, and Ministral-3-14B, SparseOPD outperforms both cross-entropy and distillation on fixed dense states in mathematical reasoning and code generation while retaining the measured speedups of sparse execution. It recovers Qwen3-4B AIME Mean@8 from 44.17% to 63.33% and Ministral-3-14B HMMT from 37.12% to 50.38%, bringing both close to their dense-attention performance. To make SparseOPD training more efficient, we introduce LowP-40, which supervises only the lowest-probability 40% of tokens while maintaining comparable performance, accelerating end-to-end recovery by $1.19\times$.

Together, we (i) uncover the mechanism of reasoning drift under sparse decode, tracing how autoregressive state feedback turns early token divergence into growing distribution mismatch, overconfidence, and failed termination; (ii) demonstrate through gradient conflict, held-out loss, component ablations, and cross-state replay that dense-distribution supervision and on-policy sparse-state collection provide complementary recovery gains; and (iii) develop SparseOPD, which combines both signals to improve reasoning recovery while retaining the speedups of sparse execution. More broadly, when an efficiency-oriented model transformation changes autoregressive state visitation, the original model can supervise its transformed counterpart on the states that transformation induces.

2 Preliminary

2.1 Sparse Attention

A decoder-only model receives a prompt $\boldsymbol{x}$ and autoregressively generates $\boldsymbol{y}$. At step t, its state is $s_t = (\boldsymbol{x}, \boldsymbol{y}_{<t})$. Dense causal attention exposes each query to every preceding key and value, whereas sparse attention selects only a subset. For the same weights W, the two executions can therefore

Table 1: Prefill–decode attribution with identical Qwen3-4B weights.

Attention phase		RULER	LongBench	HMMT Feb. 2026		
Prefill	Decode			Mean@4	Pass@4	Hit-cap ↓
Dense	Dense	83.01	53.21	56.94%	72.22%	8.33%
Sparse	Dense	60.26	50.22	51.39%	72.22%	11.11%
Dense	Sparse	72.81	52.47	40.28%	55.56%	51.39%
Sparse	Sparse	46.72	48.19	37.50%	50.00%	55.56%

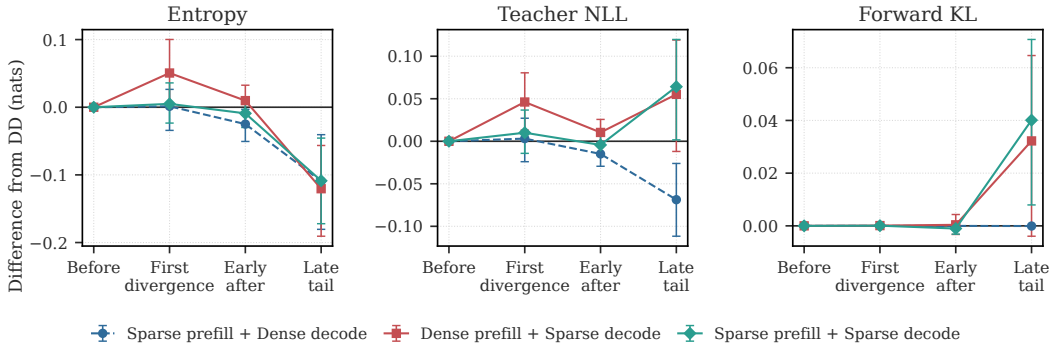


Figure 2: Trajectory changes relative to matched DD across four stages. Near the first divergence, DS entropy rises; later, DS and SS become lower-entropy while their disagreement with the dense teacher increases. Error bars are 95% bootstrap confidence intervals over prompt–seed pairs.

induce different next-token distributions,

$$p^{W}_{\text{dense}}(\cdot \mid s_t), \qquad p^{W}_{\text{sparse}}(\cdot \mid s_t). \tag{1}$$

Query-dependent sparse attention can select tokens, pages, or blocks from the past context. A common distinction lies in how many are retained. Top-k selection keeps the k highest-scoring units (Tang et al., 2024; Lu et al., 2026; Liu et al., 2025), whereas top-p selection keeps the smallest set whose cumulative score mass reaches a target threshold (Lin et al., 2026; Xu et al., 2025).

Top-k selection is the prevailing choice in recent sparse-attention works (Tang et al., 2024; Lu et al., 2026; Yuan et al., 2025; Liu et al., 2025; Qiu et al., 2026b). Accordingly, our main experiments use block-level top-k selection during both prefill and decode. Each query attends to mandatory sink and local blocks together with a fixed number of globally selected blocks. We additionally evaluate top-p selection in complementary experiments.

2.2 Reasoning Drift in Sparse Decode

Sparse Decode Drives Reasoning Degradation. To separate the effects of sparse prefill and sparse decode, we run the same Qwen3-4B checkpoint with every combination of dense and sparse attention in the two phases. DD uses dense attention for both prefill and decode; SD uses sparse prefill and dense decode; DS uses dense prefill and sparse decode; and SS uses sparse attention for both phases. The checkpoint and all other settings remain unchanged. Table 1 reports RULER and LongBench (Hsieh et al., 2024; Bai et al., 2024), together with HMMT February 2026 from MathArena (Dekoninck et al., 2026). For HMMT, Mean@4 is generation accuracy, Pass@4 is problem-level success, and hit-cap is the fraction reaching the output limit. We use fixed diagnostic sets of 768 RULER examples, 240 LongBench examples, and 18 HMMT problems sampled four times.

The sensitive phase depends on the task. RULER and LongBench are hurt more by sparse prefill, showing that prompt-side retrieval and aggregation matter. HMMT shows the complementary reasoning pattern: SD preserves DD Pass@4 and raises hit-cap only from 8.33% to 11.11%, whereas DS lowers Mean@4 from 56.94% to 40.28% and raises hit-cap to 51.39%. Sparse decode is therefore the main source of reasoning degradation and failed termination. Appendix B.2 extends this

comparison across fixed-budget top-k and cumulative-mass top-p selectors and shows that the same phase asymmetry holds across selection rules.

Reasoning Drift Compounds Autoregressively. Trajectory-aligned comparisons reveal how reasoning drift develops. We pair generations by problem and seed, retain cases where DD is correct but the corresponding sparse condition is wrong, and align each pair at its first generated-token divergence. Figure 2 combines three complementary signals. Entropy measures local uncertainty, teacher NLL measures whether the dense teacher supports the generated token, and FKL measures disagreement across the full next-token distribution. Exact definitions are given in Appendix B.1.1.

At the first divergence, DS entropy and teacher NLL rise, indicating an uncertain early deviation. The pattern changes later: DS and SS become lower-entropy while teacher NLL and FKL rise. The sparse model becomes more certain even as its trajectory increasingly disagrees with the dense teacher. Section 5.1 examines how these sparse states become harder to recover and where on-policy collection improves them.

Sparse decode therefore does more than change an isolated token. A changed token is written into the prefix, shifts later predictions, and can produce an incorrect, and hard-to-terminate trajectory. We call prefixes visited along dense and sparse trajectories *dense states* and *sparse states*, respectively. Once the two trajectories diverge, training only on dense states no longer covers the sparse states visited at inference. We next examine whether CE preserves enough information even on the states where supervision is available.

2.3 Two Requirements for Sparse Recovery

CE uses one sampled target at each fixed dense state, whereas FKL directly matches the dense next-token distribution. On our fixed training trajectories, their logit gradients have negative inner products at 20.65% of measured positions. This diagnostic quantifies disagreement between single-token and distribution supervision at individual states (Appendix B.3).

This local disagreement is reflected in training. In the CE recovery training, CE decreases from 0.549 to 0.457, while held-out CE remains essentially unchanged (1.750 to 1.752) and held-out FKL increases from 0.208 to 0.256. CE therefore fits the sampled tokens without reliably preserving the dense next-token distribution.

These results motivate dense distribution matching as the recovery training signal. Its effectiveness also depends on where that signal is applied. As shown in Section 2.2, sparse decode changes subsequent prefixes through autoregressive feedback, producing an evolving state distribution that fixed dense trajectories do not track. These findings identify two requirements for sparse recovery: *dense distribution supervision* and *sparse state coverage*. We therefore introduce sparse on-policy distillation (**SparseOPD**), which matches the dense teacher's next-token distribution on trajectories generated by the current sparse student.

3 Methodology

3.1 Overview

Let T be a frozen dense teacher and S_θ a trainable sparse student. Given a state $s_t = (\boldsymbol{x}, \boldsymbol{y}_{<t})$, the teacher and student define next-token distributions $p_T(\cdot \mid s_t)$ and $p_\theta(\cdot \mid s_t)$ using dense and sparse attention, respectively. Sparse recovery trains p_θ to preserve p_T under sparse execution.

Each round, the sparse student generates trajectories and learns from dense-teacher distributions evaluated on the same prefixes (Figure 3). The updated student then collects the next round's trajectories.

3.2 Distillation Objectives

We use KL divergence to transfer the dense next-token distribution. Forward KL is

$$\mathcal{L}_{\mathrm{FKL}}(s;\theta) = D_{\mathrm{KL}}(p_T(\cdot \mid s) \,\|\, p_\theta(\cdot \mid s)) \,. \tag{2}$$

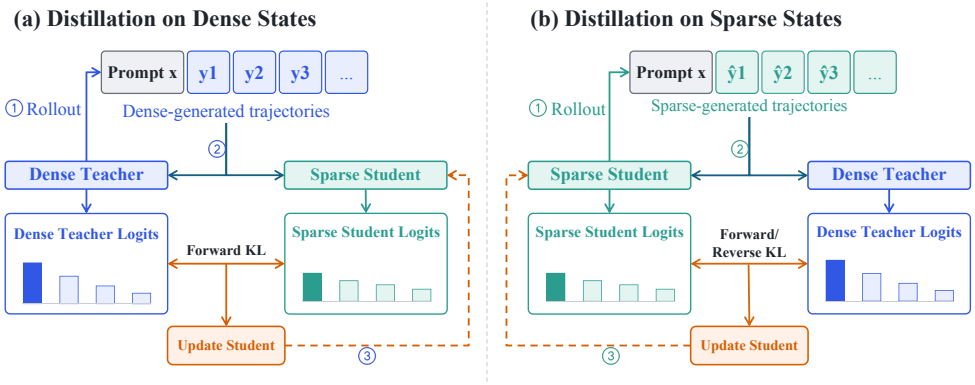


Figure 3: Dense-state distillation and SparseOPD. (a) standard distillation matches the dense teacher on states from fixed dense trajectories. (b) SparseOPD generates states with the current sparse student, applies dense-teacher supervision on the same prefixes, updates the student, and regenerates trajectories in the next round.

FKL weights discrepancies by teacher probability, preserving alternatives supported by the teacher but not selected in a sampled dense trajectory.

We also study reverse KL,

$$\mathcal{L}_{\mathrm{RKL}}(s;\theta) = D_{\mathrm{KL}}(p_\theta(\cdot \mid s) \,\|\, p_T(\cdot \mid s)) \,. \tag{3}$$

RKL weights discrepancies by student probability, focusing the update on student-preferred continuations that the dense teacher assigns low probability. For both directions, we compute KL on the union of the teacher and student top-64 tokens and aggregate the remaining vocabulary into one tail event. This common support preserves probability mass while keeping distribution matching tractable.

CE provides the hard-target baseline: it replaces the dense distribution with one target token. This comparison isolates the value of distribution supervision from the source of training states.

3.3 SparseOPD: On-Policy Distillation

Let d_{dense} denote the state distribution of fixed dense trajectories. Standard FKL minimizes

$$\mathcal{L}_{\text{dense-FKL}}(\theta) = \mathbb{E}_{s\sim d_{\mathrm{dense}}}\left[\mathcal{L}_{\mathrm{FKL}}(s;\theta)\right] . \tag{4}$$

This objective preserves the dense distribution on dense states, but sparse decode can visit states outside that fixed distribution.

At round r, the sparse student S_{θ_r} generates fresh trajectories, defining the on-policy state distribution $d^{\mathrm{sparse}}_{\theta_r}$. For $X \in \{\mathrm{FKL}, \mathrm{RKL}\}$, the round objective is

$$\mathcal{L}_{\text{OPD-}X}(\theta;\theta_r) = \mathbb{E}_{s\sim d^{\mathrm{sparse}}_{\theta_r}}\left[\mathcal{L}_X(s;\theta)\right] . \tag{5}$$

For every collected state, the teacher and student receive the same sparse prefix. The teacher evaluates the dense next-token distribution, and the student is updated under sparse execution with the chosen KL direction. The next round recollects trajectories from the updated student, so the training distribution follows the sparse policy throughout recovery. Trajectory generation determines which states are trained, while optimization within a round treats the collected prefixes as fixed inputs.

