MASKLIFT: EXACT MASK REPLAY FOR RL UNDER KV-CACHE COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Online RL post-training spends most of its memory budget on the rollout phase, where the Key–Value (KV) cache of long prompts is the part of the footprint that grows with context and batch. Compressing that cache relieves the pressure but breaks the assumption the policy gradient rests on: trajectories are sampled under a compressed context while the learner scores them under the full one, so the update is off-policy by construction. Remedies for it correct the resulting estimator; we remove what it estimates. MaskLift records the discrete retention mask the rollout compressor produced for each sequence and replays it in the learner's attention, so the sparse/dense importance ratio is identically 1 rather than reweighted or rejected. That identity is exact at the parameters that produced the trajectory, and it is a property of the *sparse* policy; the dense policy deployment uses inherits the improvement only through a second term, which detaches the RL-improved sparse pass and fits the dense pass to it. On GSM8K under our retention rule, where *every* uncorrected-learner run finishes below the untrained checkpoint's 40.7, MaskLift reaches 72.8 over its surviving runs against 72.3 for a dense-cache learner that never compressed anything, collapsing on 1 of 5 runs against 0 of 5. It tracks that learner on three further datasets, and we map where the mismatch is severe and where it costs nothing. Qwen3-1.7B, contexts up to 4K; the memory saving is confined to the rollout-decode phase.

1 INTRODUCTION

Large Language Models (Brown et al., 2020; Touvron et al., 2023; Yang et al., 2025) are post-trained with reinforcement learning (Ouyang et al., 2022; Bai et al., 2022), commonly with GRPO (Shao et al., 2024). In long-context settings (Ding et al., 2024) the rollout phase is memory bound: the Key–Value (KV) cache (Vaswani et al., 2017; Kwon et al., 2023) grows linearly in sequence length and batch size, and it, not parameters or optimizer state, caps context and batch per device.

Rollout-time KV compression such as SnapKV (Li et al., 2024) or H2O (Zhang et al., 2023) relieves the decode footprint but creates a structural mismatch: generation samples from a policy conditioned on a sparse context (π_{sparse}) while the learner scores those trajectories over the full context (π_{dense}), violating the policy-gradient assumption that the differentiated policy generated the data (Sutton & Barto, 2018). The cost is not subtle, but neither is it constant. In our diagnostic ablation on GSM8K 5 of 5 mismatched-learner runs finished below 40.7, the score of the same model with *no* RL at all: uncorrected, compression at rollout can make training worse than not training. On a short, redundant task the same configuration costs little. What decides this is how much information eviction destroys (Section 4.4). Remedies proposed for this gap are statistical: rejection sampling and reweighting (Luo et al., 2026) correct the estimator rather than removing the mismatch (Appendix A).

MaskLift takes an architectural route in two steps. First, it records, during the sparse rollout, the discrete information set the actor attended to (a binary retention mask over key tokens) and replays that exact mask in the learner's attention logits. Under exact replay the sparse/dense importance ratio is identically 1: not corrected, not bounded, but absent.

Replay buys that alignment by changing what is optimised. The masked learner improves the *sparse* policy, the one that generated the data, while inference uses the *dense* one, so a second step carries the improvement across. We add a KL term between the two passes in which the masked pass is detached and serves as the teacher, and the dense pass is the student; Remark 1 says why that way

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.

round. Over 18 seeds paired by index the transfer term gives the better run on 14 (median $+11.5$ points) without narrowing the spread over all 18; Section 4.2 gives the comparison and the collapse counts it conditions on.

Our contributions are fourfold. **(i) Diagnostic:** we isolate the sparse-rollout / dense-learner mismatch that inference-time KV compression creates in an online RL loop and exhibit it with a mismatched learner (an ablation, not a competing algorithm), which on GSM8K trains to below the untrained checkpoint, and we map it across regimes to show where it does not. **(ii) Method:** we record the KV visible set and replay it as the exact retention mask inside an otherwise dense learner pass, so the sparse/dense importance ratio is identically 1; a same-size *wrong* mask replayed instead is at best tied ($1.00\times$) and is up to $9.2\times$ worse, by a factor that moves with the regime. **(iii) Transfer:** we close the train/deploy gap alignment creates by detaching the masked pass and fitting the deployed dense pass to it, with the evidence over 18 paired seeds that adding the term helps; we do not separate transfer from consistency regularisation. Section 5 places both steps against the prior work whose ingredients they compose. **(iv) Evidence:** we report four datasets over multiple seeds, state which settings are design choices rather than sweep outcomes and which selection was scored on the wrong split, and account for what the compressed rollout cache buys and which memory peaks it leaves untouched.

2 Background: The Asymmetric Context Between Rollout and Learner

In GRPO (Shao et al., 2024) the actor samples K responses $\{y_i\}$ per prompt x and the learner updates θ with group-relative advantages $\hat{A}_i$. Under KV compression at generation time (e.g., SnapKV (Li et al., 2024)) each token is conditioned on a *subset* of the key/value entries; call this constrained policy π_{sparse}. The learner, scoring the same trajectory over the full context, is a different policy π_{dense}: the data are off-policy for the objective optimised, and correcting that requires an importance ratio $\pi_{\text{dense}}/\pi_{\text{sparse}}$ which is neither 1 nor concentrated. Prior work absorbs the mismatch statistically, via reweighting and rejection sampling (Luo et al., 2026), paying in gradient variance or discarded trajectories. MaskLift instead makes the learner condition on the key set the actor used, and then treats the residual sparse/dense gap as a *transfer* problem rather than an estimation problem.

3 Methodology

Scope of the contribution. No ingredient here is new on its own; the claim is the composition, and Section 5 places it against R3 and QaRL. MaskLift *records* the rollout's discrete information set, *replays* it in the learner, and then carries the resulting improvement from the sparse policy to the dense one deployment uses (Figure 1).

3.1 Step 1: Replaying the Rollout Retention Mask in the Learner

Recording and replay. During rollout we record which key positions survive each eviction. For exposition, $M_i \in \{0,1\}^L$ is one binary vector per response y_i with $M_i[j] = 1$ iff key token j was available to the actor; our eviction drops the evicted keys from the decode cache uniformly across heads and layers, and the mask is recorded at that same granularity, which is what makes replay exact. A rule selecting different positions per head must be recorded per head (Appendix J.4). Prefill is dense, so eviction and its saving apply to the decode phase. In the learner we inject M_i additively into the pre-softmax attention logits,

$$\mathrm{Attn}(Q,K,V,M_i) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}} + \mathcal{M}_{\text{causal}} + \mathcal{M}_{\text{retain}}(M_i)\right)V, \tag{1}$$

where $[\mathcal{M}_{\text{retain}}(M_i)]_{q,j} = -\infty$ whenever $M_i[j] = 0$ and q is a *response* position, else 0. Two details are load-bearing. The condition is on *key* tokens and broadcast across response queries, so post-softmax weight on evicted keys is exactly zero. And the restriction to response queries matches rollout: eviction happens during decoding, so prompt positions were computed densely by the actor and must be scored densely by the learner. Masking prompt queries too, a natural but incorrect simplification, breaks the identity. Writing $\pi_\theta(y_i \mid x, M_i)$ for the masked policy, replay gives

$$\pi_{\text{sparse}}(y_i \mid x) \;/\; \pi_{\theta_{\text{old}}}(y_i \mid x, M_i) = 1 \tag{2}$$

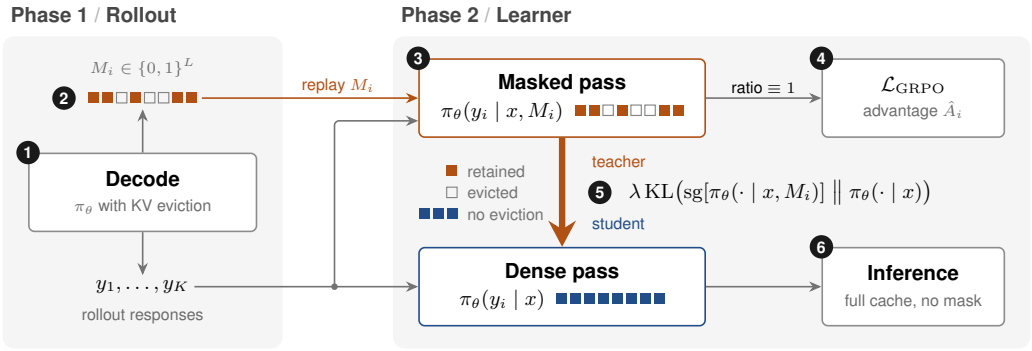


Figure 1: MaskLift. Phase 1 (rollout): **(1)** the actor decodes under a KV-eviction rule, and **(2)** the surviving key positions are recorded as the retention mask M_i. Phase 2 (learner): **(3)** both passes see the same responses, but only the masked one replays M_i, so the sparse/dense importance ratio is identically one, and **(4)** the policy gradient is taken from that pass. **(5)** A dense pass then runs without the mask, fitted with weight λ to the masked pass's detached distribution (Remark 1), while **(6)** inference uses the dense policy.

at the parameters θ_{old} that produced the trajectory: the surrogate is evaluated against the behaviour policy itself, so no importance ratio is estimated and no trajectory rejected. Eq. (2) is an identity, not an approximation. In `bfloat16`, the precision our training loop uses, the replayed learner's mean ratio deviation is 1.8×10^{-2} against 0.107 without replay over 18 configurations, and it does not vary with the eviction rule: the residual is floating-point error, not a modelling gap. The `float32` figures and the full sweep are in Appendix J.4.

What replay removes, and what it does not. Eq. (2) is the *sparse/dense mismatch* ratio, which replay makes identically 1 because both sides are masked. It is not the *policy-improvement* ratio $\pi_\theta / \pi_{\theta_{\text{old}}}$, which equals 1 only before θ moves; MaskLift eliminates the first, not the second, which is why we still clip. Total gradient variance remains (reward noise, advantage estimation, minibatch sampling), and this is not a proof of training stability.

Exact replay is a precondition. The identity holds only if the learner *replays the saved mask* rather than recomputing eviction, and the mask-to-sequence association must survive rollout workers, per-head layouts, packing, padding and re-sharding; lost or approximated, the masked policy is no longer the behaviour policy (Appendix J.4). A control replaying a *wrong* mask of the same size is at best tied ($1.00\times$) in ratio deviation and is far worse where the regime is severe, which is the sense in which the right mask and not merely a sparse one is what the method needs; no task score was measured under a wrong mask.

What alignment costs: an objective substitution. Replay changes the estimand. The masked learner performs *on-policy optimisation of the sparse policy* (it maximises the return of π_{sparse}, which generated the data, rather than estimating π_{dense}'s return from sparse rollouts), so it removes the ratio-product variance by substituting a different objective. On-policy is not unbiased: clipping and advantage normalisation bias the update (Appendix A.5). Since the trained model is deployed on the full context, a variance problem has become a train/deploy *objective mismatch*, and Step 2 addresses it directly rather than leaving it to weight sharing.

3.2 Step 2: Transferring the Improvement to the Deployed Policy

Step 1 leaves the RL signal on $\pi_\theta(\cdot \mid x, M_i)$ while inference uses $\pi_\theta(\cdot \mid x)$. To move it across, a second forward pass runs after the masked one with full dense causal attention and is fitted to it:

$$\mathcal{L}_{\text{Transfer}}(\theta) = \mathbb{E}\Big[\, \mathcal{D}_{\text{KL}}\big(\mathrm{sg}\big[\pi_\theta(\cdot \mid x, M_i)\big] \,\|\, \pi_\theta(\cdot \mid x)\big)\Big]. \tag{3}$$

The masked distribution is detached; the gradient of Eq. (3) enters the network only through the dense pass. Teacher and student share one θ and differ only by the recorded mask, so no separate teacher

model is needed, and the cost is one additional forward pass per step, measured in Appendix D rather than asserted.

Remark 1 (Which pass teaches which). The passes are distinguished by *where the reward signal enters*, not by how much context each sees. The policy gradient of Eq. (4) is computed under the replayed mask, so it acts on $\pi_\theta(\cdot \mid x, M_i)$ directly and reaches $\pi_\theta(\cdot \mid x)$ only through the shared parameters, which makes it the target and the deployed dense distribution the student. Both are evaluated at the same θ, before the update, so the asymmetry is in the computation graph, not in time. Appendix B gives the gradient and the ablation's limits.

3.3 Objective

The policy-gradient loss is the negative clipped GRPO surrogate,

$$\mathcal{L}_{\text{PG}}(\theta) = -\,\mathbb{E}\Big[\frac{1}{K}\sum_{i=1}^{K}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min\big(p_{i,t}\hat{A}_i,\ \text{clip}(p_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i\big)\Big], \tag{4}$$

with expectations over $x \sim \mathcal{D}$ and $\{y_i\}_{i=1}^{K} \sim \pi_{\text{sparse}}$, and with the *per-token* ratio $p_{i,t} = \pi_\theta(y_{i,t} \mid x, y_{i,<t}, M_i)/\pi_{\theta_{\text{old}}}(y_{i,t} \mid x, y_{i,<t}, M_i)$: the clip acts on each token, the clipped terms are averaged over a response's tokens and then over the group, and $\hat{A}_i$ is constant within a response. $|y_i|$ counts the sampled tokens up to and including the terminating EOS, or up to the generation cap when none is emitted; each response is scored in its own forward pass, so no padding is ever summed over. Because M_i hides evicted keys from *response* queries only, $t = 1$ is scored under the dense prefill, exactly as the actor sampled it. We retain GRPO's reference-KL term but set its coefficient $\beta = 0$ in every run and build no reference model (Appendix C). Here $\text{sg}[\cdot]$ is the stop-gradient and $\mathcal{D}_{\text{KL}}$ is the token-wise KL summed over the vocabulary and *averaged* over the response's generation steps: the normalisation matters, since summing instead would rescale λ by the response length. The full objective is

$$\min_{\theta}\ \mathcal{L}_{\text{total}}(\theta) = \mathcal{L}_{\text{PG}}(\theta) + \lambda\,\mathcal{L}_{\text{Transfer}}(\theta), \tag{5}$$

both terms minimised, as in Algorithm 1, which also gives the full procedure; Appendix C gives the hyperparameters.

What the theorem says. Correcting the mismatch after the fact means multiplying one importance ratio per generated token, and the variance of that product grows *exponentially* in the response length. Replay makes the product identically 1, so the factor is absent rather than estimated or clipped.

Theorem (Informal) (Variance of the full-trajectory importance-ratio estimator). Correct the sparse-rollout / dense-learner mismatch with the *product* of per-token importance ratios over a response of length L. If each per-token ratio has conditional variance at least $\sigma_0^2 > 0$, the gradient–advantage product is bounded away from zero, and its mean is bounded above independently of L, that estimator's variance is $\Omega\big((1+\sigma_0^2)^L\big)$. Under exact replay the product is identically 1, the factor is absent, and what remains is the variance of the on-policy estimator.

Appendix A proves it. One condition matters for Section 4: the bound is on the *uncorrected full-trajectory product*, not on the clipped per-token ratios practitioners run, so we attribute exponential variance to no baseline. The transfer term is not covered; its gap is measured, not bounded.

4 Experiments

Setup. Qwen3-1.7B, 200 GRPO steps, learning rate 2×10^{-6}, clipping $\epsilon = 0.2$, a position-proximity retention rule (Appendix G.1), not SnapKV's attention scorer (Li et al., 2024), which Appendix H reports on separately, at the budget stated per experiment, H100/H200 NVL in BF16 with `float32` master weights; K is 8 on GSM8K and TL;DR, 4 on HotpotQA and GovReport (Appendix C). Rewards are *programmatic* (exact match for GSM8K (Cobbe et al., 2021), ROUGE-L for TL;DR (Stiennon et al., 2020) and GovReport (Huang et al., 2021), token F1 for HotpotQA (Yang et al., 2018)): online RL with automatic rewards, not RLHF. On the summarisation sets reward and metric share a functional form on disjoint splits, so those columns measure how efficiently a method hill-climbs ROUGE-L, not summary quality (Appendix C).

Protocol discipline. Each cell is n independent training runs differing in seed, summarised by the collapse count and the mean and spread of the survivors (Appendix C.2). Runs are not reproducible across our two cards, and four comparisons are confounded with it: clip + reject on GSM8K, the TL;DR arms added in revision, the 50% retention block below, and $\lambda = 0.1$. Table 9 gives the card of every cell, Table 21 of every λ, and Appendix C.5 measures the effect on 13 seed-matched pairs. λ was selected on the GSM8K validation split, which Tables 6 and 21 also report, and the retention budget was fixed by design at 30%. The learning rate is the exception and we report it as one: its calibration was scored on the first 400 questions of the GSM8K test column, which is the split Table 5 reports, and Appendix C gives that grid, its criterion and the rerun on validation. Two seams we did not remove are disclosed in Appendix C: on GSM8K the clip-and-reject row, the published objective and every collapse floor were scored at a shorter generation cap than the remaining columns, and clip + reject ran mostly on the other card. All arms are evaluated with the *full, uncompressed* context: eviction is a training-time intervention, so the benefit is rollout-decode capacity, not inference-time compression.

Baselines. *No RL*, the checkpoint every trained arm must clear; *Dense* GRPO at 100% KV, matched in data and optimiser settings but *not* in memory (Appendix I.1); *Naive* eviction, a rollout-evicting run with an uncorrected dense learner, a **diagnostic ablation** rather than a competing algorithm; *Replay only*, mask replay at $\lambda = 0$; and *Clip + reject*, our simplified trajectory-rejection baseline in the style of Sparse-RL (Luo et al., 2026), the closest published system: our retention and mask, a dense learner, per-token ratios clipped to $[0.8, 1.2]$, and rejection of the most deviated quarter of each group. It is **not** the published algorithm, which leaves its sparsity ratio outside the clip and rejects on a support test; we implemented that algorithm as written and ran it on all four datasets (Appendix C.1).

4.1 Recovering What the Mismatch Destroys

Reading the figure. On GSM8K the mismatch is not a tax but a failure: 5 of 5 naive runs finish below the untrained checkpoint, so under this eviction rule training on an uncorrected compressed rollout is worse than not training at all (Appendix H reports the same pair under SnapKV's scorer, where it is a tax rather than a failure). Replay recovers most of that ground alone (67.9 against a 40.7 floor), and the transfer term brings the mean of the non-collapsed runs to 72.8 against 72.3 for a learner that never compressed anything. Both are conditional means and the arms collapsed at different rates (1 of 5 against 0 of 5); over every run they are 61.6 and 72.3, the whole gap being that one collapse (Appendix C.6).

The other three columns separate much less, and Section 4.4 says why: TL;DR, HotpotQA and GovReport sit where eviction destroys little. The arm means span about six points on HotpotQA (uncorrected lowest, 55.4 against dense's 61.5), about five on TL;DR and about one on GovReport. **We can establish no quality advantage for MaskLift on these three and claim none**, under this eviction rule; under SnapKV's own scorer Appendix H orders the two arms the same way on all four tasks and separates them on two.

The two rejection baselines. 3 of 3 runs of our clip-and-reject arm finish below the untrained checkpoint on GSM8K, while on TL;DR the same configuration trains without a collapse to 19.7 against a 16.0 floor. That row is our simplification of the published algorithm, not evidence about it, and two of its three GSM8K runs are on the card the other arms mostly did not use. The published objective as written does not collapse in that corner: its support test vetoes 81.8% of the trajectories it scored and loses the whole group at 76% of its steps, which then take no update at all, while on TL;DR it vetoes 19.8% and trains (Appendix C.1). Both devices act on the estimator built from the ratio rather than on the ratio, and neither moves it down (Figure 2(b)): clip + reject sits where the uncorrected learner does and the published objective above it. What the pair supports is narrow: in this regime correcting that estimator was not sufficient in our hands, declining to train on it bought little, and removing the mismatch was what worked.

Resolution. GSM8K's 1319 problems give a 95% Wilson half-width of about ± 2.4 pp for a single run, but the seed-to-seed spread is several times that (7.8 points s.d. for the dense arm), so we read only large separations as results (a collapsed arm against a surviving one) and treat a point or two as closeness, not equivalence.

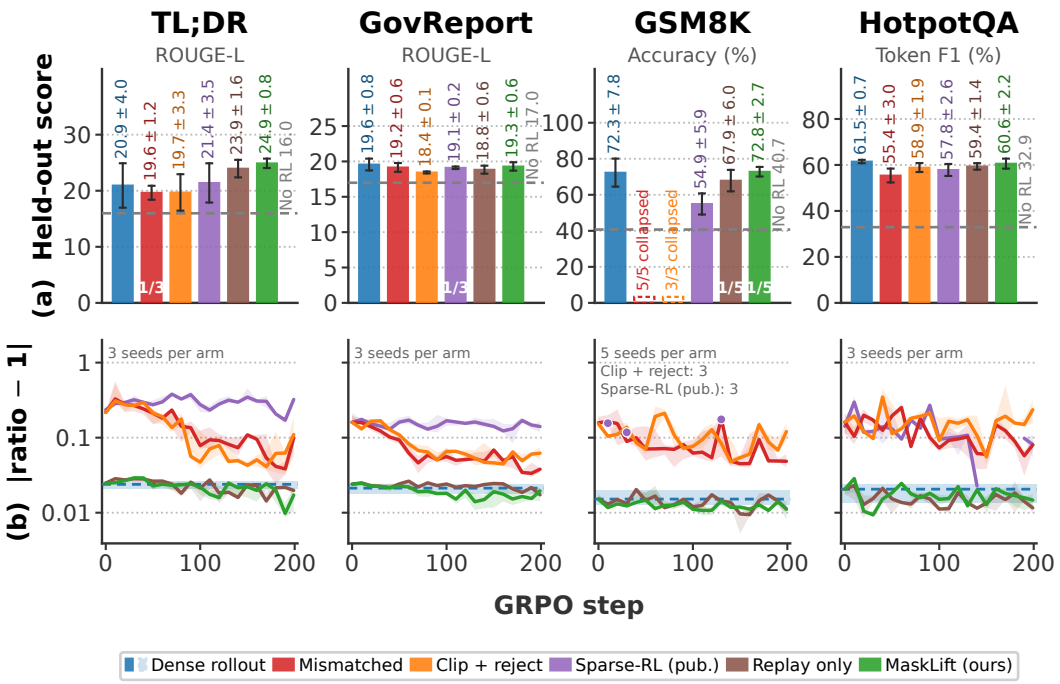


Figure 2: **(a)** Held-out score by task and arm: bars are the mean over the runs that did *not* collapse, whiskers their s.d., the dashed line the untrained checkpoint, which is also the collapse threshold. An arm with no bar had every run finish below it; a dashed outline says it ran and died rather than that it was never run, and partial collapses are annotated (collapsed/total). Metrics differ by panel, and HotpotQA is scored on its validation split, the only public one, on which nothing was tuned. **(b)** The deviation the learner actually computes: $\frac{1}{|y_i|}\sum_t |p_{i,t} - 1|$ over the group's responses, median across seeds (band: interquartile range), every tenth GRPO step and the last, shared log axis. Read off the learner's forward pass *before* that step's update, so it is replay error at fixed θ_{old}, not drift after one. The dense arm has no mismatch to remove, so its band is `bfloat16` recompute noise and marks the floor. The published objective's series is conditioned on passing its support test, a threshold on the quantity plotted, so it is not comparable with the others; on GSM8K that leaves 3 of 63 logged step-rows, drawn as dots with no band. Exact per-cell numbers: Table 5.

4.2 What the Transfer Term Adds

Mask replay aligns the learner to the sparse policy; the transfer term moves that improvement into the dense policy inference uses. The two arms differ by that term alone, over 18 runs per arm paired by seed, on the GSM8K validation split.

How these runs are summarised. Runs ending below the untrained checkpoint occur in every arm of Table 5 trained under eviction and in no dense run (12 of 66 against 0 of 14 in Table 5; 3 and 2 of 18 here), so one mean over that bimodal shape describes neither mode (Appendix C.2).

What the term buys. Paired by seed index, MaskLift scores higher on 14 of 18 seeds and lower on 4, median difference +11.5 points; on the 13 seeds where neither arm collapsed it is higher on 11, median +11.3. Among runs that did not collapse the mean moves +9.3 points, from 70.7 to 80.0, and the s.d. falls from 7.9 to 5.3; those two are computed on 15 and 16 different runs, and over all 18 the s.d. are 20.0 and 20.5 about all-run means of 63.4 and 73.2. The typical outcome is better; we do not claim the term makes a single run less seed-sensitive.

A one-run difference in collapses (3 against 2 of 18) is not evidence that the term prevents them, and what causes a collapse is unresolved. This is the only arm-level comparison here whose runs are seed-paired, and the only one we test: an exact sign test on the 18 signs gives $p = 0.031$, with the caveat that 18 was reached by extending an inconclusive 6-seed pilot (Appendix C.3).

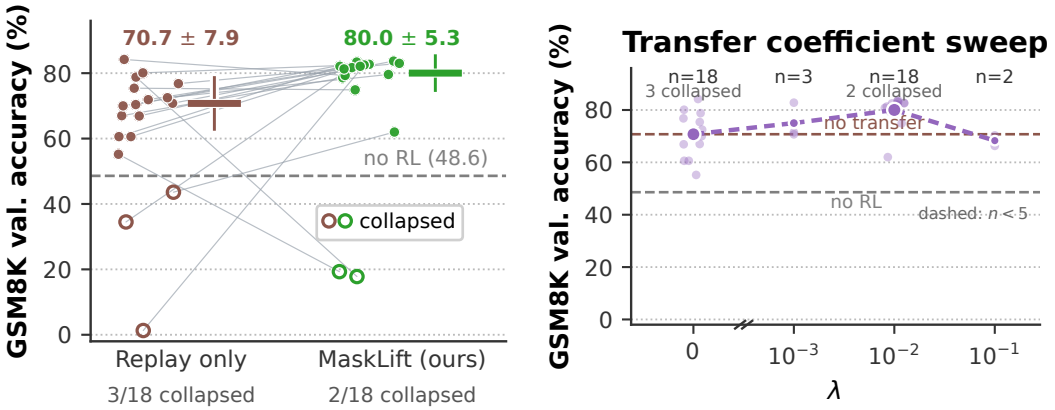


Figure 3: **Left:** every one of the 18 paired seeds, each seed's two runs joined by a hairline. Hollow markers finished below the untrained checkpoint (dashed line) and count as collapsed; bars and whiskers are the mean and s.d. of the runs that did not. **Right:** the λ sweep with every non-collapsed run shown. Marker area scales with the number of runs behind each mean; a segment touching a mean with fewer than five runs is dashed, and only $\lambda = 0$ and 10^{-2} are well sampled, so the interior shape is not established. Numbers: Tables 6 and 21.