This construction yields OPD-FKL and OPD-RKL. We also use OPD-CE as a hard-target control, replacing the KL target at each sparse state with the dense teacher's most likely token. Together, CE/FKL and OPD-CE/OPD-FKL separate the effect of distribution supervision from the effect of training on sparse states.

Table 2: Reasoning performance across model and recovery training methods. AIME and HMMT report eight samples per problem. LiveCodeBench v6 reports single-sample Pass@1. Lower hit-cap is better; all other metrics are higher-is-better. Best and second-best trained methods are marked in **bold** and underlined, respectively.

Method	AIME 2026			HMMT Feb. 2026			LiveCodeBench v6	
	Mean@8	Pass@8	Hit-cap ↓	Mean@8	Pass@8	Hit-cap ↓	Pass@1	Hit-cap ↓
Qwen3-4B								
Dense Ref.	63.75	76.67	10.42	42.05	60.61	12.50	46.86	5.14
Sparse Ref.	44.17	70.00	29.17	27.27	30.30	45.83	33.14	17.14
CE	55.42	76.67	21.67	30.68	42.42	38.26	36.00	40.00
FKL	60.42	80.00	15.83	35.23	48.48	29.92	41.14	16.00
OPD-FKL	62.08	83.33	12.92	35.98	**54.55**	25.38	40.57	**5.71**
OPD-RKL	**63.33**	**86.67**	**10.42**	**36.74**	**54.55**	**22.35**	**43.43**	9.14
Qwen3-14B								
Dense Ref.	71.67	86.67	7.92	47.73	69.70	0.00	57.71	0.57
Sparse Ref.	63.75	90.00	16.67	36.74	63.64	32.20	40.00	41.71
CE	54.17	76.67	38.75	34.47	51.52	41.67	34.86	55.43
FKL	70.42	86.67	11.67	40.91	57.58	24.62	44.00	18.29
OPD-FKL	72.08	**93.33**	11.25	43.56	**66.67**	28.79	46.86	17.14
OPD-RKL	**73.75**	86.67	**8.75**	**46.59**	**66.67**	**12.12**	**49.14**	**14.29**
Ministral-3-14B								
Dense Ref.	72.08	90.00	5.42	53.03	81.82	20.83	52.57	6.29
Sparse Ref.	51.25	86.67	17.08	37.12	72.73	32.20	52.00	18.86
CE	42.50	**90.00**	12.92	33.33	66.67	33.33	42.86	27.43
FKL	57.50	86.67	10.00	40.15	66.67	33.71	50.86	15.43
OPD-FKL	62.08	**90.00**	**9.17**	**50.38**	66.67	**22.35**	54.29	**13.71**
OPD-RKL	**65.00**	**90.00**	**9.17**	45.08	**72.73**	27.65	**57.14**	**13.71**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and recovery methods. We evaluate Qwen3-4B, Qwen3-14B (Yang et al., 2025), and Ministral-3-14B-Reasoning-2512 (Ministral-3-14B) (Liu et al., 2026). Dense Ref. and Sparse Ref. share weights and use dense causal and top-k sparse attention, respectively (Section 2.1). All recovery methods start from Sparse Ref. with the same sparse configuration. We compare CE, FKL, OPD-FKL, and OPD-RKL, with OPD-CE included in the Qwen3-4B ablation.

Training. All methods share a problem pool and a token mixture of 40% reasoning, 25% long-context replay, and 35% code, and train for 200 optimizer steps. CE and FKL use fixed dense-teacher trajectories; SparseOPD refreshes sparse-student trajectories every 20 steps. Training time is measured on one node with eight NVIDIA RTX 6000D GPUs. Data sources, trajectory selection, and optimization settings appear in Appendix C.

Evaluation. We evaluate AIME 2026 and HMMT February 2026 problems from MathArena (Dekoninck et al., 2026), reporting Mean@8 and Pass@8 from eight generations per problem. Code evaluation uses Pass@1 on LiveCodeBench v6 problems (Jain et al., 2025). We also report the fraction of generations reaching the output limit. Full evaluation protocols appear in Appendix C.

4.2 MAIN RESULTS

Sparse attention creates a clear recovery target across all three backbones. Sparse Ref. trails Dense Ref. by 7.92–20.83 points on AIME and 10.99–15.91 points on HMMT. Both Qwen models also lose 13.72–17.71 points on LiveCodeBench, while Ministral-3-14B reaches the output limit more often.

Dense distribution matching is consistently stronger than hard targets. FKL improves CE across all three benchmarks and backbones, including gains of 5.00, 4.55, and 5.14 points on Qwen3-4B AIME, HMMT, and LiveCodeBench, and AIME gains of 16.25 and 15.00 points on the two 14B models.

On-policy sparse states improve recovery further. OPD-FKL raises AIME and HMMT over FKL on every backbone, improves LiveCodeBench on both 14B models, and reduces the Qwen3-4B LiveCodeBench hit-cap rate from 16.00% to 5.71%. Both KL directions improve fixed-state distillation by applying dense supervision to states visited by the sparse student.

4.3 Dense Distributions and Sparse States Both Matter

Table 2 establishes the benefit of matching dense next-token distributions on fixed dense states. To isolate the contribution of state coverage, we retain CE supervision and collect trajectories from the current sparse student. On Qwen3-4B, OPD-CE reaches 61.67 on AIME, 33.71 on HMMT, and 37.71 on LiveCodeBench, improving over CE by 6.25, 3.03, and 1.71 points, respectively. Sparse-state collection therefore contributes independently of distribution matching.

The two components are complementary: OPD-FKL improves over OPD-CE on all three benchmarks, while OPD-RKL achieves the strongest overall results. SparseOPD combines dense distribution supervision with states induced by the current sparse model. Section 5.1 localizes these gains to sparse states that fixed dense trajectories do not cover. Appendix B.4 reports the complete component ablation and compares two additional Qwen3-4B on-policy objectives: GRPO learns from verifier rewards on sparse-policy trajectories, while OPRD aligns their hidden representations with a frozen dense teacher. Across these objectives, SparseOPD achieves the strongest results on AIME, HMMT, and LiveCodeBench, showing that dense output distributions provide the most consistent recovery signal.

4.4 SparseOPD Retains Recovery with 40% Token Supervision

OPD-FKL improves recovery by refreshing sparse-student trajectories throughout training. This policy-dependent collection couples rollout with optimization and adds a run-specific cost relative to fixed-state FKL, whose dense-teacher trajectories can be generated once and reused across training seeds. Appendix C.9 reports the total wall-clock costs and their amortization across repeated runs.

We reduce this per-run cost by concentrating distribution supervision on the most informative token positions. At each on-policy round, LowP-40 ranks the eligible positions within every trajectory by the sparse student's sampled-token log-probability and retains the lowest 40%. Every prompt and trajectory remains in the training set, and the teacher and student project vocabulary logits at the selected positions. LowP-40 shares the complete OPD-FKL training contract; Appendix C.4 gives the selection and implementation details.

LowP-40 maintains comparable performance to full OPD-FKL across AIME, HMMT, and LiveCodeBench. It reduces total recovery time from 22.70 to 19.04 hours, corresponding to a $1.19\times$ end-to-end speedup.

4.5 SparseOPD Improves the Quality–Speed Tradeoff

Figure 4 separates per-token efficiency from the cost of complete generations. The measured per-token gain grows with context length and batch size, reaching $3.47\times$ at 32K with batch size 8. Sparse execution therefore provides substantial throughput gains in the regime where long-context attention is most expensive.

The generation quality–speed tradeoff determines how much of this gain survives at the request level. Sparse Ref. is faster per token but produces fewer correct answers and reaches the output limit more often. CE and FKL progressively improve the quality–speed tradeoff. OPD achieves the strongest recovered point, reaching 54.55% HMMT Pass@8 at a measured $1.73\times$ request-level speedup while reducing the hit-cap rate from 45.83% to 25.38%. SparseOPD therefore preserves sparse execution speed while recovering much of the quality lost during dense-to-sparse conversion. Measurement details appear in Appendix C.8.

Table 3: Qwen3-4B recovery with selective supervision. LowP-40 applies OPD-FKL to the lowest-probability 40% of eligible positions within each trajectory. Total wall-clock time is reported in hours. The final row reports quality differences and end-to-end speedup relative to full OPD-FKL.

Method	AIME Mean@8	HMMT Mean@8	LCB Pass@1	Total wall-clock time (h)
OPD-FKL	62.08	35.98	40.57	22.70
OPD-FKL LowP-40%	61.67	36.36	40.00	19.04
LowP-40% vs. OPD-FKL	−0.41	+0.38	−0.57	1.19×

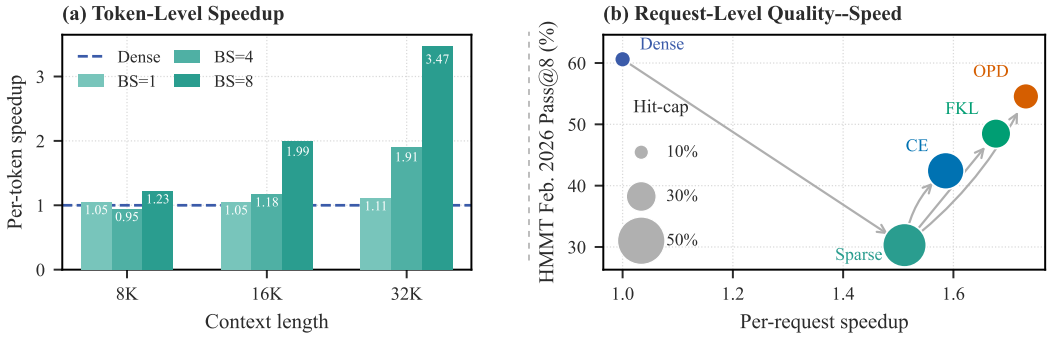


Figure 4: Efficiency under sparse execution and recovery. (a) measured per-token speedup of sparse execution over dense execution. (b) HMMT Pass@8 against measured request-level speedup; circle area denotes the hit-cap rate. OPD denotes OPD-FKL.

5 Analysis

5.1 On-Policy Recovery Targets Sparse States and Redirects Updates

Sparse trajectories exhibit three failure modes: local reasoning errors, continued search to the output limit, and periodic repetition (Appendix B.1.5). Continuing the same sparse-generated prefix under dense and sparse execution shows that dense attention restores natural termination only before drift stabilizes. Once an incorrect decision or periodic tail enters the prefix, it restores neither correctness nor convergence. Recovery must therefore cover sparse states while drift is forming.

Cross-state replay shows that SparseOPD provides this coverage. We evaluate FKL and OPD-FKL on the same 72 held-out HMMT dense and sparse trajectories across four equal stages. The methods remain closely matched on dense states. On sparse states, no gain appears in Q1; relative to FKL on fixed dense states, OPD-FKL lowers dense-teacher FKL by 3.7% in Q2, 9.9% in Q3, and 14.0% in Q4. On-policy gains thus emerge where sparse trajectories leave states covered by fixed dense distillation. Detailed experiments and results appear in Appendix B.1.6 and Appendix B.1.7.

Changing the training states also redirects the parameter update. Compared with fixed-state FKL, OPD-FKL allocates more update energy to attention and shifts FKL's late-layer emphasis toward earlier and middle transformer layers (Figure 5). Appendix B.6 provides the complete parameter-space analysis.

5.2 Recovery Signal Concentrates at Low-Probability, High-Entropy Tokens

OPD-FKL applies distribution supervision across the generated response, but the resulting recovery signal is highly concentrated. We track ten checkpoints from steps 20–200 and rescore fresh sparse-student trajectories with the dense teacher. For each generated position t, we measure the squared norm of the FKL gradient with respect to the student logits, $e_t = \|\nabla_{\mathbf{z}_t} \mathcal{L}_{\mathrm{FKL}}\|_2^2$. The analysis covers 29,607 positions from 312 trajectories; Appendix C.5 gives the complete protocol and the corresponding RKL analysis.