4.3 What the Compressed Rollout Cache Buys, and Its Cost

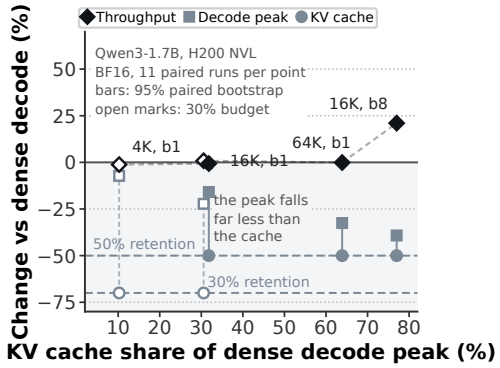


Figure 4: What compression buys, measured as a rollout runs it: dense prefill, eviction from the full cache, then decode (Qwen3-1.7B, one H200 NVL, BF16, 128 decoded tokens, 11 paired runs per configuration on an empty card; bars are a 95% paired bootstrap). The cache falls in proportion to the budget and the decode peak by less, since the weights dominate at this model size. The episode peak does not fall at all: both arms prefill the same context before either evicts. Only one configuration shows a throughput gain of appreciable size. The two 16K, batch-1 configurations share an x and are drawn ± 0.6 points apart so both are visible; that offset is the only quantity here that is not a measurement.

The compressed rollout cache buys decode-phase working set and costs one extra forward pass per learner step. We measure it the way a rollout runs: dense prefill, eviction from the full cache, then decode (Figure 4). The post-eviction cache falls in proportion to the budget, and the decode *peak* falls by less (5.61 to 4.72 GiB at 16K, batch 1, 50%) because the weights dominate at this model size. Two things do *not* follow. The peak over the whole episode is unchanged, identically so in all 5 configurations, because both arms prefill the same context first: eviction buys working set, not the memory the card must have. And decoding speeds up in 1 of the 5 configurations, not as a trend: +21.0% at 16K and batch 8, interval $[+20.8, +21.3]$. Of the rest, one is resolved and negligible and 3 cannot be signed, including 64K at batch 1, where the cache is the largest share of the decode peak of any single-sequence configuration we ran (Appendix D). Eviction also cannot reduce the learner peak, which we do not measure.

4.4 Sensitivity: λ, Retention, and Whether the Mask Is the Right One

The mask must be the recorded one, not merely a sparse one. A random mask of the *same size* is never better than exact replay and is up to 9.2× worse across 4 configurations, and in one of them no worse at all (Figure 6). The range is the finding, and it tracks the *budget*, not the severity of the mismatch: both 50% cells cost more than either 30% one, because at 30% the protected windows nearly fix the retained set (Appendix G.1). The mask also need not come from our rule: Appendix H

repeats the uncorrected and corrected arms under SnapKV's own scorer, with a separate top-k per layer and per KV head, where the recorded mask still replays to `float32` round-off and the correction still orders the two arms on all four.

How much the mismatch costs depends on the prompt, not only the budget. Over 60 cells the unreplayed deviation spans a factor of 19, moving with the budget and with the few-shot count, while replay holds it flat in every one. Adding shots changes prompt length, content and question position together, so we report the trend and do not attribute it to redundancy alone. The mildest corner is many shots at a light budget, and a study placed there sees almost none of this.

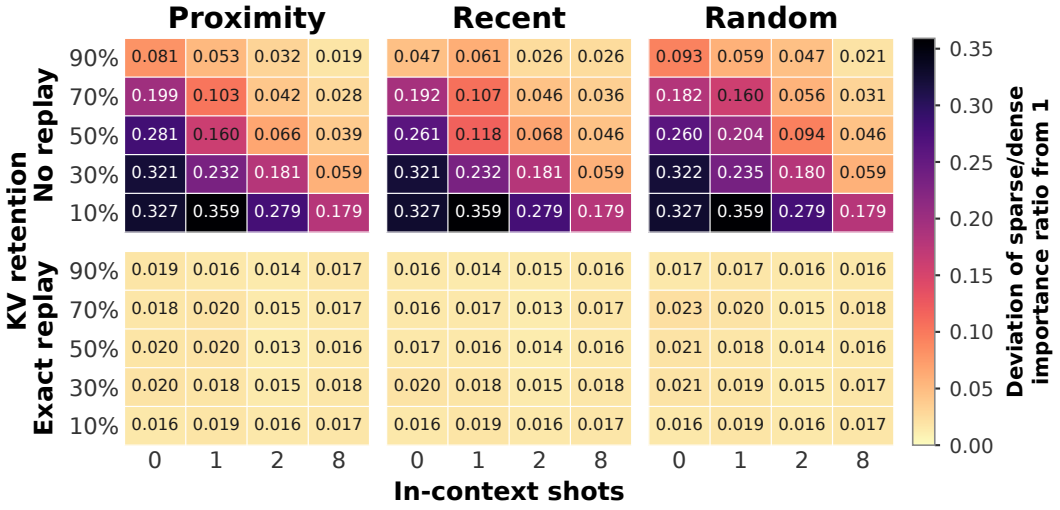


Figure 5: **The regime decides how much the mismatch costs; replay does not care, and does not care which rule was used.** Mean deviation of the sparse/dense ratio from 1 over 4 shot counts $\times$ 3 eviction rules $\times$ 5 budgets, 48 of them distinct, because all three rules coincide at 10% and two of them at 30%, 32 sequences per cell, one shared colour scale. No replay (top): 0.019–0.359, a 19$\times$ range set by the shot count and the budget. Exact replay (bottom): 0.013–0.023 in all 60 cells and flat across the three rules. The bottom two rows are not three independent rules: fixed head and tail windows leave 0 or 1 position free there (Appendix G.1).

Those measurements are taken outside training; Figure 2(b) reads the deviation the learner actually computes, and both replaying arms sit on the dense arm's `bfloat16` floor on all four datasets while the uncorrected learner sits 2.8–5.1$\times$ above it, clip + reject 3.2–7.4$\times$, and the published objective 5.7–12.4$\times$, the highest of any arm.

λ. The transfer coefficient is not self-limiting in either direction, and a validation sweep (Appendix J) is what selects it. Too small and the deployed policy does not follow the improvement; too large and the transfer target competes with the reward. At $\lambda = 0.1$ the two runs we have sit below the 10^{-2} mean and inside the $\lambda = 0$ range, which describes two runs rather than a harm boundary. We use 10^{-2} throughout.

Retention. Repeating the two GSM8K arms at a 50% budget separates the regime from the method. The uncorrected learner is still mostly destroyed there (2 of 3 runs collapse, against 5 of 5 at 30%), while MaskLift does not separate across the two budgets (72.2 ± 5.7 over 3 runs, 0 collapsed, against 72.8); those 3 runs are all on machine B while the 30% arm they are read against is mostly machine A (Table 9), and the gap is inside both the seed spread and the cross-card band of Appendix C.5. The collapse count is also the only statistic that moves that way: the uncorrected runs finish at $55.50, 3.03, 1.52$, all-run 20.0 ± 30.7 against 36.8 ± 3.5 at 30%, so failure is *deeper* at the looser budget while the deviation we call the driver is lower there (0.160 against 0.232). We report the reversal and do not explain it. This is one dataset at two budgets, so it says nothing about graceful degradation in general (Appendix J.3).

5 Related Work

Memory-Efficient RLHF and GRPO. RL post-training via PPO (Schulman et al., 2017) requires several model copies and GRPO (Shao et al., 2024) removes the critic, but the rollout phase remains bottlenecked by KV cache allocation (Kwon et al., 2023); we target that cost and are orthogonal to the alignment objective (Ouyang et al., 2022).

KV Cache Compression in LLMs. Eviction methods such as H2O (Zhang et al., 2023) and SnapKV (Li et al., 2024) keep a subset of key tokens by accumulated attention score; StreamingLLM (Xiao et al., 2024) exploits attention sinks, KIVI (Liu et al., 2024) lowers the precision of what is kept, multi-query attention (Shazeer, 2019) shrinks it by construction. All are designed for inference, where the compressed pass is the only pass; in an RL loop the learner scores the sequence again, and MaskLift treats any *discrete* eviction rule as a black box.

Off-Policy Correction, Distillation, and Trained Sparse Attention. Off-policy correction is classical (Sutton et al., 1999; Sutton & Barto, 2018) and importance-sampling variance is normally controlled by truncating the ratio (Schulman et al., 2015; 2017; Mnih et al., 2016), today at sequence level (Zheng et al., 2025) or as a truncated correction for rollout/training mismatch (Yao et al., 2025): a variance–bias trade, so we treat an uncorrected full-product ratio as a diagnostic reference point, not the estimator practitioners run. A milder, continuous form of our mismatch already arises between inference-engine rollouts (Kwon et al., 2023; Zheng et al., 2023) and a separate training stack (Shoeybi et al., 2019), and aligning the compute path rather than the estimator is an established response: R3 (Ma et al., 2025) replays the inference engine's routing decisions in the learner, QaRL (Gu et al., 2026) matches the learner's forward to the rollout's quantization. Record-and-replay is theirs, not ours; MaskLift applies it to a third component, the KV visible set, which is also the first whose replay leaves the *deployed* policy untrained, hence the second step. Teacher-to-student KL transfer (Kullback & Leibler, 1951; Tunstall et al., 2023) is likewise standard and likewise not claimed as new; what differs is which side is detached and why (Remark 1). Sparse attention has been trained end-to-end (Beltagy et al., 2020; Zaheer et al., 2020; Yuan et al., 2025; Lu et al., 2025), but there the pattern the learner trains is the pattern deployment uses, architecturally fixed or jointly learned, so no train/inference gap arises; here it is chosen at runtime, per sequence, so the learner must be told which pattern was used.

Closest Prior Work. Sparse-RL (Luo et al., 2026) is the first framework we are aware of to target long-context RL under KV compression explicitly, combining sparsity-aware rejection sampling with importance reweighting; that correction is statistical, so its cost is paid in data, vetoing rollouts it has already generated and reweighting the rest. Appendix A lower-bounds the *uncorrected, full-trajectory-product* estimator's variance by $\Omega((1+\sigma_0^2)^L)$ in the length L, which motivates recording the information set instead. Sparrow (Zhou et al., 2026) attacks the same mismatch from the other end, holding the *lower tail* of the per-token mismatch above a threshold with an adaptive sparsity schedule rather than removing the ratio; the two are complementary, since its schedule governs how sparse the actor may be while replay makes the learner's ratio exact at whatever sparsity results, and its tail account is a live candidate mechanism for the collapses we report and cannot explain (Appendix I.1).

6 Limitations and Conclusion

Three scope conditions bound everything above. The theory covers Step 1 only: exact replay removes the sparse/dense ratio's variance contribution, not total gradient variance, and the transfer gap is measured rather than bounded. Our eager-attention learner caps trainable context, since an additive mask materialises the attention scores, so cost grows with the square of sequence length and GovReport is truncated to 4K tokens; kernels that take an arbitrary mask without materialising them exist and we did not use one. And every training run here is Qwen3-1.7B: the replay identity, the regime map and the wrong-mask control also hold on a 4B checkpoint (Appendix J.4), but no task score does, so nothing we report speaks to how either step behaves at a scale where the cache dominates the weights rather than the other way round. Appendix I.1 lists the rest. Within those bounds: replay removes the sparse/dense mismatch rather than estimating it, and the transfer term carries the improvement into the policy deployment uses.

REPRODUCIBILITY STATEMENT

We have organized the appendix so that every quantitative claim in the paper can be traced to a stated protocol. The appendix reports the full hyperparameter table for MaskLift and for every baseline (model, precision, KV retention ratio, retention scorer, number of GRPO steps, learning rate, K responses per prompt, ratio-clipping range, reference-policy KL and transfer coefficient λ), the per-dataset evaluation protocol and reward function for each of the four datasets, the configuration of the clip-and-reject baseline together with a statement of how it differs from the published Sparse-RL algorithm and that its rejection fraction, clipping range and retention ratio were fixed rather than swept per dataset, and the protocol for the systems measurements (single H200 NVL, BF16, which phase each number is sampled in, how many independent repetitions each throughput figure averages, and which of its columns are arithmetic from the model configuration rather than measurements). Algorithm 1 in Appendix O gives the complete training loop, including the components that the loss actually uses. The memory accounting for the two passes' logits is analytical and is derived in the appendix from the model's published vocabulary size and the batch and sequence shapes, so it can be recomputed without access to our cluster; the variance argument and its assumptions are stated in full in Appendix A, with its scope (the full-trajectory-product importance-ratio estimator) made explicit.

Two properties of our evidence should be read alongside these protocols. First, every task number is taken over independent training runs differing in seed, not a single run, and because a minority of runs collapses far below the untrained checkpoint in most arms trained under eviction, each cell reports how many collapsed alongside the mean and s.d. of those that did not. Second, the transfer coefficient was selected on the GSM8K validation split and the split each number comes from is stated in its caption; the learning rate's calibration was not clean, it was scored on the first 400 questions of the GSM8K test column, and Appendix C reports that calibration in full rather than describing it as a validation selection. The systems numbers were collected on a single device and software stack, so the throughput ratios, not the absolute values, are the transferable part. The capacity measurement reports a batch ceiling at two fixed contexts and no context ceiling.