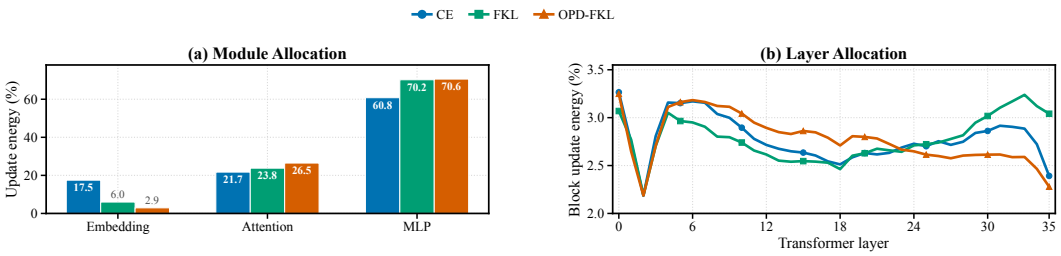


Figure 5: On-policy states redirect parameter updates on Qwen3-4B. (a) Final update-energy allocation across modules. (b) Final allocation across transformer layers. OPD-FKL assigns more update energy to attention and shifts fixed-state FKL's late-layer emphasis toward earlier and middle layers.

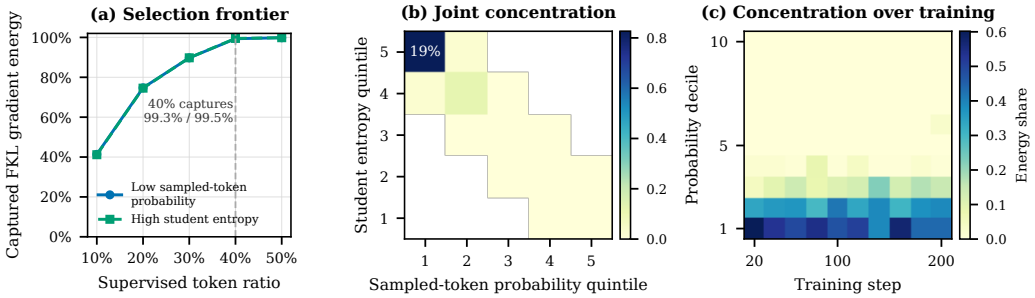


Figure 6: FKL recovery signal concentrates at low-probability, high-entropy token positions. (a) The lowest-probability and highest-entropy 40% of positions capture approximately 99% of FKL logit-gradient energy. (b) Both criteria identify a shared low-probability, high-entropy region; its corner cell contains approximately 19% of positions. (c) Low-probability concentration persists from step 20 to 200. Energy is normalized within each trajectory before aggregation.

The selection frontier rises sharply (Figure 6a). The lowest-probability 20% of positions capture 74.7% of FKL logit-gradient energy, the lowest 30% capture 89.7%, and the lowest 40% capture 99.3%. The 40% selection exceeds 90% at every checkpoint. Ranking by student entropy gives the same frontier: the highest-entropy 40% capture 99.5%. Their joint grid concentrates in the lowest-probability, highest-entropy corner, which contains approximately 19% of positions and carries most of the gradient energy (Figure 6b).

This structure persists throughout training: the lowest probability deciles dominate the energy profile from step 20 to 200 (Figure 6c). Sampled-token log-probability therefore provides a simple selector for the same informative positions and is already recorded during rollout. This concentration motivates LowP-40 in Section 4.4, which preserves recovery using supervision at only 40% of eligible positions.

6 Conclusion

This work frames dense-to-sparse conversion as an on-policy recovery problem. SparseOPD lets a sparse model learn from its dense counterpart on the states induced by sparse execution, while LowP-40 makes recovery more efficient by concentrating supervision on informative token positions. Together, they provide a practical workflow for exploring sparse attention designs from pretrained checkpoints. Beyond sparse attention, SparseOPD suggests a broader conversion-and-recovery workflow: transform a pretrained model for a target execution, collect the states induced by that execution, and let the original model supervise those same states. Future work can test this principle for sliding-window or linear attention, KV-cache eviction, and adaptive depth. This perspective separates the design of an efficient execution mechanism from the recovery procedure used to preserve model behavior, making new deployment mechanisms easier to explore from a shared pretrained checkpoint.

AI USE STATEMENT

In this work, we used generative AI tools to write and debug parts of the experimental code. Additionally, we used these tools for selected aspects of language editing in the manuscript, and the non-text visual elements in the left panel of Figure 1 were AI-generated.

We did not use generative AI tools to generate synthetic datasets, propose or refine hypotheses, design or provide feedback on the research methodology or experiments, clean or reformat datasets, interpret experimental results, or produce any reported measurement. Tasks involving theoretical or conceptual model development, mathematical claims or proofs, translation, and qualitative or thematic data analysis are not applicable to this work.

We reviewed all AI-assisted work: experimental code was inspected and tested, and edited text and visual elements were manually reviewed by the authors. We take responsibility for the final content of this work, including text, claims, code, figures, and other artifacts produced with the aid of generative AI.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Yuchen Cai, Ding Cao, Liang Lin, Chunxi Luo, Xin Xu, Kai Yang, Weijie Liu, Saiyong Yang, Tianxiang Zhao, Guangzhong Sun, et al. Learning to foresee: Unveiling the unlocking efficiency of on-policy distillation. *arXiv preprint arXiv:2605.11739*, 2026.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4599–4610, 2021.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. *arXiv preprint arXiv:2605.00674*, 2026.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 32694–32717, 2024.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1317–1327, 2016.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. *arXiv preprint arXiv:2604.13016*, 2026.

Chaofan Lin, Jiaming Tang, Shuo Yang, Hanshuo Wang, Tian Tang, Boyu Tian, Ion Stoica, Song Han, and Mingyu Gao. Twilight: Adaptive attention sparsity with hierarchical top-p pruning. *Advances in Neural Information Processing Systems*, 38:119406–119432, 2026.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *Advances in Neural Information Processing Systems*, 38:17790–17815, 2026.

Wentao Ni, Kangqi Zhang, Zhongming Yu, Oren Nelson, Mingu Lee, Hong Cai, Fatih Porikli, Jongryool Kim, Zhijian Liu, and Jishen Zhao. Double-p: Hierarchical top-p sparse attention for long-context llms. *arXiv preprint arXiv:2602.05191*, 2026.

Quantong Qiu, Zhiyi Hong, Yi Yang, Haitian Wang, Kebin Liu, Qingqing Dang, Juntao Li, and Min Zhang. Flux attention: Context-aware hybrid attention for efficient llms inference. *arXiv preprint arXiv:2604.07394*, 2026a.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, et al. On the design of qwen3. 8-next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026b.

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

Zhenyi Shen, Junru Lu, Lin Gui, Jiazheng Li, Yulan He, Di Yin, and Xing Sun. Ssa: Sparse sparse attention by aligning full and sparse attention outputs in feature space. *arXiv preprint arXiv:2511.20102*, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Zecheng Tang, Quantong Qiu, Yi Yang, Zhiyi Hong, Haiya Xiang, Kebin Liu, Qingqing Dang, Juntao Li, and Min Zhang. Elastic attention: Test-time adaptive sparsity ratios for efficient transformers. *arXiv preprint arXiv:2601.17367*, 2026.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. In *International Conference on Learning Representations*, volume 2025, pp. 37228–37253, 2025.

Ruyi Xu, Guangxuan Xiao, Haofeng Huang, Junxian Guo, and Song Han. Xattention: Block sparse attention with antidiagonal scoring. *arXiv preprint arXiv:2503.16428*, 2025.

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.

Shenzhi Yang, Guangcheng Zhu, Bowen Song, Haobo Wang, Mingxuan Xia, Xing Zheng, Yingfan Ma, Zhongqi Chen, Weiqiang Wang, Junbo Zhao, et al. Oprd: On-policy representation distillation. *arXiv preprint arXiv:2606.06021*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23078–23097, 2025.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

Hanqing Zhu, Zhenyu Zhang, Hanxian Huang, DiJia Su, Zechun Liu, Jiawei Zhao, Igor Fedorov, Hamed Pirsiavash, Zhizhou Sha, Jinwon Lee, et al. The path not taken: Rlvr provably learns off the principals. *arXiv preprint arXiv:2511.08567*, 2025.

A Related Work

A.1 Sparse Attention

Modern sparse attention selects the KV tokens or blocks most relevant to each query. Top-k methods retain a fixed number of candidates, as in Quest (Tang et al., 2024), MoBA (Lu et al., 2026), SeerAttention (Gao et al., 2024), Native Sparse Attention (Yuan et al., 2025), and DeepSeek Sparse Attention (Liu et al., 2025). Top-p methods allocate the budget from the score distribution itself. Twilight introduces hierarchical top-p pruning (Lin et al., 2026); XAttention combines antidiagonal block scoring with top-p selection (Xu et al., 2025); and Double-P performs coarse-to-fine top-p selection (Ni et al., 2026). A related direction adapts the amount of dense computation to the input. Elastic Attention assigns attention heads to different computation modes (Tang et al., 2026), while Flux Attention makes this decision at the layer level to preserve regular execution and measured speedups (Qiu et al., 2026a). These methods determine which context is retained and where sparse computation is applied. Our work studies how the model should be trained after sparse execution is introduced.

A.2 Recovery Training for Sparse Attention

Sparse models are commonly adapted through continued language modeling. MoBA trains the backbone under sparse attention (Lu et al., 2026), while DeepSeek-V3.2 and Qwen3.8-Next introduce sparse components through staged continued training (Liu et al., 2025; Qiu et al., 2026b). Other methods draw a direct training signal from dense execution. SeerAttention distills dense attention patterns into a sparse gate (Gao et al., 2024), and DuoAttention learns which heads require full context by matching hidden states from dense and compressed execution (Xiao et al., 2025). DeepSeek Sparse Attention first aligns its indexer with dense attention before sparse training (Liu et al., 2025). SSA jointly trains full and sparse attention and aligns their layer outputs in both directions (Shen et al., 2025).

These approaches establish dense execution as an effective source of supervision for sparse adaptation. Their alignment is computed on fixed training sequences and primarily targets attention selection or intermediate representations. SparseOPD matches the dense next-token distribution on trajectories generated by the current sparse model, bringing dense supervision to the states created by sparse execution itself.

A.3 On-Policy Distillation

Knowledge distillation traditionally transfers teacher predictions on a fixed dataset (Hinton et al., 2015), and sequence-level distillation trains on fixed teacher generations (Kim & Rush, 2016). MiniLLM instead optimizes reverse KL on student generations (Gu et al., 2024), while GKD lets the student generate training sequences and then applies teacher feedback on the resulting states (Agarwal et al., 2024). Recent analyses show that effective on-policy distillation depends on compatible teacher–student behavior and alignment at student-visited states (Li et al., 2026). OPSD constructs an on-policy teacher from the same model by conditioning it on privileged reasoning information (Zhao et al., 2026), and OPRD extends on-policy supervision to hidden representations (Yang et al., 2026).

SparseOPD connects on-policy distillation with sparse recovery. Dense and sparse execution begin from the same checkpoint, so they share the tokenizer, parameters, and pretrained capabilities. Dense execution supplies the teacher distribution, and sparse execution supplies the states that require recovery. This pairing turns the original dense model into a naturally aligned teacher for its sparse counterpart.

Table 4: Location of the first sampled divergence on paired HMMT trajectories. High-entropy states are the highest-entropy fifth of states within each trajectory, measured under the dense teacher.

Condition	Pairs	First divergence in high-entropy states (%)
DS	12	100.0
SS	15	93.3

B Additional Analysis

B.1 Sparse Reasoning Drift

B.1.1 Trajectory Comparison

Figure 2 uses paired HMMT trajectories for which DD is correct and the corresponding sparse condition is wrong. We align each pair at its first generated-token divergence. Let s_t^c be the prompt followed by the tokens generated before step t under condition c, and let y_t^c be the next generated token. We evaluate both the distribution under condition c, p_c, and the dense-teacher distribution, p_T, on this same prefix:

$$H_c(s_t^c) = -\sum_v p_c(v \mid s_t^c) \log p_c(v \mid s_t^c), \tag{6}$$

$$\mathrm{NLL}_T(s_t^c, y_t^c) = -\log p_T(y_t^c \mid s_t^c). \tag{7}$$

Entropy measures uncertainty under condition c, whereas teacher NLL measures how strongly the dense teacher supports the generated token on the same prefix.