Every figure is regenerated the same way and ships with the data behind it: the supplementary contains the tidy CSV each panel is drawn from and the script that draws it, and running those scripts from the unpacked archive reproduces all five data figures of this paper, Figures 2 to 6, drawn by six panel scripts, with no cluster access and no path editing; the rasters are byte-identical between runs and the PDFs differ in nothing (their creation timestamp is pinned), and the figures are set in a font that ships with `matplotlib` so a reader's own copy does not depend on what is installed locally. Figure 1 is a diagram and is drawn from the manuscript source. The complete reimplementation and every result file are included in the supplementary material, together with the scripts that turn those files into the numeric macros the manuscript uses. No figure in this paper is transcribed from a log by hand: regenerating the macros from the released JSON files reproduces the tables exactly, and a cell with no data behind it renders as a visible placeholder rather than a plausible number. The archive also ships one per-step trace per run (reward, gradient norm, ratio deviation, clip fraction, rejected count and minimum support ratio, response length, skipped and degenerate groups), filed under the machine that executed it, so the quantities the tables take from training rather than from an evaluated checkpoint are recomputable rather than only traceable to a CSV we wrote. The queue's own stdout logs are not shipped.

One provenance note belongs here rather than in an appendix. An earlier preprint on this problem, Zhu et al. (2026), was withdrawn by its authors in September 2026 because errors were found in its experimental results; its task numbers came from an implementation since established not to execute the method it described, and its systems numbers timed prefill together with decode in a benchmark whose compressed arm prefilled a shorter context. No number in this paper is taken from it. Everything reported here was measured with the reimplementation in the supplementary material, and the appendix marks each place where a claim of that kind was withdrawn rather than restated (Appendices D, E and M).

Reproducing the alignment property itself requires one implementation invariant: the retention mask must stay bound to the sequence it was recorded for. A reimplementation must persist, for every sampled sequence, the discrete mask over key positions actually used at rollout, and must re-apply that same mask to that same sequence in the learner (across rollout-worker to learner-worker transfer, attention heads and layers, sequence packing, padding and dynamic re-batching). If this mask-to-

sequence association is broken or only approximately restored, the importance ratio is no longer identically 1 and the guarantee degrades to an approximation whose error we do not characterize; the appendix states this failure condition and how to detect it.

AI USE STATEMENT

In this work we used generative AI tools for implementing methods, designing and executing experiments, and interpreting results, and for drafting and editing the text of this paper. They were also used to search for related work and to check the bibliography against publishers' and preprint servers' records; the citations that resulted, and the positioning built on them, are the authors' responsibility. The research question and the decision of what to investigate were set by the authors. The reimplementation used for every experiment reported here was written with AI assistance and then verified by the authors against properties that do not depend on the implementation being correct by assumption: the mask-replay identity of Eq. (2) is checked numerically against a hand-computed reference, the ROUGE-L implementation is checked to agree exactly with the standard `rouge_score` package, and each task's reward function is checked to return its maximum on the reference answer. Analysis code was likewise AI-assisted; its statistical conclusions were re-derived by the authors, and at least one intermediate conclusion was retracted after an author observed that a mean-and-standard-deviation summary was inappropriate for the bimodal distributions involved, which changed a reported finding.

We have not used generative AI tools to generate synthetic data or to produce any experimental measurement: every number in this paper is written by a training or evaluation run into a file, and the manuscript's figures are generated mechanically from those files rather than transcribed. The mathematical claims and the proof in Appendix A were drafted with AI assistance and checked line by line by the authors.

We have reviewed all AI-assisted work and we take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only publicly available datasets (Reddit TL;DR, GovReport, GSM8K, HotpotQA) and publicly released model checkpoints; it involves no human subjects, no new data collection, and no personally identifying information beyond what those public corpora already contain. MaskLift is an efficiency mechanism for RL post-training and is neutral with respect to the reward or preference signal it optimizes: it lowers the working memory a long-context rollout needs while it decodes, which lowers it equally for beneficial tuning and for tuning that removes safety behavior. It also introduces a specific technical risk worth naming: because rollouts and updates both condition on a compressed context, evidence that the retention scorer discards is invisible to the learner, so a safety-relevant instruction or piece of context can be dropped without any signal in the aggregate score. A second risk is measured rather than hypothetical: 18 of the 90 eviction-trained runs behind Table 5 and Appendix H finished *below* the untrained checkpoint, with no cause we could identify and no detector that fires before a run is scored, so a practitioner adopting this must score every run against that checkpoint rather than assume training helped. We therefore do not recommend KV-compressed rollouts for safety-critical alignment runs without a dense-context evaluation of the resulting policy. Appendix P discusses these considerations in more detail.

REFERENCES

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document Transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. LongRoPE: Extending LLM context window beyond 2 million tokens. In *International Conference on Machine Learning*, pp. 11091–11104, 2024.

Hao Gu, Hao Wang, Jiacheng Liu, Lujun Li, Qiyuan Zhu, Bei Liu, Binxing Xu, Lei Wang, Xintong Yang, Sida Lin, Sirui Han, and Yike Guo. QaRL: Rollout-aligned quantization-aware RL for fast and stable training under training–inference mismatch. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17051–17064, San Diego, California, United States, 2026. Association for Computational Linguistics.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1419–1436, 2021.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*, 2023.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning*, 2024.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, et al. MoBA: Mixture of block attention for long-context LLMs. *arXiv preprint arXiv:2502.13189*, 2025.

Sijia Luo, Xiaokang Zhang, Yuxuan Hu, Bohan Zhang, Ke Wang, Jinbo Su, Mengshu Sun, Lei Liang, and Jing Zhang. Sparse-RL: Breaking the memory wall in LLM reinforcement learning via stable sparse rollouts. *arXiv preprint arXiv:2601.10079*, 2026.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing MoE reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Noam Shazeer. Fast Transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize from human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pp. 3008–3021, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12, 1999.

Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, et al. Zephyr: Direct distillation of LM alignment. *arXiv preprint arXiv:2310.16944*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W 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*, 2018.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. On the rollout–training mismatch in modern RL systems. In *OPT 2025: Optimization for Machine Learning Workshop at NeurIPS*, 2025.

Jingyang Yuan, Huazuo Gao, Damai Dai, 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.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big Bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Chujie Zheng, Shixuan Liu, Mingze Li, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2023.

Yang Zhou, Ranajoy Sadhukhan, Zhaofeng Sun, et al. Sparrow: Sparse rollout for stable and efficient long-context RL of large language models. *arXiv preprint arXiv:2606.08446*, 2026.

Rui Zhu, Weiheng Bai, Qiushi Wu, Yang Ren, Haixu Tang, and Yuchu Liu. How to compress KV cache in RL post-training? Shadow mask distillation for memory-efficient alignment. *arXiv preprint arXiv:2605.06850*, 2026. Withdrawn by the authors, v2, September 2026.

Appendix

The appendix is organised as follows. Section A gives the variance analysis of the full-trajectory importance-ratio estimator, with an explicit statement of what the result does and does not cover. Section B writes out the gradient of the transfer term and says what that derivation establishes about which pass teaches which, and what it does not. Section C lists every hyperparameter and evaluation protocol, including the per-task reward functions and the precision caveat behind the replay identity. Section D reports the systems measurements and states which quantities are and are not improved. Section E gives an analytical accounting of the logit memory the transfer term requires. Section F places each approach on the mismatch it acts on, and Section I collects the failure modes and the boundaries of the claims, including the statistical-reporting section that explains why every table here reports a collapse count beside the mean and s.d. of the surviving runs, rather than one mean over both modes. Sections K and L discuss learner-side sparse attention kernels and the boundary with quantization-based compression. Section M states why the multi-scale study is absent rather than reprinted, Section N derives the sequence-level token-saving accounting, Section O gives the complete training procedure, and Section P discusses broader impacts.

A Variance of the Full-Trajectory Importance-Ratio Estimator

This section formalises one specific statement: for the *full-trajectory-product* importance reweighting (IR) estimator, the second moment grows at least geometrically in the trajectory length L whenever the sparse rollout policy and the dense learner policy differ by a conditional variance bounded away from zero. We then state, as a separate and conditional claim, what exact mask replay does to the corresponding quantity in MaskLift. Section A.5 states the scope of both results; we ask the reader to read it together with the theorem, since the theorem is an explanatory result under its stated assumptions and not a statement about the training dynamics of GRPO or of MaskLift.

A.1 Setting and notation

Let $\tau = (s_1, a_1, \ldots, s_L, a_L)$ be a trajectory of length L generated by the compressed rollout policy π_{sparse}, so that $\tau \sim \pi_{\text{sparse}}$. Write $h_t = (s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t)$ for the generation history available before the t-th action, and let $\mathcal{F}_{t-1} = \sigma(h_t)$ be the corresponding filtration. Define the token-level importance ratio and its cumulative product

$$w_t \;=\; \frac{\pi_{\text{dense}}(a_t \mid s_t)}{\pi_{\text{sparse}}(a_t \mid s_t)}, \qquad \rho_{1:t} \;=\; \prod_{u=1}^{t} w_u, \qquad \rho_{1:0} \equiv 1.$$

The full-trajectory IR gradient estimator that we analyse is

$$\hat{g}_{\text{IR}} \;=\; \rho_{1:L}\, \nabla_\theta \log \pi_{\text{dense}}(\tau)\, A^{\pi_{\text{dense}}}(\tau), \qquad \tau \sim \pi_{\text{sparse}}. \tag{6}$$

All expectations below are taken over $\tau \sim \pi_{\text{sparse}}$ with θ held fixed, and all conditional statements are meant to hold π_{sparse}-almost surely.

Remark 2 (Well-posedness). We assume absolute continuity, $\operatorname{supp} \pi_{\text{dense}}(\cdot \mid s_t) \subseteq \operatorname{supp} \pi_{\text{sparse}}(\cdot \mid s_t)$ for every t, so that w_t is finite almost surely. For softmax policies over a fixed vocabulary both distributions have full support, so this holds; it is stated because it is what makes Lemma 1 an identity rather than an assumption.

Lemma 1 (The unit conditional mean is an identity, not an assumption)**.** *Under Remark 2, for every t,*

$$\mathbb{E}_{a_t\sim\pi_{sparse}(\cdot|s_t)}[w_t \mid h_t] \;=\; \sum_a \pi_{sparse}(a \mid s_t)\,\frac{\pi_{dense}(a \mid s_t)}{\pi_{sparse}(a \mid s_t)} \;=\; \sum_a \pi_{dense}(a \mid s_t) \;=\; 1.$$

Consequently the centred sequence $(w_t - 1)_{t\geq 1}$ *is a* martingale difference sequence *with respect to* $(\mathcal{F}_t)_{t\geq 0}$*, since* $\mathbb{E}[\, w_t - 1 \mid \mathcal{F}_{t-1}\,] = 0$*; and the cumulative product* $(\rho_{1:t})_{t\geq 0}$ *is a non-negative* martingale *with* $\mathbb{E}[\rho_{1:L}] = \rho_{1:0} = 1$.

Proof. The displayed computation is the change of measure; it requires only that the summand is well defined, which is Remark 2. For the second claim, $\mathbb{E}[\rho_{1:t} \mid \mathcal{F}_{t-1}] = \rho_{1:t-1}\,\mathbb{E}[w_t \mid \mathcal{F}_{t-1}] = \rho_{1:t-1}$, because $\rho_{1:t-1}$ is $\mathcal{F}_{t-1}$-measurable. Iterating gives $\mathbb{E}[\rho_{1:L}] = 1$. □

We record Lemma 1 explicitly because $\mathbb{E}[w_t \mid h_t] = 1$ was previously listed as an assumption, and because w_t itself is not a martingale difference sequence: an MDS requires conditional expectation 0, which holds for $w_t - 1$ and not for w_t.

A.2 Assumptions

Assumption 1 (Pointwise lower bound on the gradient–advantage magnitude)**.** *There is a constant* $C_{\min} > 0$ *such that*

$$\big\|\nabla_\theta \log \pi_{dense}(\tau)\, A^{\pi_{dense}}(\tau)\big\|^2 \;\geq\; C_{\min} \qquad \textit{for } \pi_{sparse}\textit{-almost every trajectory } \tau,$$

equivalently $\operatorname{ess\,inf}_{\tau\sim\pi_{sparse}} \|\nabla_\theta \log \pi_{dense}(\tau) A^{\pi_{dense}}(\tau)\|^2 \geq C_{\min} > 0$.

Assumption 1 is a *pointwise almost-sure lower* bound on a random variable whose law is induced by $\tau \sim \pi_{\text{sparse}}$ with θ fixed; the distribution is stated here explicitly because the bound is used inside an expectation over exactly that law. It is a lower bound, not a boundedness condition, and it is the single technical device that lets the proof avoid factorising a product of dependent random variables (Section A.3). It is also a genuine restriction: it fails at a stationary point, and more generally on any positive-probability set of trajectories where the gradient–advantage product vanishes. Group-relative advantage estimation makes such sets easy to realise in practice (if all K responses in a group receive equal reward, the normalised advantages are zero for that group), so Assumption 1 should be read as describing a regime away from convergence and away from reward-degenerate groups.

Assumption 2 (Non-degenerate information gap)**.** *There is a constant* $\sigma_0^2 > 0$ *such that, for every t,*

$$\operatorname{Var}_{a_t\sim\pi_{sparse}(\cdot|s_t)}[w_t \mid h_t] \;\geq\; \sigma_0^2 \qquad \pi_{sparse}\textit{-almost surely.}$$

Assumption 2 now assumes *only* a variance lower bound: the unit conditional mean is supplied by Lemma 1. Substantively it says that physically evicting part of the KV cache changes the next-token distribution by a non-vanishing amount at every step; if the evicted entries never mattered, σ_0^2 would be 0 and there would be no off-policy problem to solve.

Assumption 3 (Uniformly bounded estimator mean)**.** *There is a constant* $G < \infty$*, independent of L, with* $\|\mathbb{E}[\hat{g}_{IR}]\| \leq G$.

Assumption 3 is a regularity condition only. By Lemma 1 the estimator (6) is unbiased for the dense policy gradient, so it amounts to assuming that the dense policy gradient itself does not grow without bound in L; it is needed because a variance is a second moment minus a squared mean and we want the lower bound to survive the subtraction.

A.3 Lower bound

Theorem 1 (Geometric lower bound for the full-trajectory IR estimator)**.** *Let* $\hat{g}_{IR}$ *be the full-trajectory-product importance-ratio estimator* (6)*. Under Remark 2 and Assumptions 1, 2 and 3,*

$$\mathbb{E}\big[\rho_{1:L}^2\big] \;\geq\; (1+\sigma_0^2)^L, \qquad \mathbb{E}\big\|\hat{g}_{IR}\big\|^2 \;\geq\; C_{\min}(1+\sigma_0^2)^L,$$

and therefore

$$\operatorname{Var}\big(\hat{g}_{IR}\big) \;\geq\; C_{\min}(1+\sigma_0^2)^L - G^2 \;=\; \Omega\big((1+\sigma_0^2)^L\big).$$

Proof. **Step 1 (conditional second moment).** Fix t. By Lemma 1 the conditional mean of w_t is 1, so

$$\mathbb{E}\big[w_t^2 \mid \mathcal{F}_{t-1}\big] = \mathrm{Var}\left[w_t \mid \mathcal{F}_{t-1}\right] + \big(\mathbb{E}[w_t \mid \mathcal{F}_{t-1}]\big)^2 = 1 + \mathrm{Var}\left[w_t \mid \mathcal{F}_{t-1}\right] \;\geq\; 1 + \sigma_0^2$$

almost surely, using Assumption 2.

Step 2 (iterated conditioning and induction). Let $P_t = \prod_{u=1}^{t} w_u^2 = \rho_{1:t}^2$ with $P_0 = 1$. We show $\mathbb{E}[P_t] \geq (1+\sigma_0^2)^t$ for $t = 0, 1, \ldots, L$ by induction on t. The base case $t = 0$ is the identity $\mathbb{E}[P_0] = 1$. For the inductive step, assume $\mathbb{E}[P_{t-1}] \geq (1 + \sigma_0^2)^{t-1}$. The factor P_{t-1} is $\mathcal{F}_{t-1}$-measurable and non-negative, and $w_t^2 \geq 0$, so by the tower property and Step 1,

$$\begin{aligned}\mathbb{E}[P_t] = \mathbb{E}\big[P_{t-1} w_t^2\big] &= \mathbb{E}\Big[\mathbb{E}\big[P_{t-1} w_t^2 \mid \mathcal{F}_{t-1}\big]\Big] = \mathbb{E}\Big[P_{t-1}\,\mathbb{E}\big[w_t^2 \mid \mathcal{F}_{t-1}\big]\Big] \\ &\geq\; (1 + \sigma_0^2)\,\mathbb{E}[P_{t-1}] \;\geq\; (1 + \sigma_0^2)^t.\end{aligned}$$

Pulling P_{t-1} out of the inner conditional expectation is legitimate precisely because it is $\mathcal{F}_{t-1}$-measurable; the inequality then survives the outer expectation because $P_{t-1} \geq 0$. Taking $t = L$ gives $\mathbb{E}[\rho_{1:L}^2] \geq (1 + \sigma_0^2)^L$. Note that no independence between the w_t is used or needed: the argument only ever conditions on the past.

Step 3 (from ρ^2 to the estimator). Write $X(\tau) = \|\nabla_\theta \log \pi_{\text{dense}}(\tau) A^{\pi_{\text{dense}}}(\tau)\|^2$. Then $\|\hat{g}_{\text{IR}}\|^2 = \rho_{1:L}^2 X(\tau)$, and $\rho_{1:L}$ and $X(\tau)$ are both functions of the same trajectory τ, hence dependent in general. We therefore do *not* factorise the expectation. Instead, Assumption 1 gives $X(\tau) \geq C_{\min}$ pointwise almost surely, so $\rho_{1:L}^2 X(\tau) \geq C_{\min} \rho_{1:L}^2$ pointwise almost surely (both sides being non-negative), and monotonicity of the expectation yields

$$\mathbb{E}\big\|\hat{g}_{\text{IR}}\big\|^2 = \mathbb{E}\big[\rho_{1:L}^2\, X(\tau)\big] \;\geq\; C_{\min}\,\mathbb{E}\big[\rho_{1:L}^2\big] \;\geq\; C_{\min}(1 + \sigma_0^2)^L.$$

This is the step that the pointwise form of Assumption 1 exists to enable: a bound in expectation, $\mathbb{E}[X] \geq C_{\min}$, would not suffice, because it would require splitting $\mathbb{E}[\rho^2 X]$ into $\mathbb{E}[\rho^2]\mathbb{E}[X]$ and hence an independence assumption that is false here.

Step 4 (variance). For a vector-valued estimator, $\mathrm{Var}(\hat{g}_{\text{IR}}) = \mathbb{E}\|\hat{g}_{\text{IR}}\|^2 - \|\mathbb{E}[\hat{g}_{\text{IR}}]\|^2$. Combining Step 3 with Assumption 3 gives $\mathrm{Var}(\hat{g}_{\text{IR}}) \geq C_{\min}(1 + \sigma_0^2)^L - G^2$, and since G does not depend on L while $(1 + \sigma_0^2) > 1$, the right-hand side is $\Omega((1 + \sigma_0^2)^L)$. □

A.4 The corresponding claim for MaskLift, stated conditionally

Theorem 2 (Exact mask replay removes the sparse/dense ratio)**.** *Suppose the recorded retention mask is replayed exactly: for every generation step t, every attention head, and every query position, the set of* key *positions visible to the learner is identical to the set of key positions that were resident in the KV cache when a_t was sampled during rollout. Then the masked learner policy satisfies $\pi_\theta(a_t \mid s_t, M) = \pi_{sparse}(a_t \mid s_t)$ for all t π_{sparse}-almost surely, hence the sparse/dense importance ratio is identically one,*

$$w_t^{replay} = \frac{\pi_\theta(a_t \mid s_t, M)}{\pi_{sparse}(a_t \mid s_t)} \equiv 1, \qquad \rho_{1:L}^{replay} \equiv 1,$$

so the ratio's second moment is the constant $\mathbb{E}\big[(\rho_{1:L}^{replay})^2\big] = 1$ for every L, in place of the lower bound $\mathbb{E}\big[\rho_{1:L}^2\big] \geq (1 + \sigma_0^2)^L$ of Theorem 1, and $\mathrm{Var}\left(\rho_{1:L}^{replay}\right) = 0$.

Proof. The additive mask sets the pre-softmax logit of every non-retained key position to $-\infty$, so the post-softmax attention weight on those positions is exactly zero and the attention output at each query position is a function of exactly the retained keys and values. Under the stated exact replay this is the same set as at rollout, and the remaining computation (weights θ, positional encoding, and the sampled prefix) is identical, so the two conditional distributions coincide and the ratio is the constant 1. Substituting $\rho_{1:L} \equiv 1$ into (6) removes the ρ factor entirely. □

Three things must be said about Theorem 2. First, its conclusion is conditional on exact replay; Section J.4 states the implementation conditions under which the premise fails, in which case the conclusion does not hold. Second, it removes *one* variance contribution, not the variance of the training signal. The total gradient variance of MaskLift retains every on-policy source: stochastic

sampling of prompts and of the K responses per prompt, reward noise, the finite-sample error of the group-relative advantage estimate, and minibatch noise. We make no claim that MaskLift's total gradient variance is zero, small, or independent of L, and Theorem 2 is not a proof of training stability. Third, what the theorem makes constant is the ratio factor, not the estimator. With $\rho_{1:L}^{\text{replay}} \equiv 1$ the masked estimator is $\nabla_\theta \log \pi_\theta(\tau \mid M)\, A(\tau)$, whose second moment is $\mathbb{E}\big\|\nabla_\theta \log \pi_\theta(\tau \mid M)\, A(\tau)\big\|^2$, the masked counterpart of $X(\tau)$ in Step 3 of the proof of Theorem 1. Theorem 1 says nothing about how that quantity behaves in L, and we do not claim it is independent of L.

A.5 Scope of the analysis

We state the limits of the above explicitly.

1. **It concerns one estimator.** Theorem 1 lower-bounds the variance of the *full-trajectory-product* importance-ratio estimator (6). It says nothing about estimators that do not form the product $\prod_{t=1}^{L} w_t$.
2. **Practical PPO/GRPO does not use that estimator.** Practical implementations apply the PPO surrogate $\min\big(w_t\hat{A},\ \mathrm{clip}(w_t, 1-\epsilon, 1+\epsilon)\hat{A}\big)$ to one token at a time and therefore never form the product $\prod_{t=1}^{L} w_t$, which is why Theorem 1 does not apply to them. The $\min$ is not a hard bound on w_t: when $\hat{A} > 0$ it caps the term at $(1+\epsilon)\hat{A}$, but when $\hat{A} < 0$ and $w_t > 1+\epsilon$ the unclipped product $w_t\hat{A}$ is the smaller of the two and is therefore the one selected, so the per-token contribution is unbounded in w_t. Clipping buys bias in exchange for suppressing large ratios in one direction only; it does not bound the gradient variance, and we do not claim that it does. Sparse-RL is of exactly this kind: it clips a per-token staleness ratio. It applies its sparsity correction outside that clip, however, and rejects trajectories by a support test rather than by rank, so our clip-and-reject arm is not an implementation of it (Appendix C.1). The theorem should therefore be read as explaining *why* such bias-inducing devices are needed once the rollout and learner policies differ, not as a claim that a deployed baseline exhibits geometrically growing variance.
3. **The update we run is on-policy, not unbiased.** Under exact replay the data come from the policy being differentiated, which is what removes the sparse/dense ratio. That does not make the update an unbiased estimate of that policy's gradient. The group-relative advantage is standardised by the group's own reward standard deviation; the per-response loss is averaged over response tokens and then over trajectories; the gradient norm is clipped globally; and a prompt whose K samples all receive the same reward carries no gradient and is redrawn, up to a fixed number of draws, rather than stepped on, which conditions the prompt distribution on the reward. Each of these is standard practice and each biases the estimator. We make no unbiasedness claim for the estimator we actually run, and Theorem 2 makes none either: it is a statement about the ratio factor, not about the update.
4. **It is an explanatory result under its assumptions, not a dynamical statement.** Both Theorem 1 and Theorem 2 hold at a fixed parameter θ for a fixed pair of policies. Neither is a proof about the training dynamics of GRPO or of MaskLift: we do not analyse the optimisation trajectory, convergence, or the effect of the transfer term.
5. **Assumption 1 excludes stationary points** and, more generally, any positive-probability set of trajectories on which the gradient–advantage product vanishes; at such points $C_{\min} = 0$ and the bound is vacuous.
6. **The empirical results do not test the theorem.** The bound is stated in L; our trained contexts reach 4K tokens (Section I) and we do not measure gradient variance as a function of L. Nothing we report is a test of the bound's dependence on L.

B Why the Masked Pass Is the Teacher

Remark 1 assigns the teacher role to the masked pass and the student role to the dense one. This appendix writes out the gradient that assignment produces, and states plainly what the argument does and does not establish.

B.1 THE GRADIENT OF THE TRANSFER TERM

Fix a step and write $p_m(\theta) = \pi_\theta(\cdot \mid x, M_i)$ for the masked pass and $p_d(\theta) = \pi_\theta(\cdot \mid x)$ for the dense pass; both are functions of the *same* θ. With the masked pass detached,

$$\mathcal{L}_{\text{Transfer}}(\theta) = \mathcal{D}_{\text{KL}}\big(\text{sg}[p_m] \,\|\, p_d\big),$$

and, because the detached argument contributes no gradient and its entropy is constant in θ,

$$\nabla_\theta \mathcal{L}_{\text{Transfer}} = -\sum_{v \in \mathcal{V}} p_m(v)\, \nabla_\theta \log p_d(v). \tag{7}$$

This is a cross-entropy gradient whose differentiated object is the dense pass and whose target is the masked one. The policy gradient of Eq. (4), by contrast, is computed entirely under the replayed mask: the reward signal acts on p_m directly and reaches p_d only through the shared parameters. Both p_m and p_d are evaluated at the same θ, and the step applies one update after both have been computed, so within a step neither has yet been moved by this step's reward. The asymmetry is in the computation graph, not in time: the policy gradient enters through p_m, and Eq. (7) takes p_m as its target while differentiating p_d, the pass deployment will use. Across steps, the parameters that the policy gradient shaped through p_m are the parameters p_d is then fitted to.

The scope of this is the shape of one gradient, and it is the whole of what we establish. It does *not* follow that p_m is unaffected by Eq. (7): θ is shared, so any update to it moves both passes. We make no claim that the transfer term leaves the aligned objective's estimand untouched, and no claim to bound the size of the sparse-to-dense gap. Both are empirical questions, and Section 4.2 reports the measurement.

What the ablation separates. Table 6 compares $\lambda = 0$ with $\lambda = 10^{-2}$ at one fixed direction, $\mathcal{D}_{\text{KL}}(\text{sg}[p_m] \,\|\, p_d)$. The two arms differ by the presence of the term, not by its direction, so what the 18 paired seeds support is the joint effect of adding this loss. They do not rule out an account in which the term acts as a consistency regulariser between two views that share one parameter set, whose benefit would not depend on which view is detached. Separating the two needs the reverse direction, $\mathcal{D}_{\text{KL}}(\text{sg}[p_d] \,\|\, p_m)$, at the same seed count; we ran it at a single seed for each of two λ values, too few to report, and we therefore make no claim that the chosen direction beats its reverse.