Forward KL uses the top-64 union-tail approximation from training. Let U_t be the union of the top-64 tokens under p_c and p_T. We aggregate the remaining vocabulary into a single tail event. The per-token metric is

$$\begin{aligned}\widetilde{D}_{\mathrm{FKL}}(s_t^c) = &\sum_{v \in U_t} p_T(v \mid s_t^c) \log \frac{p_T(v \mid s_t^c)}{p_c(v \mid s_t^c)} \\ &+ p_T(\overline{U}_t \mid s_t^c) \log \frac{p_T(\overline{U}_t \mid s_t^c)}{p_c(\overline{U}_t \mid s_t^c)}.\end{aligned} \tag{8}$$

Unlike teacher NLL, which scores only the generated token, FKL also captures differences among alternatives that were not selected.

To produce Figure 2, we divide each trajectory into four windows: all tokens before divergence, tokens 32 before through 128 after divergence, the first 2,048 post-divergence tokens, and the final 2,048 tokens. We average each metric within a trajectory and window, subtract the value from the DD trajectory with the same prompt and seed, and then average over pairs. Error bars are 95% paired-bootstrap confidence intervals.

B.1.2 High-Entropy Tokens Mark the First Divergence

For each sparse trajectory, we evaluate the entropy of the dense next-token distribution at every generated state and define the highest fifth as high-entropy states. Dense and sparse generations use the same sampling seed and share an identical prefix before their first divergence. A different sample therefore reflects a change in the next-token distribution, while high entropy determines how readily that change alters the sampled continuation.

Table 4 shows that the first divergence is strongly concentrated in high-entropy states, especially when sparse attention is used during decode. FKL nevertheless remains small at most of these divergences. The two measurements capture different stages: entropy identifies states where a small perturbation can change the sampled branch, whereas FKL captures the distributional drift that develops after the resulting token is written into the prefix.

Table 5: Finite correction on HMMT reasoning trajectories whose first divergence occurs at a high-entropy token. The table includes six DS and eight SS trajectory pairs. Each entry is the change in correct answers after following dense rather than sparse tokens; positive values favor the dense correction.

Correction length	DS	SS
1	-1	0
4	$+1$	-2
16	$+1$	-1
64	-1	0

B.1.3 Finite Dense Corrections Do Not Restore Sparse Decode

We test whether correcting the trajectory at its first divergence is enough to restore sparse decode. For each DS and SS HMMT reasoning trajectory, we create two versions. One follows the dense run for 1, 4, 16, or 64 tokens after the first divergence; the other follows the original sparse run for the same number of tokens. Both then resume sparse decode with the same random seed. All forced spans end before the answer and EOS.

Table 5 reports trajectories whose first divergence occurs at a high-entropy token. A positive value means that following the dense run produces more correct answers than following the original sparse run; a negative value means fewer.

No correction length improves correctness consistently across DS and SS. Once sparse decode resumes, the two versions usually diverge again at the next token, even after a 64-token correction. A finite dense correction therefore does not keep the trajectories aligned.

B.1.4 Sparse Trajectory Outcomes

We inspect paired Dense Ref. and Sparse Ref. trajectories from AIME, HMMT, and LiveCodeBench. Sparse failures follow three patterns. A local decision can change while the remainder of the reasoning stays coherent and terminates with a wrong answer. Other trajectories continue exploring new derivations until the output limit, even after producing a plausible answer. A smaller group enters periodic repetition.

B.1.5 Case Study

Figures 7–9 present three examples from each failure pattern. Each example shows the question and the reasoning around the decisive difference. Dense Ref. and Sparse Ref. share the same weights and differ only in attention execution.

Case 1: Counting Error (AIME 2026, Problem 4)

Question. How many integers at most 100 can be written as $a + b + ab$ for distinct positive integers a and b?

Dense Ref. (Correct). "$n + 1 = xy$. There are 73 composite numbers in $[6, 101]$; excluding the prime squares $9, 25, 49$ gives $73 - 3 = \boxed{70}$."

Sparse Ref. (Incorrect). The trajectory uses the same factorization but writes: "The total number of primes from 2 to 101 is 25, so there are 75 composite numbers. Excluding $4, 9, 25, 49$ gives $75 - 4 = \boxed{71}$." The single counting error is carried through an otherwise coherent solution.

Case 2: Wrong Geometric Branch (HMMT Feb. 2026, Problem 22)

Question. A convex pentagon $HORSE$ satisfies $\angle EHO = \angle ORS = \angle SEH = 90^\circ$, $\angle HOR = \angle RSE = 135^\circ$, $HO = 20$, $SE = 26$, and $OS = 10$. Find its area.

Dense Ref. (Correct). With $H = (0, 0)$, $O = (20, 0)$, $S = (26, 8)$, and $E = (0, 8)$, the convex branch gives $R = (27, 7)$. The shoelace formula then gives area $\boxed{191}$.

Sparse Ref. (Incorrect). From $(x - 20)^2 = y^2$, the trajectory chooses the opposite branch and sets $R = (19, 1)$. It consistently applies the shoelace formula to these coordinates and returns $\boxed{177}$.

Case 3: Missing Circular Boundary (LiveCodeBench 3750)

Question. For each queried position in a circular array, return the minimum distance to another position containing the same value, or -1 if no such position exists.

Dense Ref. (Correct). The solution stores sorted positions for each value and checks the previous and next positions together with the first and last positions. It evaluates each candidate using $\min(|i-j|, n-|i-j|)$.

Sparse Ref. (Incorrect). When the query is the first occurrence, the candidate set includes the query itself and the next occurrence but can omit the last occurrence. The code therefore misses the nearest neighbor across the circular boundary, despite using the correct binary-search structure.

Figure 7: Local decision errors. Sparse execution preserves the high-level solution but changes one count, geometric branch, or boundary condition; the remaining trajectory stays coherent and terminates normally.

Case 4: Unresolved Enumeration (AIME 2026, Problem 7)

Question. How many permutations π of $\{1, 2, 3, 4, 5, 6\}$ satisfy $\pi^6(a) = a$ for every element a?

Dense Ref. (Correct). The valid cycle lengths are $1, 2, 3$, and 6. Enumerating the eight valid cycle types gives counts $120, 40, 120, 40, 15, 45, 15, 1$, whose sum is $\boxed{396}$.

Sparse Ref. (Incorrect). The trajectory switches to an exponential generating function, obtains $491/720$, notices that it conflicts with its manual count, and repeatedly restarts the coefficient calculation: "Clearly, there is a problem. Where is the mistake? ... perhaps the generating function is ..." It continues debugging until the output limit.

Case 5: Correct Candidate without Termination (HMMT Feb. 2026, Problem 32)

Question. Complex numbers α and β satisfy $|\alpha| = |\beta| = M$ and $\alpha\beta + \alpha + \beta + 100 = 0$. Find all possible values of M.

Dense Ref. (Correct). "$(\alpha + 1)(\beta + 1) = -99$. The equal-modulus constraint gives $M^2 = 100$, hence the only possible value is $\boxed{10}$."

Sparse Ref. (Incorrect). The trajectory also reaches $M = 10$, but does not commit: "Earlier analysis suggested an interval. Let me re-express the original equation. ... Let me try $M = 9.9$. ... Let me try $M = 9.99$. ..." It continues considering new parameterizations until the output limit.

Case 6: Recurrence Debugging without Convergence (LiveCodeBench ABC399 F)

Question. Given an array A and $K \leq 10$, compute modulo 998244353 the sum of the K-th powers of all subarray sums.

Dense Ref. (Correct). Let S_r be the prefix sum. Expanding $(S_r - S_l)^K$ and maintaining prefix sums of S_l^i yields an $O(NK)$ algorithm, which the trajectory verifies on a small example before emitting complete code.

Sparse Ref. (Incorrect). A candidate recurrence gives 41 where the example requires 49. The trajectory identifies one missing multiplicity, revises the formula, then returns to the same discrepancy: "Thus, there is a mistake in the code's logic. But why? ... The correct recurrence is ..." The analysis keeps changing but never reaches executable code before the output limit.

Figure 8: Non-terminating search. Sparse trajectories continue producing new calculations and revisions, sometimes after reaching the correct candidate, but fail to commit to a final answer or program.

Case 7: Proposition-Level Loop (AIME 2026, Problem 7)

Question. How many permutations π of $\{1, 2, 3, 4, 5, 6\}$ satisfy $\pi^6(a) = a$ for every element a?

Dense Ref. (Correct). The trajectory enumerates every valid cycle type, sums their counts, and returns $\boxed{396}$.

Sparse Ref. (Incorrect). After a long valid discussion of cycle decompositions, the continuation enters the loop: "a 4-cycle $(1\ 2\ 3\ 4)$ and a transposition $(4\ 5)$? No, it's a 4-cycle ..." The same proposition repeats until the output limit.

Case 8: Numeric Absorbing State (HMMT Feb. 2026, Problem 3)

Question. Find the sum of all positive integers n that have at least six positive divisors and whose sixth-largest divisor is 6.

Dense Ref. (Correct). The divisors of 48 in descending order are $48, 24, 16, 12, 8, 6, 4, 3, 2, 1$, so the sixth-largest is 6. The trajectory excludes the remaining candidates and returns $\boxed{48}$.

Sparse Ref. (Incorrect). After an extended divisor analysis, the continuation collapses to "$480, 480, 480, 480, \ldots$" and emits the same value until the output limit.

Case 9: Repeated Code Commentary (LiveCodeBench 3805)

Question. Given a binary string, perform at most one trade between surrounded runs of ones and zeros to maximize the number of active positions.

Dense Ref. (Correct). The solution run-length encodes the string. For each interior one-run, it adds the lengths of the adjacent zero-runs and returns the original number of ones plus the largest gain.

Sparse Ref. (Incorrect). The generated program enters a comment loop: "iterate over all runs ... but this is not correct ... for the purpose of this example, we can assume that there is one run ..." The same block is copied repeatedly until the output limit.

Figure 9: Periodic repetition. After a substantial prefix, the continuation enters a stable loop over a proposition, number, or code-comment block and repeats it until the output limit. Ellipses omit repeated text.

Table 6: Continuing the same sparse-generated prefix under dense or sparse execution. Each category contains three prefixes and two new seeds per prefix. Correct and hit-cap report percentages over the six matched continuations in each row.

		Correct (%)		Hit-cap (%)	
Prefix category	Pairs	Dense	Sparse	Dense	Sparse
Both-correct control	6	100.0	100.0	0.0	0.0
Wrong decision	6	33.3	33.3	0.0	0.0
Non-periodic search	6	50.0	66.7	16.7	50.0
Periodic repetition	6	0.0	16.7	100.0	100.0

Table 7: Dense-teacher FKL on fixed HMMT trajectories. DD and SS identify the source of each fixed prefix. Positive relative reductions indicate lower FKL under OPD-FKL. The interaction subtracts the OPD-FKL gain on DD from its gain on SS; intervals are paired-bootstrap 95% confidence intervals.

State source	Checkpoint	Q1	Q2	Q3	Q4
DD	FKL	0.00828	0.01526	0.02623	0.03390
DD	OPD-FKL	0.00894	0.01584	0.02697	0.03396
SS	FKL	0.00888	0.02033	0.03425	0.04520
SS	OPD-FKL	0.00927	0.01957	0.03086	0.03886
SS	Relative reduction (%)	−4.4	3.7	9.9	14.0

Paired sparse-state interaction

	Q1	Q2	Q3	Q4
Estimate	0.00028	0.00134	0.00413	0.00640
95% CI lower	0.00008	0.00092	0.00325	0.00501
95% CI upper	0.00049	0.00177	0.00504	0.00776

B.1.6 Sparse-State Recoverability

We test recoverability using 12 sparse-generated prefixes: three from each failure pattern and three both-correct controls. Each prefix is continued with two new seeds under dense and sparse execution, while keeping the prompt, prefix, sampling parameters, and remaining output budget fixed. Table 6 reports the resulting 24 matched continuation pairs.