B.2 READING THE COMPARISON IN TABLE 6

Three properties of that comparison constrain what may be concluded from it.

1. **The conditional means are conditional.** 70.7 and 80.0 describe the 15 and 16 runs that did not collapse, out of 18 each. The collapse counts are in the same table and are part of the result, not a footnote to it.
2. **The spread is not evidence of reliability.** The transfer term moves the conditional mean by $+9.3$ and takes the conditional standard deviation from 7.9 to 5.3, but those two numbers describe 15 runs and 16 runs, not the same set. Over all 18 runs the two arms' standard deviations are 20.0 and 20.5, because both are dominated by their collapses. We therefore claim a better typical outcome and make no claim that a single run of MaskLift is more reliable than a single run without the term.
3. **Collapse rates are not a claim.** 3 against 2 out of 18 is a difference of one run. We report it and interpret nothing from it.

C HYPERPARAMETERS AND EVALUATION PROTOCOL

Reward specification. The training reward for every dataset is that dataset's own automatic metric, computed against the reference answer on the *training* split: ROUGE-L for TL;DR and GovReport, exact-match accuracy for GSM8K, and token-level F1 for HotpotQA. No human preference data and no learned reward model are used anywhere in this work. Two consequences are worth stating plainly. First, the setting is online RL post-training with programmatic rewards; it is not RLHF in the sense of Stiennon et al. (2020), whose TL;DR setup trains a reward model from human comparisons.

The rollout-phase memory problem studied here is identical under either reward source, since it is a property of trajectory generation, but the task numbers should not be read as alignment quality. Second, on TL;DR and GovReport the training reward and the reported evaluation metric are the same functional form. They are evaluated on disjoint data: RL prompts are drawn from each dataset's training split and every reported score is computed on the corresponding held-out evaluation split, so no prompt that produced a reward gradient is scored again at evaluation time. This separation is what makes the summarization columns meaningful at all, and it is worth stating because reward and metric coincide there in functional form. Optimising ROUGE-L and then reporting ROUGE-L is a weaker claim than optimising a preference model and reporting ROUGE-L: the summarization columns measure how efficiently each method hill-climbs ROUGE-L under a rollout memory budget, and ROUGE-L is known to be exploitable through length and n-gram overlap. We make no claim that MaskLift produces better summaries under an independent judgement, and we did not run a human or model-judged evaluation.

Table 1 lists the objective, data and reporting settings, and Table 2 the decoding, optimisation and software settings a reimplementation also needs. All values were fixed once and reused across datasets; no per-dataset hyperparameter search was performed for any method, including the baseline (see the note on the clip-and-reject baseline below).

† K is set by the sequence length each task admits: exact mask replay forces the attention scores to be materialised, so cost grows with the square of sequence length. ‡ BF16 parameters are not usable here. At a learning rate of 2×10^{-6} an AdamW step on a BF16 weight falls below half an ULP for most weights, so the update is silently discarded and roughly 98% of parameters do not change; an entire early round of our own runs was voided by this before it was diagnosed. ¶ 30% is not the output of a sweep. It was chosen to place the study in the regime where the mismatch is severe (Appendix G); a milder budget was expected to make every arm look alike, as TL;DR and HotpotQA do. On GSM8K it does not: at 50% the uncorrected arm's all-run mean is further below MaskLift's than at 30% (Section 4.4). § GovReport's natural prompts exceed 8K tokens; truncating rather than filtering keeps the length distribution intact, since a length filter at this budget would retain only the shortest 15% of reports. # Configured as a rejection fraction of 0.2, but the implementation keeps $\max(1, \mathrm{round}(0.8K))$ of a full group of K scored trajectories, which drops 2 of 8 and 1 of 4: a realised 25% on all four datasets (Appendix C.1).

Two settings that are easy to miss. The first is the degenerate-group redraw in Table 2. A GRPO group whose K rewards are all equal gives every response the same advantage and therefore no gradient, so the loop draws another prompt rather than spending the step. This conditions the prompt distribution on the reward and is not a uniform pass over the pool; it is applied identically in every arm. The second is that exactly one optimiser step is taken per generated group. That is what makes θ_{old} the policy that sampled the data, so the policy-improvement ratio is 1 up to recompute noise at the moment of the update, and the clip in Eq. (4) is retained for generality rather than because it binds: in the replaying arms it essentially never fires, while in the arms that score sparse rollouts densely it fires on a substantial fraction of tokens (Section C.1).

Evaluation subsampling on the two large splits. TL;DR ships 6553 test examples and HotpotQA 7405; GSM8K and GovReport are small enough to score in full and are. On the two large splits we score a fixed random subsample of 2000, drawn once with seed 0 and therefore identical for every arm and every seed, so the subsample cannot add noise to the comparison the table is making. It is a random subsample rather than a prefix because a released file's order is not guaranteed to be independent of its content. The cost in resolution is small and the saving is not: at $n = 2000$ the standard error of a ROUGE-L mean is near 0.3 points, well below the seed-to-seed spread Table 5 reports, while scoring both splits in full for every arm would have spent roughly eleven GPU-hours buying precision the paper does not claim. As a check, the untrained TL;DR baseline scores 16.0 on the seeded 2000-example subsample the paper uses, against a running 16.27 over the first 5712 of the 6553 test examples, where a full-split evaluation was stopped once the two had clearly converged. We report the partial figure as partial rather than rounding it up to a full-split number we did not finish computing.

Exactly which metric. A metric named in a table is not a specification, so we state the implementations. **ROUGE-L** is the LCS-based F-measure at $\alpha = 0.5$ with the standard tokenizer (lowercase,

Table 1: Complete hyperparameter and protocol specification. Apart from the per-task generation and prompt budgets given here, values were fixed across datasets and no per-dataset sweep was performed.

Category	Setting	Value
Model / precision	Policy model	Qwen3-1.7B
	Numerical precision	BF16 compute, `float32` master weights[‡]
	Hardware, RL training	$1\times$ H200 NVL and $1\times$ H100 (shared)
	Hardware, systems study (Section D)	$1\times$ NVIDIA H200 NVL
RL objective	Algorithm	GRPO (no value network)
	Responses per prompt, K	8 (GSM8K, TL;DR); 4 (HotpotQA, GovReport)[†]
	GRPO ratio clipping, ϵ	0.2
	Learning rate	2×10^{-6}; the accuracy maximum of a grid run with the dense arm at 100 steps, seed 0, scored first on the test column and then again on validation (Appendix C)
	GRPO steps	200
MaskLift	Transfer coefficient, λ	10^{-2}; selected once on the GSM8K validation split and reused unchanged on the other three tasks (Appendix J)
	Transfer direction	$\mathcal{D}_{\mathrm{KL}}(\mathrm{sg}[\text{masked}] \,\|\, \text{dense})$; an empirical choice, not a design one: the opposite orientation was measured first and dropped (Appendix B)
	Reference-KL coefficient, β	0 (no reference model is constructed)
	KV eviction rule	position-proximity proxy, not SnapKV (Appendix G.1); SnapKV's own scorer in Appendix H
	KV retention ratio	30% retained[¶]
	Learner realisation of the mask	additive $-\infty$ mask (no physical slicing)
Clip + reject	Learner	dense forward, no mask replay
	Importance ratio	$\pi_{\text{dense}}/\pi_{\text{sparse}}$, clipped to $[0.8, 1.2]$
	Trajectory rejection	most deviated quarter of each group[#]
	KV eviction and retention	as MaskLift
Training data	TL;DR	short summarization, prompt budget 768 tokens
	GSM8K	mathematical reasoning, 1-shot, prompt budget 1200 tokens
	HotpotQA	multi-hop QA, prompt budget 3072 tokens
	GovReport	long-document summarization, truncated to 4096 tokens[§]
Evaluation protocol	Decoding	greedy, dense (no eviction), full context
	Split discipline	λ selected on validation; the retention budget and ϵ are design choices, not sweeps; the learning-rate grid was scored on the test column and rerun on validation, which returns the same rate (Appendix C)
	GSM8K	accuracy, test $n = 1319$, validation $n = 1000$
	TL;DR / GovReport	ROUGE-L (LCS F-measure, no stemming)
	HotpotQA	token-level F1 (SQuAD normalisation)
Runs	Seeds per configuration	3–5; transfer-term study 18
	Reported statistic	collapse count; mean and s.d. of non-collapsed
	Collapsed runs	reported, never trimmed (Section C.2)

Table 2: Decoding, optimisation and software settings. Every value here is the one the released code runs; where a value is a library or script default that no run overrides, it is marked as such. These are specifications rather than choices: none was swept, and none is claimed to be tuned.

Category	Setting	Value
Rollout decoding	Sampler	multinomial over the full vocabulary
	Temperature	1.0 (default; no run script overrides it)
	Top-p / top-k / repetition penalty	none applied
	Logged behaviour log-probabilities	$\log \mathrm{softmax}$ of the unscaled logits; at temperature 1 this is exactly the law each token was drawn from
	Stop rule	first EOS (three ids: the tokenizer's, 151,645, 151,643), otherwise the per-task generation cap
Generation budget	Response cap, per task	256 (GSM8K), 96 (TL;DR), 64 (HotpotQA), 512 (GovReport)
	Prompt budget, per task	1200, 768, 3072, 4096 tokens
	Over-budget prompts	skipped, except GovReport, where they are truncated
	Prompt pool	the first 2000 within-budget prompts of the training file, shuffled by seed
Optimisation	Prompts per step	1; its K responses are the GRPO group
	Optimiser steps per rollout group	exactly 1: no replay buffer and no second epoch, so θ_{old} is the sampling policy and no data is reused
	Optimiser	AdamW, $\beta = (0.9, 0.95)$, weight decay 0
	Learning-rate schedule	none: constant 2×10^{-6}, no warmup
	Gradient-norm clip	1.0
	Degenerate groups	a group whose K rewards have zero spread carries no advantage; up to 6 prompts are drawn per step to find one that does, and if none does, that step takes no optimiser step
Evaluation	Decoding	greedy (`do_sample=False`), dense, full context
	Batch size	16 sequences
	Response cap	320 tokens for the GSM8K runs of Tables 5 and 6; the per-task training cap elsewhere and for every untrained checkpoint (see the seam noted below)
Software	PyTorch / Transformers	2.10.0+cu128 / 5.1.0, identical on both machines
	Python	3.13.11 on machine A, 3.12.14 on machine B
	Attention implementation	`eager` in training, the only implementation that honours an arbitrary additive 4D mask bit-exactly

non-alphanumeric runs to whitespace) and *no* stemming; it is not ROUGE-Lsum, which sentence-splits first and generally scores higher on long summaries. **Token F1** for HotpotQA uses SQuAD normalisation: lowercase, strip punctuation, drop the articles *a*, *an*, *the*, then F1 over the token multiset. **GSM8K** accuracy is exact match on the final numeric answer, read from the first `\boxed{}` when present and otherwise the last number in the response, after truncating the generation at any point where it begins a new problem; a run-on few-shot generation that starts the next exercise would otherwise contribute that exercise's digits. The same truncation applies on the other three tasks with the marker set each one needs: a summary is cut at the first line beginning `SUBREDDIT:`, `TITLE:`, `POST:` or `Summarize the following`, and a HotpotQA answer is the first line of the block before the first `Read the following`, `Question:` or `[`. These are scoring rules, not decoding rules: generation itself stops only at EOS or at the token budget, and the identical rule is applied to the training reward and to the reported score.

We implement these rather than importing them, because our two training machines do not carry the same metric packages and a score that differed between machines would make their runs incomparable.

The implementation is checked to agree with `rouge_score`'s `rougeL` to within floating-point equality on every case we tested, and each task's reward function is checked to return its maximum on the reference answer itself. Both checks ship in the supplementary material and run without a GPU.

Which examples train, and which are scored. RL prompts are read in file order from each task's training file until 2000 have been collected that fit the task's prompt budget; that pool is permuted once by the run's seed and drawn from at most 1200 times, so within a run no prompt is used twice and no run sees the whole pool. Scores come from separate files: GSM8K's 1319-problem test split and its 1000-problem validation split, HotpotQA's released validation file, which is its only public held-out file, and the test files of TL;DR and GovReport.

GSM8K's validation file needs one sentence more, because the isolation there is by index rather than by file. We built it as the last 1000 examples of the released training file, and we did not delete them from that file; the isolation comes from the pool being the first 2000 examples in file order. At the 1-shot prompt budget no example is over budget, so the pool is exactly lines 0 to 1999 and the validation block is lines 6473 to 7472: disjoint, with more than four thousand examples between them, but disjoint because the pool is small rather than because the file was split. A reimplementation that raised the pool past line 6472 would train on its own validation set. No prompt that produced a reward gradient is scored again in any run we report. One asymmetry is worth stating rather than leaving to the code: at training a prompt longer than the budget is dropped from the pool on GSM8K, TL;DR and HotpotQA and truncated to the budget on GovReport, while at evaluation every over-budget prompt is truncated to the budget.

How the learning rate was chosen, and a split error in how it was chosen. The learning rate is the one hyperparameter of this paper whose calibration we have to report as a defect rather than as a protocol. It was picked by a 3-point grid, $\{2 \times 10^{-7}, 5 \times 10^{-7}, 2 \times 10^{-6}\}$, training the *dense* arm for 100 steps at seed 0 and 30% retention and keeping the largest rate that still improved on the untrained checkpoint. A fourth point, 1×10^{-6}, was launched and died before it was scored, and a fifth, 5×10^{-6}, had been rejected earlier on a different (larger) checkpoint. The scores were:

Learning rate	**Accuracy (%), 400 questions**
baseline (no RL)	42.75
2×10^{-7}	62.25
5×10^{-7}	64.50
2×10^{-6}	74.75

The defect is the split those 400 questions came from. The calibration script invoked the GSM8K evaluator without passing `--data`, and that argument's default is the *test* file, so the grid was scored on the first 400 questions of the split Table 5 reports. We found this while auditing the release and we are reporting it rather than quietly reconciling the text: earlier drafts of this paper said in five places that no hyperparameter had been selected on a reported split, and for the learning rate that was false.