Both executions preserve correctness and termination on every control pair. For non-periodic continuations, dense execution reaches the output limit once, compared with three times under sparse execution. Once a wrong decision has entered the prefix, dense execution does not improve correctness. At periodic-loop entry, every continuation reaches the output limit under both modes. Dense execution can restore termination before drift stabilizes, while later prefixes carry the failure forward independently of the current attention mode.

B.1.7 Cross-State Recovery

We isolate the effect of state coverage using the 72 paired DD and SS HMMT trajectories from the phase analysis. The generated tokens remain fixed. We evaluate the FKL and OPD-FKL checkpoints under sparse execution on every prefix, using the frozen dense model as teacher, and divide each trajectory into four equal stages. Each stage is averaged within a trajectory before aggregation across the 72 pairs.

FKL and OPD-FKL remain closely matched on dense states. Their behavior separates as sparse trajectories progress: OPD-FKL reduces dense-teacher FKL by 3.7%, 9.9%, and 14.0% over Q2–Q4. The paired state-source interaction grows over the same stages, localizing the benefit of on-policy collection to states reached along sparse trajectories.

B.2 Sparse Degradation Across Selection Rules

We repeat the prefill–decode attribution experiment with fixed-budget Block-Top-k and cumulative-mass Block-Top-p attention. The Block-Top-k grid retains 2K, 4K, or 8K KV tokens. The Block-Top-p grid uses $p \in \{0.7, 0.8, 0.9\}$ with router temperature 0.1. Every condition uses the same Qwen3-4B checkpoint, prompts, sample identifiers, generation seeds, and scorers as the Preliminary experiment. Tables 8 and 9 report the complete results.

Table 8: Block-selector and phase attribution on the frozen RULER and LongBench diagnostic sets. The shared DD scores are 83.01 on RULER and 53.21 on LongBench. Higher is better.

	RULER			LongBench		
Selector	SD	DS	SS	SD	DS	SS
Top-k 2K	49.39	62.61	33.98	43.55	51.76	38.75
Top-k 4K	60.26	72.81	46.72	50.22	52.47	48.19
Top-k 8K	70.79	79.45	63.79	53.31	52.57	52.84
Top-p 0.7	46.69	71.00	42.21	21.61	49.54	18.86
Top-p 0.8	63.13	79.79	60.89	32.28	52.29	30.79
Top-p 0.9	75.01	85.48	75.26	43.14	52.71	42.91

Table 9: Block-selector and phase attribution on the frozen HMMT February 2026 diagnostic set (18×4). The shared DD baseline is 56.94 Mean@4, 72.22 Pass@4, and 8.33 hit-cap. All values are percentages.

	Sparse prefill, dense decode (SD)			Dense prefill, sparse decode (DS)			Sparse prefill, sparse decode (SS)		
Selector	Mean@4	Pass@4	Hit-cap	Mean@4	Pass@4	Hit-cap	Mean@4	Pass@4	Hit-cap
Top-k 2K	51.39	72.22	11.11	31.94	33.33	61.11	30.56	38.89	65.28
Top-k 4K	51.39	72.22	11.11	40.28	55.56	51.39	37.50	50.00	55.56
Top-k 8K	51.39	72.22	11.11	56.94	72.22	22.22	50.00	66.67	20.83
Top-p 0.7	51.39	72.22	11.11	23.61	38.89	59.72	23.61	38.89	48.61
Top-p 0.8	51.39	72.22	11.11	31.94	44.44	30.56	34.72	44.44	36.11
Top-p 0.9	52.78	77.78	11.11	43.06	61.11	27.78	41.67	55.56	29.17

Both selector families reproduce the same phase pattern. Sparse prefill causes the larger loss on RULER and LongBench, while HMMT degradation concentrates in sparse decode. Smaller budgets provide too little context: Top-k 2K and Top-p 0.7 produce the weakest SS results across the three benchmarks. Larger budgets recover quality while retaining many causal blocks; at 32K, Top-k 8K retains 44.04% and Top-p 0.9 retains 67.62%. We therefore select Top-k 4K and Top-p 0.8 as the balanced settings for post-training.

Top-p 0.8 dynamically expands its KV span with context length. It retains an average of 1.21K, 2.22K, 4.32K, and 8.01K KV tokens at 4K, 8K, 16K, and 32K, respectively. Its receptive field consequently exceeds the fixed Top-k 4K budget on longer contexts while preserving stronger sparsity on shorter ones.

B.3 CE and FKL Gradient Alignment

A hard target reduces the dense next-token distribution to one sampled outcome. We compare sample-specific CE gradients with distribution-matching gradients at the same dense state. Let $p_T(\cdot \mid s)$ and $p_\theta(\cdot \mid s)$ denote the dense and sparse next-token distributions, and let y be the token sampled along the dense trajectory. Their gradients with respect to the student logits are

$$\boldsymbol{g}_{\mathrm{CE}} = p_\theta - \boldsymbol{e}_y, \qquad \boldsymbol{g}_{\mathrm{FKL}} = p_\theta - p_T, \tag{9}$$

where $\boldsymbol{e}_y$ is the one-hot distribution for the sampled dense token. Unlike FKL, CE contains no information about the probabilities assigned to alternative continuations.

We measure alignment using the cosine similarity between the two gradients and define a local conflict when

$$\boldsymbol{g}_{\mathrm{CE}}^{\top} \boldsymbol{g}_{\mathrm{FKL}} < 0. \tag{10}$$

Table 10: Ablating training states and objectives on Qwen3-4B. AIME and HMMT report Mean@8; LiveCodeBench v6 reports Pass@1. Best and second-best methods are marked in **bold** and underlined, respectively.

Method	States	Objective	AIME	HMMT	LiveCodeBench
CE	Dense	CE	55.42	30.68	36.00
FKL	Dense	FKL	60.42	35.23	41.14
OPD-CE	Sparse	CE	61.67	33.71	37.71
OPD-FKL	Sparse	FKL	62.08	35.98	40.57
OPD-RKL	Sparse	RKL	**63.33**	**36.74**	**43.43**

A conflict means that the CE update locally increases FKL to the dense model. Across $727{,}857$ supervised tokens, the mean gradient cosine is 0.537. The two updates nevertheless conflict on $150{,}330$ tokens, corresponding to 20.65% of the evaluated positions.

The sampled dense token need not be the teacher's most likely continuation. We measure its relative support using the teacher margin

$$m_T(s, y) = \log p_T(y \mid s) - \max_{v \neq y} \log p_T(v \mid s). \tag{11}$$

When $m_T < -2$, the teacher's preferred alternative is more than seven times as likely as the sampled token. The gradient conflict rate in this region reaches approximately 83%, compared with approximately 9% when $m_T \geq 4$. Local conflict is therefore concentrated where the sampled token represents the dense distribution least faithfully.

The positive average alignment explains why CE can learn useful behavior from dense trajectories. The local conflicts explain why fitting one sampled token does not reliably preserve the dense next-token distribution. Even when $p_\theta = p_T$, FKL has reached its optimum while the single-sample CE gradient continues to sharpen the sampled token. The endpoint comparisons in Section 4 show how this difference affects sparse recovery.

B.4 Ablations of Recovery Supervision and State Coverage

Table 10 separates the contributions of dense distribution supervision and on-policy state coverage on Qwen3-4B.

B.4.1 Alternative On-Policy Objectives

We use two additional Qwen3-4B baselines to separate on-policy state coverage from the source and form of supervision. GRPO learns from verifier rewards on fresh sparse-policy trajectories (Shao et al., 2024). OPRD evaluates the frozen dense teacher on sparse-policy trajectories and aligns their hidden representations (Yang et al., 2026). Both baselines begin from the same Sparse Ref. common-start checkpoint and train for 200 optimizer steps. Table 11 reports the full comparison.

Table 11: Alternative on-policy objectives on Qwen3-4B. AIME and HMMT report eight samples per problem; LiveCodeBench v6 reports greedy Pass@1. Lower hit-cap is better; all other metrics are higher-is-better. Best and second-best trained methods are marked in **bold** and underlined, respectively.

Method	AIME 2026, 30 × 8 (%)			HMMT Feb. 2026, 33 × 8 (%)			LiveCodeBench v6, 175 × 1 (%)	
	Mean@8	Pass@8	Hit-cap ↓	Mean@8	Pass@8	Hit-cap ↓	Pass@1	Hit-cap ↓
Dense Ref.	63.75	76.67	10.42	42.05	60.61	12.50	46.86	5.14
Sparse Ref.	44.17	70.00	29.17	27.27	30.30	45.83	33.14	17.14
GRPO	38.75	63.33	31.25	24.24	45.45	42.42	28.57	60.57
OPRD	62.08	80.00	**10.00**	35.98	45.45	**21.21**	37.71	46.29
OPD-FKL	62.08	83.33	12.92	35.98	**54.55**	25.38	40.57	**5.71**
OPD-RKL	**63.33**	**86.67**	10.42	**36.74**	**54.55**	22.35	**43.43**	9.14

Table 12: Long-context performance on Qwen3-4B. RULER is reported by context length, and LongBench is grouped by task category. Best and second-best trained methods are marked in **bold** and underlined, respectively.

Method	RULER				LongBench						
	8K	16K	32K	Avg.	S-Doc QA	M-Doc QA	Summ.	In-Context	Synthetic	Code	Avg.
Dense Ref.	72.68	77.58	73.34	74.53	47.74	40.42	24.47	62.88	66.13	51.22	47.75
Sparse Ref.	73.31	56.18	40.52	56.67	33.38	31.88	24.67	58.61	9.17	51.29	34.49
CE	75.80	57.83	42.31	58.64	39.30	34.10	15.37	35.83	38.33	8.86	30.05
FKL	76.09	55.64	40.77	57.50	40.80	35.17	24.26	59.59	57.96	**54.52**	43.91
OPD-FKL	**78.94**	**61.08**	**44.43**	**61.48**	**41.98**	**36.66**	24.03	59.40	**59.68**	51.93	**44.34**
OPD-RKL	76.79	58.03	41.89	58.90	41.84	34.54	**24.56**	**60.11**	57.08	52.13	43.79

GRPO falls below Sparse Ref. by 5.42 points on AIME, 3.03 points on HMMT, and 4.57 points on LiveCodeBench. Under the matched 200-step budget, on-policy sparse-state training with verifier rewards does not recover the capability lost under sparse execution. Dense-teacher representation supervision provides a much stronger signal: OPRD exceeds GRPO by 23.33, 11.74, and 9.14 points on the three benchmarks.

OPRD matches OPD-FKL on AIME and HMMT Mean@8, showing that hidden-state alignment provides a strong reasoning signal. Output-distribution supervision is more consistent on code: OPD-FKL raises LiveCodeBench Pass@1 from 37.71% to 40.57% and reduces hit-cap from 46.29% to 5.71%. OPD-RKL achieves the strongest primary score on all three benchmarks. Dense teacher supervision drives recovery, and output distributions provide the most reliable signal across reasoning and code.

B.5 Long-Context Tasks Separate Recovery from Selection

Long-context benchmarks test distinct sparse capabilities. LongBench (Bai et al., 2024) combines long-input understanding and generation, making it sensitive to sparse decode drift. RULER (Hsieh et al., 2024) is a synthetic retrieval benchmark with answer-relevant information in the prompt, making prefill selection central.

Table 12 reflects this distinction. FKL and OPD-FKL recover 9.30 and 9.85 LongBench points over Sparse Ref. Recovery training yields limited improvements on RULER. We measure the fraction of blocks containing answer-relevant information selected by any KV head. Under top-k selection, this selected fraction falls from 87.0% at 8K to 39.7% at 32K; OPD-FKL changes it by only 2.8 points. Sparse prefill restricts how information from unselected blocks is integrated into contextual representations.

Keeping the checkpoint and sparse decode fixed, we replace only prefill selection with XAttention-style threshold selection (Xu et al., 2025). At similar block budgets, the selected fraction rises from 62.6% to 78.9%, and RULER accuracy rises from 44.7% to 55.7% (Figure 10). LongBench shows that recovery extends to long-input generation, while RULER highlights the importance of prefill selection. SparseOPD and improved selection are complementary: the former trains behavior on sparse-generated states, while the latter improves the incorporation of task-relevant prompt information during prefill.

Table 13: Prefill block selection on frozen RULER validation. Relevant blocks contain information needed for the answer and count as selected if any KV head selects them. Both conditions use the same checkpoint and sparse decode, while retaining 32.0 and 33.0 blocks per KV head on average.