Three things bound what it can have bought. The grid contains only the dense arm, so the criterion saw nothing about MaskLift; the selection ran 100 steps at seed 0 rather than the 200-step configuration every reported run uses; and the chosen value, 2×10^{-6}, is hard-coded identically into every arm of every reported run, so no arm is tuned relative to another. The exposure is that the reported column's first 400 questions influenced which of 3 rates all the arms share.

The same grid, rerun on the validation split. Rather than argue about how much that exposure is worth, we repeated the calibration. Everything is identical (same policy, same 100 steps, same dense arm, same seed, same 30% retention, same criterion) except that the evaluator is given `--data val.jsonl`, and the grid is extended above its old ceiling, because 2×10^{-6} was the largest rate the first grid contained and a criterion that picks the largest rate cannot distinguish the best point from the last one. On the validation split:

Learning rate	**Accuracy (%),** 400 **validation questions**
baseline (no RL)	47.75
2×10^{-7}	68.50
5×10^{-7}	70.25
1×10^{-6}	71.00
2×10^{-6}	81.50
5×10^{-6}	77.00

Validation accuracy peaks at 2×10^{-6} (81.50), which is the rate every reported run was trained at, and falls again at 5×10^{-6} (77.00). The value in use is therefore the one a clean selection returns, and no reported number depends on the test-split scoring.

Extending the grid also exposes a second defect, in the criterion rather than in the split. Read literally, "the largest rate at which the dense arm still improves over the untrained checkpoint" selects 5×10^{-6} here, because every rate in this grid improves over 47.75; it only ever returned 2×10^{-6} because 2×10^{-6} was the largest rate the original grid contained. A criterion that returns the ceiling of whatever grid it is given is not a criterion, and what the calibration actually did, on both splits, is take the accuracy maximum. We state it that way rather than reprinting the sentence the scripts were written with.

We report the sequence and not only its conclusion, because the order in which these things happened is part of what a reader is entitled to know, and because a confirmation after the fact is not the same thing as having done it right the first time.

One further seam is in the evaluators themselves. The GSM8K runs of Table 5 and all of Table 6 were scored by a GSM8K-specific script whose defaults are a 320-token generation cap and a 1400-token prompt cap; every other cell, including every untrained-checkpoint floor and, in the GSM8K column, both the clip-and-reject row and the published objective added in revision, was scored by the shared task evaluator at the per-task training caps (256 and 1200 on GSM8K). The two are not interchangeable: the same untrained checkpoint on the same 1000 validation problems scores 48.6 under the first and 48.3 under the second. The gap is small, but it means the clip-and-reject GSM8K row was scored under a shorter generation budget than the four arms beside it, and that the collapse floor of that column was measured under a different cap from the arms it thresholds. We report this rather than restate the column, because rescoring one arm under the other evaluator would give that arm a protocol no other arm in this paper has.

C.1 The clip-and-reject baseline is our simplification; the published objective, run separately

The row labelled *Clip + reject* in Table 5 is not an implementation of the algorithm published by Luo et al. (2026). It is a simplified trajectory-rejection baseline of our own, built in that spirit, and it differs from the published method in two places that matter. The first half of this subsection states both differences and withdraws any reading of that row as evidence about the published algorithm. The second half reports what happened when we implemented that algorithm as written and ran it, on all four datasets.

What our arm computes. The rollout is generated under the same eviction mask and the same retention budget as MaskLift. The learner scores each trajectory with a *dense* forward pass, so for token t of trajectory i it forms the single ratio $\rho_{i,t} = \pi_{\text{dense}}(o_{i,t})/\pi_{\text{sparse}}(o_{i,t})$ against the behaviour log-probabilities recorded during generation. The per-trajectory loss is the usual clipped surrogate $-\frac{1}{|o_i|}\sum_t \min\left(\rho_{i,t}\hat{A}_i,\ \text{clip}(\rho_{i,t}, 0.8, 1.2)\,\hat{A}_i\right)$. The trajectories of a group are then ranked by $\frac{1}{|o_i|}\sum_t |\rho_{i,t} - 1|$, the most deviated fraction is dropped, and the surviving per-trajectory losses are averaged.

What the published method computes. Sparse-RL factors the same discrepancy into two ratios: a staleness ratio $w_{i,t} = \pi_\theta/\pi_{\theta_{\text{old}}}$ between the learner and a dense old policy, and a sparsity-consistency ratio $\xi_{i,t} = \pi_{\theta_{\text{old}}}/\pi_{\text{sparse}}$. Only w enters the clip; ξ multiplies the clipped term from outside it. Rejection is a support test rather than a rank: a trajectory's mask M^{RS} is zero if any single token has $\xi_{i,t}$ below a small threshold (given as 10^{-4}) and one otherwise, and Luo et al. (2026) report

an average rejection rate of 0.07 under that rule, fluctuating between roughly 0.05 and 0.11, in the setting they measure. Their paper states no numeric clipping range.

The first difference: we clip the sparsity correction. Our runs take one optimiser step per generated group, so $\theta_{\text{old}} = \theta$ and $w \equiv 1$. Our single ρ is therefore numerically ξ, and we clip it to $[0.8, 1.2]$, whereas the published objective applies ξ unclipped. This is not a formality. Over the logged steps of this arm on all four datasets (every tenth GRPO step), the per-step fraction of response tokens with $|\rho - 1| > \epsilon$ has a per-task median between 9.5% (GovReport) and 18.2% (HotpotQA) and reaches 43% at individual steps. The ratio therefore leaves the clip range on a substantial share of tokens, and those are exactly the tokens on which the published correction factor departs furthest from one.

The second difference: we reject by rank, not by support. Our rule removes a fixed share of every group whether or not any trajectory is anomalous. The published rule removes only trajectories that leave the dense policy's support, and removes none from a consistent group. The two rules can select disjoint sets: a trajectory with one token at $\xi = 10^{-5}$ but a small mean deviation survives ours and is vetoed by theirs.

The realised rejection rate. We configured a rejection fraction of 0.2, but the implementation keeps $\max(1, \text{round}(0.8K))$ of a full group of K scored trajectories, which keeps 6 at $K = 8$ (GSM8K, TL;DR) and 3 at $K = 4$ (HotpotQA, GovReport). The realised rate is therefore 25% on all four datasets, not the 20% configured, and it is fixed by construction rather than measured: the per-step rejection count is computed inside the loss but was never written to the training logs, so we have no empirical rate to report. The 20% given by Zhu et al. (2026) was the configured value, not the realised one.

What the comparison supports. The values were fixed once and used unchanged on all datasets. We did not sweep the rejection fraction, the clipping range, or the retention budget for this arm, and we did not sweep the retention budget for MaskLift either; λ was selected on the GSM8K validation split and transferred unchanged (Section J). A reader should treat the row as a matched-configuration comparison against *our* simplification at fixed retention and fixed optimiser settings. It is not a comparison against the published algorithm. What that algorithm does in this regime is measured below, on all four datasets at one budget, and is a separate result that this row does not stand in for. The rejection step is unit-tested on purpose: a silently disabled rejection would turn this row into a naive learner with clipping, and nothing in its output would say so. The test checks *which* trajectories are dropped, not merely how many, at three rejection fractions; it and the metric test in the supplementary run both as scripts and under `pytest`.

The published objective, implemented as written. We implemented Eq. 7 of Luo et al. (2026) in the released code (`-method sparse_rl_true` in `masklift/losses.py`) and ran it on all four datasets at the same 30% retention budget, the same eviction rule, the same rollouts, the same optimiser settings, the same generation caps and the same seeds as every other arm. The implementation follows the published form in each place where our simplification departs from it. The surrogate is built from the staleness ratio $w_{i,t} = \pi_\theta / \pi_{\theta_{\text{old}}}$ and only w enters the clip. The sparsity-consistency ratio $\xi_{i,t} = \pi_{\theta_{\text{old}}} / \pi_{\text{sparse}}$ multiplies the clipped term from outside it and is never clipped; it carries no gradient, since both of its policies are fixed with respect to θ. Rejection is the support test and not a rank: a trajectory with any single token at $\xi_{i,t}$ below 10^{-4} is dropped whole, and a group all of whose trajectories are dropped contributes no update at all.

Why $w \equiv 1$ here, and what that limits. Our loop takes exactly one optimiser step per generated group, so $\theta_{\text{old}} = \theta$ at the moment of the update and w is identically one in value. It is still built rather than substituted, because it is the term the gradient flows through. The clip therefore never binds, and under our update schedule the surrogate reduces to $\xi_{i,t} \hat{A}_i$ with ξ unclipped. That is the published objective under this schedule and not a simplification of it, and it is why a surviving trajectory can take a large step when one token's dense and sparse probabilities disagree sharply. It is also a limit on what these runs test. A schedule taking several optimiser steps per group would separate θ from θ_{old} and would exercise the clip; ours cannot. What is measured here is the behaviour of the unclipped sparsity correction and of the support test, not of the staleness clip.

What the support test did, and how differently across the four tasks. Table 3 gives the counts, and the range across tasks is wider than anything else this arm does. On GSM8K the test vetoed 81.8% of trajectories, 3924 of 4800, and 76% of scored groups lost every trajectory and produced no optimiser step at all. The group's smallest ξ cleared the 10^{-4} floor in 0 of 600 steps: the median group minimum is 8.7×10^{-15} and even the largest we logged is 1.1×10^{-5}, still below the floor, so on that task the veto is not a marginal decision that a slightly lower threshold would reverse. At the other end, on GovReport the same rule vetoed 13.2%, and 0% of groups lost every trajectory: that arm took an update at every logged step. HotpotQA sits at 43.5% with 25% of groups fully vetoed, and TL;DR at 19.8% with 3%. Luo et al. (2026) report an average rejection rate of 0.07, fluctuating between roughly 0.05 and 0.11, in the setting they measure. All of our counts are over the steps written to the training log, every tenth step and the last, and are further restricted to groups with non-zero reward variance, which are the only groups scored at all; they describe a sample of each run rather than its total, and the sample size is the first column of the table.

What it scored. Table 4 puts all four tasks beside the arms they are read against. On GSM8K the published objective finished at 54.9 ± 5.9 over 3 seeds $(58.2, 48.1, 58.5)$, with 0 runs below the untrained checkpoint. That is above the floor 40.7 and well below dense GRPO's 72.3 and MaskLift's 72.8. On TL;DR it finished at 21.4 ± 3.5 $(17.4, 23.7, 23.1)$, inside dense GRPO's spread of 20.9 ± 4.0 and below MaskLift's 24.9 ± 0.8; on HotpotQA at 57.8 ± 2.6 against 61.5 and 60.6; on GovReport at 19.1 ± 0.2 against 19.6 and 19.3. MaskLift finishes above it on all four. It is also the only place in this paper where the published rule collapsed: 1 of its 3 GovReport runs finished below the untrained checkpoint, on the task where its veto rate is lowest, which is not the direction the veto explains.

Three protocol notes. It was scored by the shared task evaluator at the training caps, which is the evaluator that produced the untrained floors and the clip-and-reject column, so this arm and the threshold it is read against share a protocol; it is not the evaluator that produced the GSM8K dense and MaskLift columns, which is the seam Appendix C discloses. On GSM8K it does not carry the card confound that the clip-and-reject row carries there. But on TL;DR it carries a total one of its own, and in the opposite direction: every run of this arm is on machine A while every other TL;DR arm of Table 5 ran entirely on machine B (Appendix C.5), so the TL;DR comparison between this arm and the columns beside it is confounded with the card, and the cross-card measurement in that appendix bounds how much that can be worth.

How to read it. Where the veto is mild the rule trains: on TL;DR and GovReport it vetoed 19.8% and 13.2% of trajectories, took an update at almost every step, and finished inside or near dense GRPO's spread. In the severe corner it declined to train instead. One run does not fit that reading and we flag it rather than smooth it over: 1 of the 3 GovReport runs finished below the untrained checkpoint, on the task the rule vetoes least of the four, so the support test does not explain that collapse and we do not offer an explanation for it. The support test exists to remove trajectories the dense policy cannot score, and on GSM8K at this budget it finds that nearly every trajectory is one of those, so most groups produce no update. The rule's cost is denominated in discarded rollouts, which is what its authors describe; in the setting they measure that cost is 0.07 of the batch, on our benign task about a fifth of it, and in our severe corner nearly all of it. One comparison deserves stating plainly and it does not favour us: our own clip-and-reject simplification finished below the untrained checkpoint in 3 of 3 GSM8K runs, all-run mean 32.4, while the published rule did not collapse there at all. Of the two rejection devices, the published one is the better behaved in the corner that breaks learners, and our simplification is the worse. We cannot say what produced the distance between 54.9 and 40.7: the updates that survived the veto were taken with ξ unclipped, so their size is not bounded by the surrogate, and we recorded no gradient norms for this arm. What the four tasks together establish is a direction rather than a threshold: the veto rate is mild where the sparse and dense policies stay close and near-total where they do not, and the objective the test guards is then barely exercised. The two rejection devices are far from the dense arm in different directions. Clipping ξ and dropping a fixed quarter by rank keeps a biased estimator and trains on it, and on GSM8K those runs finished below the untrained checkpoint; leaving ξ unclipped and vetoing on support keeps the estimator faithful to the objective and then discards it. Removing the mismatch, rather than correcting or discarding an estimator built on it, is what trains and stays near the dense arm in both corners.