	Relevant blocks selected (%)			RULER accuracy (%)		
Slice	Top-k	Threshold	Δ	Top-k	Threshold	Δ
Overall	62.64	78.87	+16.23	44.68	55.72	+11.04
8K	86.83	94.12	+7.29	59.67	73.96	+14.30
16K	60.81	80.11	+19.30	43.28	58.38	+15.10
32K	40.29	62.39	+22.11	31.09	34.80	+3.71

RULER accuracy by task group

Task group	Top-k	Threshold	Δ
Needle retrieval	53.07	67.40	+14.32
Question answering	31.25	45.83	+14.58
Common-word extraction	39.58	33.12	-6.46
Variable tracking	21.25	29.79	+8.54

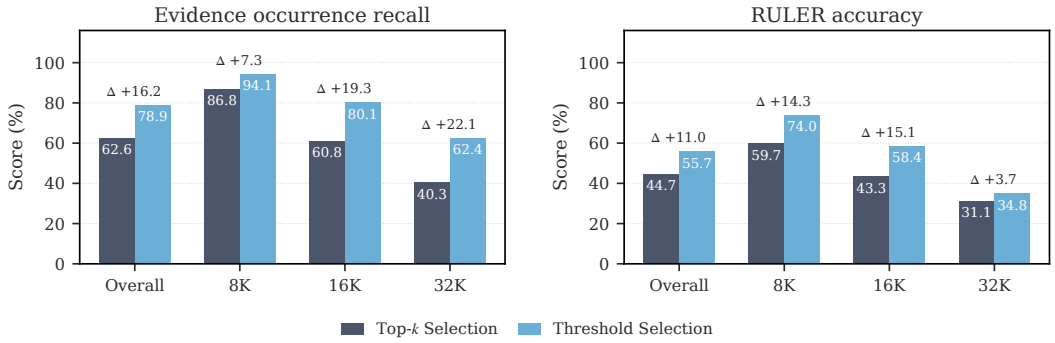


Figure 10: At similar block budgets, changing only prefill block selection includes more answer-relevant blocks and improves RULER accuracy across context lengths.

B.5.1 Block-Selection Details

The controlled comparison uses the same Sparse Ref. checkpoint, RULER validation set, and top-k sparse decode. Thresholds are calibrated by retained block count on one example from each task–length group; the remaining 168 examples are used for evaluation. Top-k and threshold selection retain 32.0 and 33.0 blocks per KV head on average, respectively.

The eight KV heads under top-k selection often choose the same blocks, covering 40.7 distinct global blocks per layer despite 120 available slots. Table 13 reports the complete breakdown by length and task group at the matched block budget.

B.6 Parameter Update Analysis

B.6.1 Measurements

We compare the Qwen3-4B CE, FKL, and OPD-FKL checkpoints at steps $20, 40, \ldots, 200$ from the same Sparse Ref. checkpoint. The tied input and output embedding is counted once. Let $\boldsymbol{\theta}_0$ denote the common start and $\Delta\boldsymbol{\theta}_t = \boldsymbol{\theta}_t - \boldsymbol{\theta}_0$ the cumulative update. The cumulative relative magnitude is $\|\Delta\boldsymbol{\theta}_t\|_F / \|\boldsymbol{\theta}_0\|_F$. We measure convergence to the final update direction using

$$A_t = \cos(\Delta\boldsymbol{\theta}_t, \Delta\boldsymbol{\theta}_{200}). \tag{12}$$

For a parameter group G, its update-energy share is

$$E_G(t) = \frac{\sum_{W \in G} \|\Delta W_t\|_F^2}{\sum_W \|\Delta W_t\|_F^2}. \tag{13}$$

Table 14: Final Qwen3-4B parameter-update statistics. Relative norms are measured against Sparse Ref.; matrix statistics are medians over 252 attention and MLP projection matrices. Early–final cosine compares the step-20 cumulative update with the final update.

Method	Rel. norm (%)	Early–final cosine	Update r_{90}	Outside original rank-64 (%)
CE	0.333	0.640	772	91.62
FKL	0.322	0.695	739	90.67
OPD-FKL	0.305	0.730	704	91.16

The layer analysis renormalizes this quantity over the 36 transformer blocks, excluding the tied embedding and normalization parameters.

All three methods make similarly small complete-model updates, remaining below 0.35% of the initial parameter norm. OPD-FKL has the smallest final update and aligns with its final direction earlier than CE and fixed-state FKL (Figure 11).

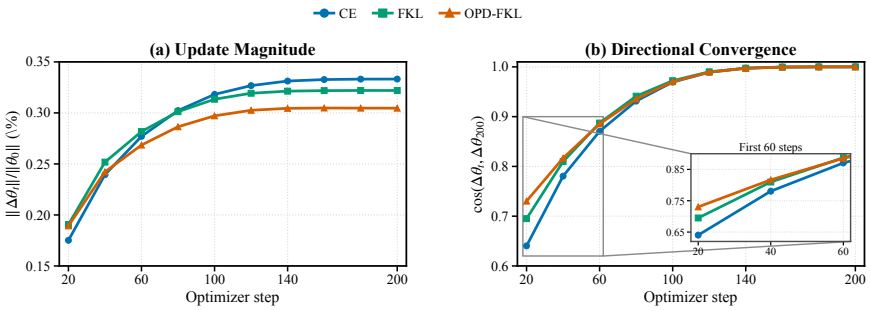


Figure 11: Complete-model update dynamics on Qwen3-4B. (a) Cumulative update magnitude relative to the initial weights. (b) Alignment with each method's final update direction; the inset enlarges the first 60 steps.

B.6.2 Cross-Method Update Direction

We compute the cosine between each pair of complete-model cumulative updates at every shared checkpoint. The low alignment persists across training, showing that changing the supervision or the training states redirects the full-model update rather than merely rescaling it.

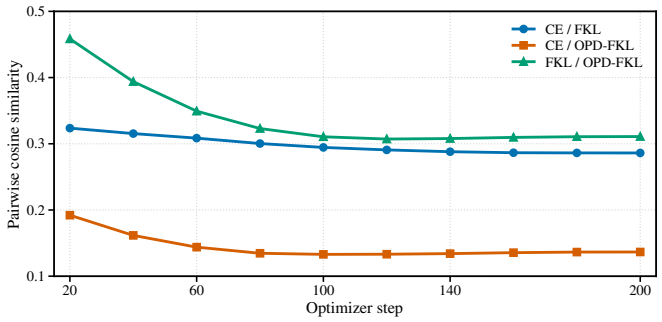


Figure 12: Pairwise cosine similarity between cumulative parameter updates. CE, FKL, and OPD-FKL follow distinct update directions throughout recovery training.

B.6.3 On-Policy States Broadly Redistribute Attention Updates

Sparse-state supervision redistributes attention updates across the full GQA structure. We compare FKL and OPD-FKL after normalizing their complete updates to equal norm. Each GQA group contains the four query and output heads that share one key–value head. Figure 13 shows the squared attention-update difference across the 36 layers and eight GQA groups. Each group carries close to one eighth of the difference, while the first twelve layers carry 41.9% of it. The resulting change is broad across attention groups and strongest in earlier layers.

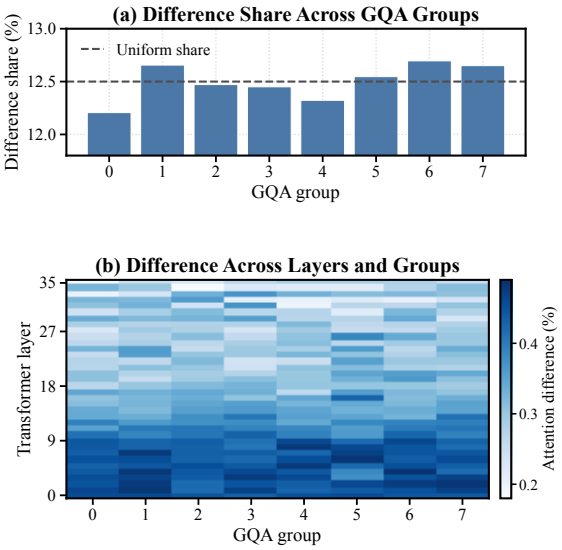


Figure 13: Attention-update redistribution induced by sparse states. We normalize the complete FKL and OPD-FKL updates to equal norm before taking their squared difference. (a) The aggregate difference is distributed across all eight GQA groups. (b) The layer–group heatmap shows that the difference is strongest in earlier layers.

B.6.4 Update Concentration and Original Structure

Following the matrix diagnostics of Zhu et al. (2025) and Cai et al. (2026), we also analyze all 252 attention and MLP projection matrices. For each matrix, r_{90} is the smallest number of singular directions containing 90% of the update energy. To measure change outside the dominant structure of the original weight, let $W_0 = U\Sigma V^\top$ and let U_{64}, V_{64} contain its leading 64 singular vectors. We report

$$E_{\perp 64} = \frac{\|(I - U_{64}U_{64}^\top)\Delta W(I - V_{64}V_{64}^\top)\|_F^2}{\|\Delta W\|_F^2}. \tag{14}$$

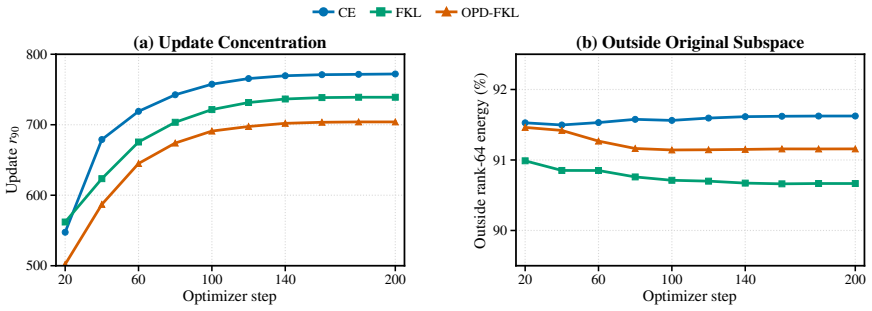


Figure 14: Update concentration and use of the original weight subspaces. (a) The number of singular directions required to explain 90% of update energy. (b) Update energy outside both original rank-64 subspaces. OPD-FKL is slightly more concentrated, while all three methods update primarily outside the original dominant structure.

The three methods produce broad updates while preserving the original dominant structure: median r_{90} remains above 700, more than 90% of update energy lies outside both original rank-64 subspaces, and the original rank-64 subspaces rotate by less than $0.061°$. This is a shared signature of sparse recovery across the three objectives. Within that regime, OPD-FKL has the lowest r_{90} and is therefore slightly more concentrated than CE and FKL.

C Experimental Details

C.1 Models and Sparse Attention

Dense Ref. and Sparse Ref. share identical weights. Dense Ref. uses FlashAttention-2 causal attention; Sparse Ref. uses the block-sparse execution used throughout training and evaluation. The KV cache is partitioned into 128-token blocks. For each query block and KV head, the main Top-k pattern includes one sink block, the 16 most recent blocks, and 15 additional global blocks.

Global blocks use normalized query–key similarity. We average the query heads associated with each KV head, use the first token as the query-block summary, and mean-pool each key block. Their dot product selects the 15 highest-scoring causal blocks. Prefill and decode use the same rule and budget across methods and model scales.

Block-Top-p converts the same normalized query–key scores into a dynamic block budget. For causal-visible blocks $\mathcal{C}$, it computes

$$z_b = \frac{\langle \mathrm{norm}(\bar{q}), \mathrm{norm}(\bar{k}_b) \rangle}{\tau}, \qquad \pi_b = \frac{\exp(z_b)}{\sum_{j \in \mathcal{C}} \exp(z_j)}. \tag{15}$$

The selector sorts blocks by π_b and retains the shortest prefix whose cumulative mass reaches p. Our post-training experiments use $p = 0.8$, $\tau = 0.1$, 128-token blocks, and a shared candidate set containing every causal-visible block. The cumulative selector directly determines the retained set.

During prefill, the query anchor supplies the current block's key summary, maintaining causal routing within the block. Contiguous attention consumes the resulting block mask directly. Paged decode represents each request and KV head with a ragged block list and refreshes the selection at every generated token. The same routing rule is used during training and evaluation.

Top-k and top-p capture two common KV selection principles in contemporary sparse attention: retaining a fixed number of high-scoring candidates or adapting that number to their score mass. Existing systems instantiate these principles with different scorers, granularities, and kernels. We implement both within one block-sparse backend so that the checkpoint, scoring function, mandatory context, and prefill and decode kernels remain controlled across training objectives. This common implementation isolates recovery training from system-specific differences while covering both top-k and top-p sparse attention.

Qwen models use their released tokenizer and chat template. Ministral-3-14B uses its canonical Tekken tokenizer and native chat-completion encoding. All models are trained and evaluated in BF16.

C.2 Training Data

The fixed training set contains 3,200 complete trajectories: 1,404 reasoning trajectories, 420 long-context replay trajectories, and 1,376 code trajectories. Because trajectory lengths differ, batches are scheduled by scored tokens to obtain the 40% reasoning, 25% replay, and 35% code mixture reported in Section 4.

Reasoning trajectories are generated by the corresponding Dense Ref. on OpenMathReasoning, AIME 2024–2025, and HMMT 2024–2025 problems. Long-context replay combines RULER-style synthetic prompts disjoint from evaluation with the training splits of HotpotQA, 2WikiMultiHopQA, and Qasper at sequence lengths from 8K to 40K (Yang et al., 2018; Ho et al., 2020; Dasigi et al., 2021). Code trajectories are complete, correct, naturally terminated Dense Ref. solutions to 800 LiveCodeBench v1–v5 problems. We reserve the latest 80 problems from the v5 increment for development and use the 175-problem v6 increment only for final testing.

Table 15: Default training configuration.

Item	Value
Training steps	200
Global batch size	16 sequences
Sequence lengths	8K, 16K, 32K, 40K
Optimizer	AdamW
Learning rate	1×10^{-5}
Weight decay	0.1
Adam β_1, β_2	0.9, 0.95
Schedule / warmup	Cosine / 0.03
Gradient clipping	1.0
Precision	BF16
KL support / temperature	Top-64 union and tail / 1.0
SparseOPD refresh interval	20 steps
Rollout sampling	$T = 0.6$, top-p $= 0.95$, top-$k = 20$

CE and FKL train on the Dense Ref. trajectories. SparseOPD uses the same prompts and source proportions, but generates new completions with the current sparse student. Each method owns its collected trajectories; responses are not shared across methods.

C.3 Optimization and On-Policy Collection

All methods update every model parameter for 200 optimizer steps. The 3,200 sequence visits form one length-bucketed pass with a global batch size of 16. We use AdamW, BF16 training, and gradient checkpointing. Table 15 lists the shared configuration.

SparseOPD uses ten rounds of 20 optimizer steps. Each round begins with fresh trajectories from the current sparse student on the next 320 prompts. The frozen dense teacher scores every visited prefix, and the student performs 20 updates on those sparse states. All methods share the prompt schedule, token mixture, optimizer budget, and sparse attention configuration.

For FKL and RKL, teacher and student probabilities are evaluated on the union of their top-64 tokens, with all remaining probability aggregated into one tail event. OPD-CE instead uses the dense teacher's most likely token at each sparse state. This isolates state collection from distribution supervision.

C.4 Low-Probability Selective Supervision

LowP-40 follows the complete OPD-FKL training contract and changes the supervised-position fraction from 100% to 40%. For a trajectory with eligible generated actions $y_1, \ldots, y_n$, let

$$\ell_t = \log p_{\theta_r}(y_t \mid s_t) \tag{16}$$

be the unwarped sampled-token log-probability recorded by the round-r sparse student during rollout. The selector retains

$$k = \max(1, \lceil 0.4n \rceil) \tag{17}$$

positions with the smallest ℓ_t within each trajectory. Action index resolves ties, and the retained positions return to sequence order before training. The eligible set incorporates the existing periodic-repetition mask.

The dense teacher and sparse student process each complete trajectory prefix. Their selected hidden states are gathered before the vocabulary projections, and the Top-64 union-tail FKL is evaluated on these compact tensors. Each round manifest records the eligible and retained token counts, selection rule, log-probability source, and realized supervision ratio.

C.5 Token-Level Concentration of Distribution Supervision

C.5.1 Measurement Protocol

We track all ten Qwen3-4B OPD-FKL checkpoints at steps $20, 40, \ldots, 200$. Each checkpoint generates fresh sparse trajectories from a fixed panel of 24 prompts, evenly divided among reasoning, code, and long-context tasks. Each trajectory contributes at most 96 evenly spaced generated positions. Steps 20, 100, and 200 include a second rollout seed. The resulting analysis contains 29,607 positions from 312 trajectories.

At each position, we compute full-vocabulary and Top-64 union-tail FKL and RKL. For objective D, its logit gradient and gradient energy are

$$\mathbf{g}_t^D = \nabla_{\mathbf{z}_t} \mathcal{L}_D, \qquad e_t^D = \|\mathbf{g}_t^D\|_2^2, \tag{18}$$

where $\mathbf{z}_t$ denotes the sparse student's logits. Distribution signal is the share of KL carried by a group of positions; gradient signal is its share of e_t^D. Sampled-token increase records whether the gradient step raises the logit of the generated token and measures direction independently of magnitude. All gradient measurements in this section are therefore in logit space. We first aggregate within each trajectory and then average across trajectories, giving every trajectory equal weight.

C.5.2 Marginal Signal Concentration

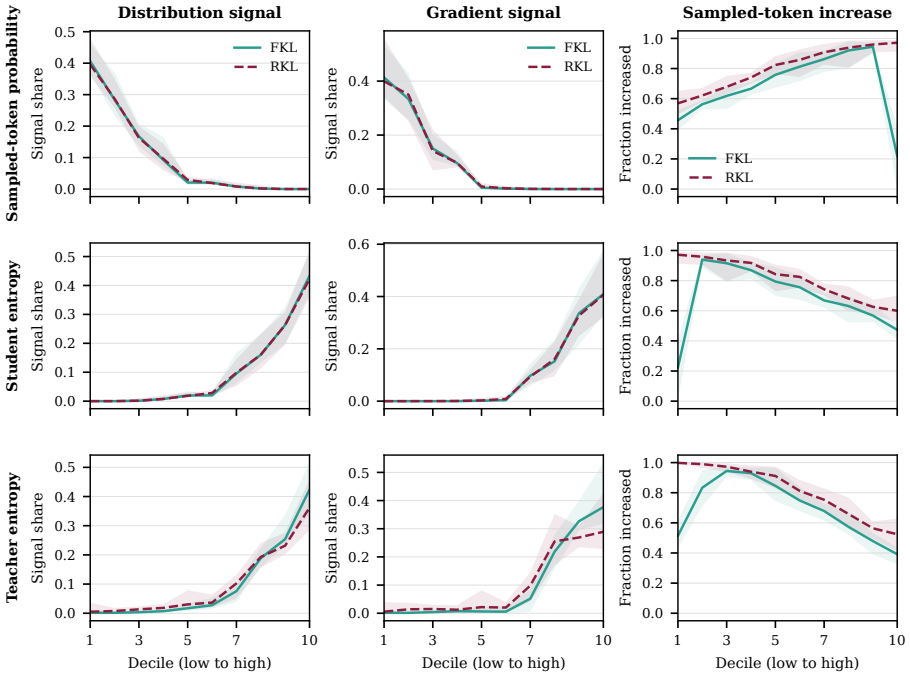


Figure 15: Marginal distribution and gradient signals. Positions are divided into deciles by sampled-token probability, student entropy, or teacher entropy. Solid lines show FKL and dashed lines show RKL; curves average ten checkpoints and bands span their minimum and maximum. The final column gives the fraction of positions whose sampled-token logit is increased.

Low-probability and high student-entropy positions carry most of both KL objectives and their gradient energy (Figure 15). The lowest-probability 20% carry 69.0% of FKL, 68.3% of RKL, 74.7% of FKL gradient energy, and 74.9% of RKL gradient energy. The highest student-entropy 20% give closely matched shares of 69.7%, 68.6%, 74.5%, and 73.4%, respectively. Teacher entropy is more objective-sensitive: its highest 20% capture 70.4% of FKL gradient energy and 55.9% of RKL gradient energy. The direction curves are interpreted together with the energy curves; high-probability and low-entropy positions carry negligible gradient magnitude.

C.5.3 Joint Structure and Selection Frontier

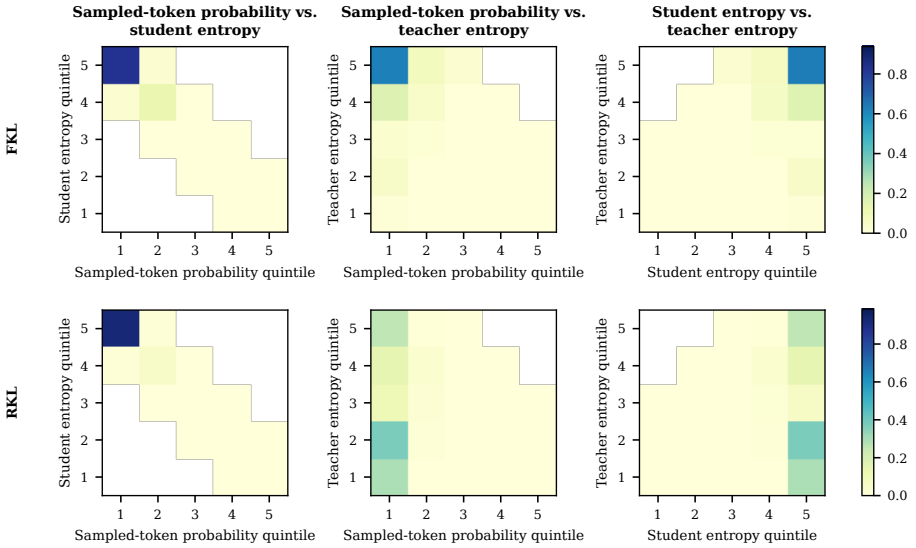


Figure 16: Joint gradient-energy allocation. Each axis divides positions into quintiles from low to high. The upper row shows FKL and the lower row shows RKL. Probability and student entropy isolate one shared high-signal region, whereas teacher entropy produces a stronger objective-dependent pattern.

The lowest-probability, highest student-entropy cell contains approximately 19% of positions and dominates both objectives (Figure 16). Probability and student entropy therefore identify nearly the same high-signal region. Their individual rankings reach the same selection frontier as the combined ranking.

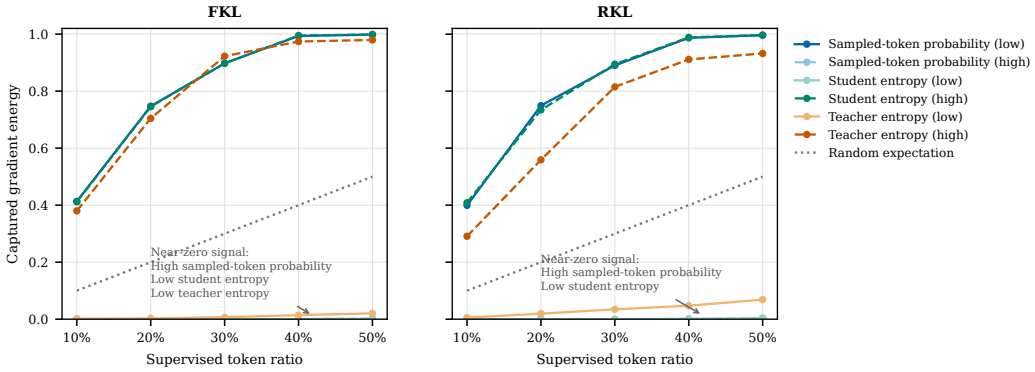


Figure 17: Gradient energy captured as the supervised token ratio increases. The gray diagonal is random selection. Low sampled-token probability and high student entropy rise fastest for both FKL and RKL; the arrows collect the opposite tails whose captured energy remains near zero.

At a 30% budget, low sampled-token probability captures 89.7% of FKL and 89.0% of RKL gradient energy; high student entropy captures 89.8% and 89.5%. At 40%, the corresponding shares reach 99.3% and 98.7% for probability and 99.5% and 98.8% for student entropy. Every checkpoint exceeds 90% capture at this budget, and the Top-64 training objective reproduces the same frontier. Sampled-token log-probability matches the entropy selector while using values already recorded during rollout.

C.5.4 Persistence Across Training

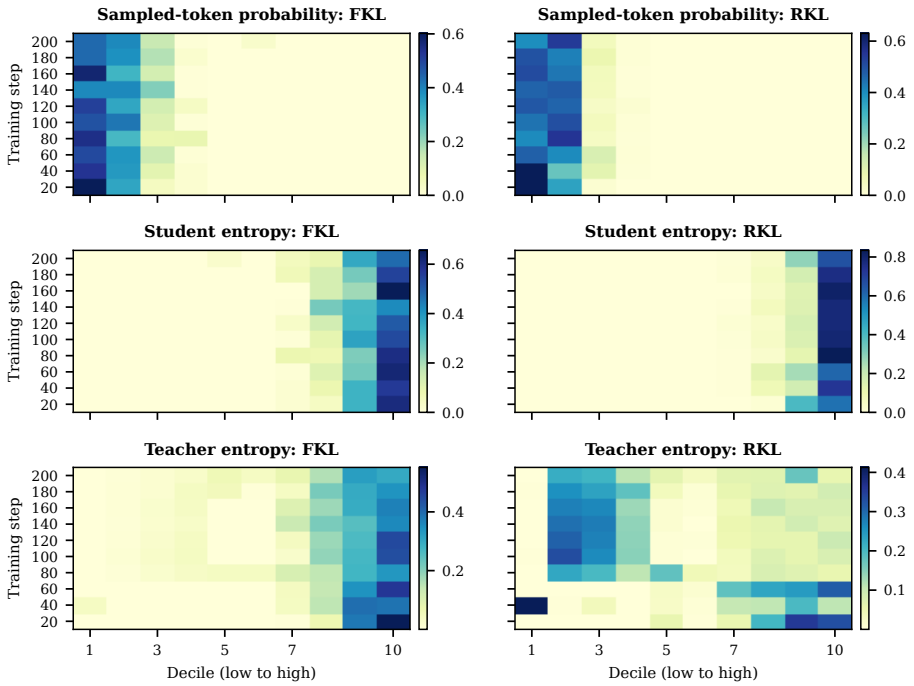
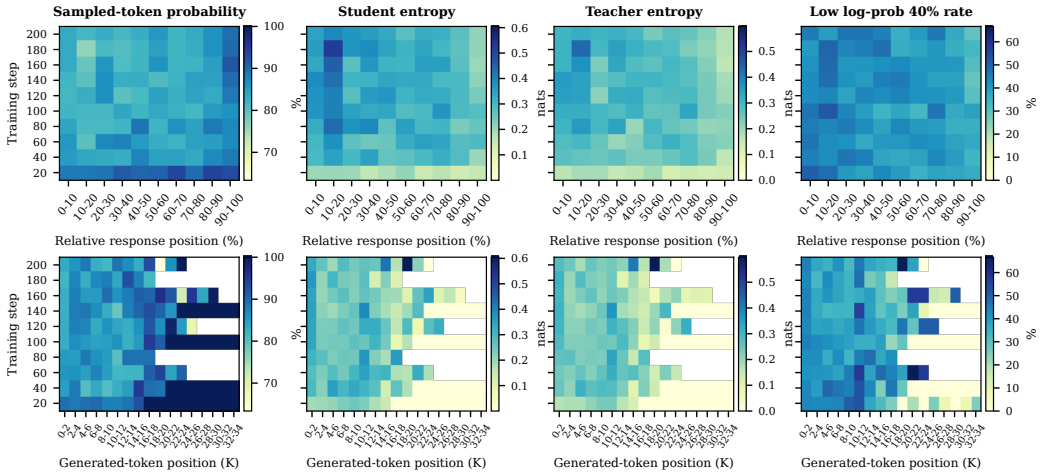


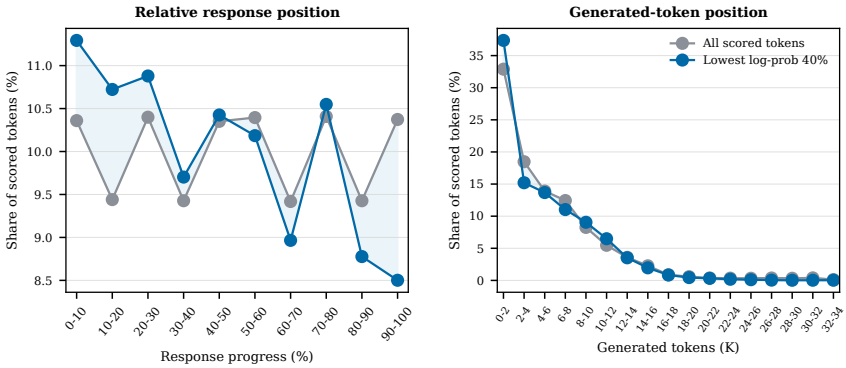
Figure 18: Gradient-energy allocation from step 20 to 200. Columns show FKL and RKL; rows group positions by sampled-token probability, student entropy, and teacher entropy. Darker cells carry a larger share of the checkpoint's gradient energy.

Low probability and high student entropy remain concentrated throughout all ten checkpoints (Figure 18). Their profiles also reproduce under the second rollout seeds at steps 20, 100, and 200, with correlations from 0.895 to 0.993. Teacher entropy varies more across training, especially for RKL, matching its objective-dependent marginal and joint profiles.

C.5.5 Relation to Sequence Position



(a) Probability and entropy by relative and absolute response position.



(b) Distribution of positions selected by the lowest-probability 40%.

Figure 19: Relation between token-level uncertainty and generated response position. Relative bins span 10% of each response and absolute bins span 2,000 generated tokens.

Relative response progress produces a modest, task-dependent shift: the fraction selected by LowP-40 changes from 44.7% in the first 20% of a response to 35.1% in the final 20%, with a flatter profile on reasoning trajectories. Absolute positions through 16K fluctuate without a monotonic pattern. Of all LowP-40 positions, 77.3% occur before 8K, closely matching the 77.7% share of all measured positions. The values beyond 16K reflect the small subset of trajectories that reach those lengths. Sequence position therefore provides a weak basis for allocating supervision. LowP-40 ranks eligible tokens within every complete trajectory, preserving coverage across the full rollout while concentrating FKL computation on high-signal positions.

C.6 GRPO AND OPRD BASELINES

GRPO and OPRD use the Qwen3-4B Sparse Ref. common-start checkpoint, the same sparse attention configuration, and the same 200-step training budget as the main recovery methods. Each optimizer step consumes 16 sparse-policy trajectories from the shared code-aware prompt pool. GRPO constructs each step from four scheduled prompts and four fixed-seed completions per prompt; OPRD uses the shared 16-visit step schedule.

GRPO. Every optimizer step begins with a fresh rollout from the current sparse policy. For response i in prompt group g, the verifier reward r_i defines the group-relative advantage

$$A_i = \frac{r_i - \mathrm{mean}_{j \in g}(r_j)}{\mathrm{std}_{j \in g}(r_j) + 10^{-6}}. \tag{19}$$

The likelihood ratio uses the behavior probability recorded during rollout, $\rho_{i,t} = \exp(\log q_\theta(a_{i,t} \mid s_{i,t}) - \log q_{\mathrm{beh}}(a_{i,t} \mid s_{i,t}))$. We optimize

$$\mathcal{L}_{\mathrm{GRPO}} = -\,\mathrm{mean}_i\, \mathrm{mean}_{t \in M_i}\, \min(\rho_{i,t} A_i, \mathrm{clip}(\rho_{i,t}, 0.8, 1.2) A_i)\,, \tag{20}$$

where M_i contains the valid response tokens. Math uses exact-answer rewards, code uses test-case rewards, and long-context tasks use their normalized task scores. Sampling uses temperature 1.0, top-$p = 1.0$, and top-$k = 0$. The loss first averages over response tokens and then over trajectories. We use a learning rate of 10^{-5}.

OPRD. OPRD uses 10 rounds of 20 updates, with 320 fresh sparse-policy trajectories collected at the start of each round. The frozen dense teacher and sparse student read the same token sequence. For every transformer layer, we minimize raw hidden-state MSE over the final 2,000 valid response positions:

$$\mathcal{L}_{\mathrm{OPRD}} = \frac{1}{|\mathcal{L}|} \sum_{l \in \mathcal{L}} \frac{1}{\sum_t m_t} \sum_t m_t \frac{1}{d} \left\| h_{\theta,t}^{(l)} - \mathrm{sg}\left(h_{T,t}^{(l)}\right) \right\|_2^2, \tag{21}$$

where m_t selects supervised response positions and d is the hidden width. The loss coefficient is 1.0, teacher activations are stop-gradient, and the learning rate is 10^{-5}.

C.7 EVALUATION PROTOCOLS

Mathematical reasoning. AIME 2026 contains 30 problems and HMMT February 2026 contains 33 problems. We sample eight thinking-enabled generations per problem using temperature 0.6, top-$p = 0.95$, and top-$k = 20$. The maximum generation length is 32K tokens for Qwen3-4B and Qwen3-14B, and 48K tokens for Ministral-3-14B. Mean@8 is sample accuracy, Pass@8 is the fraction of problems solved by at least one sample, and hit-cap is the fraction of generations that reach the output limit. Answers are scored with the fixed MathArena evaluation protocol.

Code. LiveCodeBench v6 contains 175 held-out problems. We generate one thinking-enabled greedy completion per problem, with a maximum generation length of 32K tokens for Qwen3-4B and Qwen3-14B, and 48K tokens for Ministral-3-14B. Hit-cap is the fraction of generations that reach the corresponding model's generation limit.

Long-context tasks. RULER evaluation contains 13 synthetic tasks at 8K, 16K, and 32K context lengths; we report the macro average across the 39 task–length combinations. LongBench contains 21 datasets and is reported as the macro average of their official metrics. Both suites use greedy non-thinking generation.

C.8 EFFICIENCY MEASUREMENT

The left panel of Figure 4 measures Qwen3-4B decode throughput on one RTX 6000D GPU. It compares page-native sparse execution with FlashAttention-2 dense execution in BF16 at 8K, 16K, and 32K context lengths and batch sizes 1, 4, and 8. Each setting uses 8 warmup tokens, 128 measured tokens, and five repeats; speedup is the ratio of median latencies.

The right panel measures complete HMMT requests on the same GPU and runtime. For each endpoint, we generate eight samples for each of the 33 problems and measure elapsed time from prefill through the end of decode. Model loading is excluded. Request-level speedup is the ratio of aggregate Dense Ref. time to aggregate endpoint time over the matched requests. Pass@8 and hit-cap are computed from the same generations; circle area visualizes the hit-cap rate.

C.9 Training-Time Overhead of Fixed-State and On-Policy Distillation

We report wall-clock time on the eight-GPU RTX 6000D setup specified in Section 4. Rollout collection uses four concurrent collector processes per GPU. Total recovery time includes trajectory rollout, teacher scoring, parameter updates, data preparation, checkpointing, and round orchestration.

Table 16: Total training wall-clock time on eight NVIDIA RTX 6000D GPUs.

Method	Rollout schedule	Total wall-clock time (h)
FKL	Dense teacher, generated once	26.40
OPD-FKL	Sparse student, refreshed every 20 steps	22.70
OPD-FKL LowP-40%	Sparse student, refreshed every 20 steps	19.04

OPD-FKL uses sparse rollout and refreshes its trajectories throughout training, reaching 22.70 hours for one complete run. LowP-40 preserves this refresh schedule and reduces total recovery time to 19.04 hours.

The fixed-state trajectories can be reused across training seeds. After their one-time 18.33-hour construction, each additional FKL run costs 8.07 hours. For N training runs, its average cost is therefore

$$\overline{C}_{\mathrm{FKL}}(N) = 8.07 + \frac{18.33}{N} \text{ hours per run.} \tag{22}$$

Each OPD-FKL run follows its own sequence of student checkpoints and visited states, so policy-dependent collection remains part of every run.

Table 17: Average wall-clock per training run when fixed-state FKL trajectories are reused across training seeds. On-policy trajectories follow each run's evolving student and are recollected independently.

Training runs	Static FKL (h/run)	OPD-FKL (h/run)	LowP-40 (h/run)
1	26.40	22.70	19.04
2	17.24	22.70	19.04
3	14.18	22.70	19.04
Many	8.07	22.70	19.04