What sets the veto rate, and what does not. $\xi_{i,t}$ is a per-token quantity and the support test takes its minimum over the whole response, so a trajectory survives only if every one of its tokens clears the floor. The obvious consequence is that longer responses should be vetoed more often, since the minimum ranges over more tokens and one token is enough to zero the sequence. Four tasks at four different response caps let that be checked rather than assumed, and it is false. Ordered by the response cap each task generates under, which is 64, 96, 256 and 512 tokens in the released task definitions, the tasks run HotpotQA, TL;DR, GSM8K, GovReport; ordered by veto rate they run GSM8K, HotpotQA, TL;DR, GovReport. The task with the longest responses is vetoed least and the task with the shortest is vetoed second most. Response length does not order the veto rate.

Neither does the average size of the mismatch. Figure 2(b) plots the median deviation of the ratio from one for a learner that evicts and scores densely, which is a direct measure of how far the two policies sit apart on average; ordered by that quantity the tasks run TL;DR, HotpotQA, GSM8K, GovReport, which is again not the veto ordering. That is what should be expected of a rule that thresholds a *minimum*: a central statistic does not govern a tail one.

What does order it is the lower tail of the same distribution, which the table reports directly. The median group-minimum ξ runs 8.7×10^{-15}, 1.0×10^{-8}, 6.4×10^{-5}, 4.5×10^{-4} across those four tasks in exactly the veto ordering, and it spans about ten orders of magnitude between the extremes. The rule's cost is therefore set by how deep the per-token ξ distribution reaches below the floor on a given task, not by how many tokens it is minimised over, and a task can be severe for this rule while being mild for every other measure in this paper.

The comparison with the published rate says the same thing from outside our setup: Luo et al. (2026) run responses to a 4096-token maximum against a fixed 512-entry cache budget, sixteen times our GSM8K response length, and observe a rejection rate of 0.07 against our 81.8%. Their two constants are transcribed from their paper and are not measurements of ours. What we cannot do is predict the rate in advance: the tail statistic that governs it is measured after the rollouts exist, and nothing we report forecasts it from properties of the task.

One question this invites is whether the floor is simply set too high for our regime, so that tuning it would rescue the baseline on GSM8K. We did not sweep it, and the table bounds what a sweep could do. The recorded ξ is the *group's* minimum, so it says where the worst trajectory of a group sits rather than where all of them do. On GSM8K its largest value over 600 steps is 1.1×10^{-5}, an order of magnitude below the published floor, so at that floor there is no step at which the rule admits the whole group; and its median is 8.7×10^{-15}, so admitting every trajectory in half the steps would need a floor ten orders of magnitude lower, at which the test admits everything and is no longer a support test. What the rule can do between those two ends we did not measure, and this is a statement about the regime rather than about the rule.

Table 3: What the published support test did, per task, at 30% retention. Counts are over *every* GRPO step with a scored group, from the per-step trace of each of the 12 runs, pooled over three seeds per task; "steps" is that number. An earlier version of this table was computed from the every-tenth-step subsample the queue's log printed, which is not representative of a quantity this bursty: it put GSM8K's veto rate at 99.1% where the full trace gives 81.8%. The ordering the section below rests on is the same under both. "No update" counts steps at which every trajectory in the group was vetoed, which returns no loss and takes no optimiser step. "min ξ" is the smallest sparsity-consistency ratio in the group, and the support test vetoes a trajectory whose own minimum falls below 10^{-4}.

		trajectories vetoed		**no update**	**min ξ over the group**	
Task	steps	count	%	% of steps	median	largest
GSM8K	600	3924 of 4800	81.8	76	8.7×10^{-15}	1.1×10^{-5}
TL;DR	600	948 of 4800	19.8	3	6.4×10^{-5}	1.8×10^{-1}
HotpotQA	600	1043 of 2400	43.5	25	1.0×10^{-8}	9.9×10^{-1}
GovReport	600	317 of 2400	13.2	0	4.5×10^{-4}	1.7×10^{-1}

C.2 How results are summarised

Every task number in this paper is taken over independent training runs that differ only in seed. We do not report significance tests, and the reason is not that the differences are small.

Table 4: The published objective of Luo et al. (2026), implemented as written and run at 30% retention, beside the arms it is read against. Each task uses its own metric, so cells are comparable down a block and not across blocks. "Collapsed" counts runs finishing below the untrained checkpoint on that split; the mean and s.d. describe the runs that did not collapse and the last column describes all runs of the cell. Each cell is three seeds except the GSM8K dense and MaskLift rows, which are five, and the No RL row, which is one untrained checkpoint; Table 10 lists every run individually. The published objective, the clip-and-reject row and every floor were scored by the shared task evaluator at the training caps; the GSM8K dense and MaskLift rows were not (Appendix C). The published objective ran all its seeds on machine A. Two card confounds survive in this table and run in opposite directions: the GSM8K clip-and-reject row ran two of its three seeds on machine B, and the whole TL;DR block except the published objective ran on machine B (Appendix C.5).

		collapsed	**did not collapse**		**all runs**
Task	**Arm**	runs	mean	s.d.	mean
GSM8K	No RL (the floor)	–	40.7	–	40.7
	Dense GRPO (100% KV)	0	72.3	7.8	72.3
	Clip + reject (ours)	3	–	–	32.4
	Published objective	0	54.9	5.9	54.9
	MaskLift	1	72.8	2.7	61.6
TL;DR	No RL (the floor)	–	16.0	–	16.0
	Dense GRPO (100% KV)	0	20.9	4.0	20.9
	Clip + reject (ours)	0	19.7	3.3	19.7
	Published objective	0	21.4	3.5	21.4
	MaskLift	0	24.9	0.8	24.9
HotpotQA	No RL (the floor)	–	32.9	–	32.9
	Dense GRPO (100% KV)	0	61.5	0.7	61.5
	Clip + reject (ours)	0	58.9	1.9	58.9
	Published objective	0	57.8	2.6	57.8
	MaskLift	0	60.6	2.2	60.6
GovReport	No RL (the floor)	–	17.0	–	17.0
	Dense GRPO (100% KV)	0	19.6	0.8	19.6
	Clip + reject (ours)	0	18.4	0.1	18.4
	Published objective	1	19.1	0.2	18.4
	MaskLift	0	19.3	0.6	19.3

The outcome distribution is bimodal: a minority of runs ends below the untrained checkpoint. Earlier drafts of this paper described that as arm-independent, including the dense baseline; our data do not support it. Collapses are confined to the arms that train under eviction (12 of 66 such held-out runs), while the dense baseline, which never compresses, has not collapsed in any of its 14. Both replaying arms are among the affected, so replay does not prevent collapse; and with these counts, and no identified mechanism, we do not claim that compression causes it either. A mean taken over that shape describes neither mode, and a standard deviation over it is dominated by the collapsed runs to the point of inverting: in the transfer-term comparison the *better* arm has the larger raw s.d. while being visibly tighter everywhere except at the collapse. Rank statistics avoid that particular trap, and on the one comparison in this paper where the runs are genuinely paired they are informative rather than uninformative; an earlier version of this paragraph asserted the opposite, and the assertion was wrong on our own data. Section C.3 gives the two tests and the sample-size history they have to be read against. They are reported only there: for the $n = 3$ to 5 unpaired cells of Table 5 we continue to report counts and conditional moments, because with three runs and a bimodal outcome a p-value would carry less information than the runs.

We therefore report the two things the distribution actually has. **First, the collapse count:** how many runs finished below the untrained checkpoint. The threshold is fixed in advance, is the same for every arm, and is a property of the run rather than a cut chosen after seeing the outcomes. **Second, the mean, standard deviation and range of the runs that did not collapse.** Both parts are always reported together, so a conditional mean is never presented without the count of what was conditioned away.

Table 5: The numbers behind Figure 2. Task scores (%) on the held-out split (test, except HotpotQA, which ships none public; nothing was tuned on it): accuracy (GSM8K), ROUGE-L (TL;DR, GovReport), token F1 (HotpotQA). Each cell is the mean ± s.d. of the runs that did *not* collapse; "collapsed" means finishing below the untrained checkpoint on that task, a threshold fixed in advance and identical across arms. Where any run collapsed the count follows as (collapsed/total); "–" marks a cell with no surviving run, and a cell with fewer than three runs carries its count. Cells otherwise hold five seeds on GSM8K and three elsewhere. The outcome distribution is bimodal, so the two parts are reported separately rather than averaged together (Appendix C.2). HotpotQA ships no test file; its held-out split is the released validation split. The transfer coefficient λ was selected on the GSM8K validation split, which this table does not report; the learning rate's calibration was scored on the first 400 questions of the GSM8K column, which Appendix C reports in full. The clip-and-reject row is our simplified trajectory-rejection baseline, not the published Sparse-RL algorithm; the row below it is that algorithm implemented as written, and the two are separated in Appendix C.1. "–" is the measured outcome that every run of the cell finished below the untrained checkpoint, not a cell we did not run: every configuration in this table was run.

Method	**GSM8K**	**TL;DR**	**HotpotQA**	**GovReport**
No RL (base checkpoint)	40.7	16.0	32.9	17.0
Dense GRPO (100% KV)	72.3 ± 7.8	20.9 ± 4.0	61.5 ± 0.7	19.6 ± 0.8
Naive eviction (mismatched)	– (5/5 col.)	19.6 ± 1.2 (1/3 col.)	55.4 ± 3.0	19.2 ± 0.6
Clip + reject (Sparse-RL style)	– (3/3 col.)	19.7 ± 3.3	58.9 ± 1.9	18.4 ± 0.1
Published objective, as written	54.9 ± 5.9	21.4 ± 3.5	57.8 ± 2.6	19.1 ± 0.2 (1/3 col.)
Replay only ($\lambda = 0$)	67.9 ± 6.0 (1/5 col.)	23.9 ± 1.6	59.4 ± 1.4	18.8 ± 0.6
MaskLift	72.8 ± 2.7 (1/5 col.)	24.9 ± 0.8	60.6 ± 2.2	19.3 ± 0.6

Table 6: Effect of the sparse-to-dense transfer term, 18 paired seeds per arm, GSM8K validation; every seed is listed in Table 11 and drawn in Figure 3. "Collapsed" counts runs finishing below the untrained checkpoint (48.6). The conditional columns describe only the runs that did not collapse, and therefore describe 15 runs in one row and 16 in the other; the difference of those two means is not a paired effect. The all-run columns describe the same 18 runs in both rows, and the block below compares the arms seed by seed.

		collapsed	**runs that did not collapse**			**all n**
Arm	n	(of n)	mean	s.d.	range	mean (s.d.)
Replay only ($\lambda = 0$)	18	3	70.7	7.9	$[55.2, 84.2]$	63.4 (20.0)
MaskLift (replay + transfer)	18	2	80.0	5.3	$[62.0, 83.7]$	73.2 (20.5)
Paired by seed index: MaskLift minus replay only						
all 18 seeds		median $+11.5$, range $[-61.0, +81.2]$; higher on 14, lower on 4				
13 seeds, neither collapsed		median $+11.3$, range $[-4.9, +21.5]$; higher on 11, lower on 2				

Two consequences follow for how the tables should be read. We do not claim task-score differences smaller than the run-to-run spread the same arm shows; Section 4.1 states those limits, and Section C.4 makes the Dense/MaskLift comparison paired, by seed and by question, rather than by two marginal intervals. Every interval and spread in this paper is over the sampling of a fixed benchmark's items or over repeated runs, not over the population of tasks a deployed model would meet, and where two conditional means are close we say they are close: no equivalence margin was fixed in advance, so equivalence is not established. And we do not claim that any arm collapses less often than another: the counts differ by at most one run, which we report and decline to interpret. What causes a run to collapse is unresolved.

C.3 Two rank tests on the transfer-term comparison, and how n was chosen

The 18 seeds of Table 6 are paired by index: the two arms differ by the transfer term alone and share their seed, their data order and their protocol. That makes a paired rank statistic well defined, and it is the one place in this paper where the sample supports one. MaskLift is higher on 14 of 18 pairs and lower on 4. An exact two-sided sign test on those signs gives $p = 0.031$ and an exact Wilcoxon signed-rank test $p = 0.030$; on the 13 pairs where neither arm collapsed they give 0.022 and 0.002.

The sign test uses nothing but the sign of each pair, so it is untouched by the bimodality that makes a mean over these runs uninformative, and its sufficient statistic is the $14/4$ count Table 6 already prints.

One property of the pairing is worth stating because it is not what a reader might assume. A shared seed fixes the data order and the initialisation in both arms, but it does not make the two scores move together: across the 18 pairs their correlation is -0.03. So the pairing removes the variation a seed controls and essentially none of the rest, and Figure 3 draws the connectors precisely so that this is visible rather than assumed. That is an argument for the sign test over a paired mean, not against the design.

The sample size was not fixed in advance, and that qualifies these two numbers. The comparison was first run at 6 seeds per arm. At that size the transfer arm led on the median by about 11 points and an unpaired Mann–Whitney test returned $p = 0.15$, which we read as suggestive and not decided, and we extended both arms to 18 seeds under an identical protocol (only the seed changes) and stopped there. That is an internal-pilot sample-size increase rather than sampling until something crosses a threshold: the rule recorded at the time was "not significant at $n = 6$, so collect more and stop at $n = 18$", the stopping point was fixed when the extension was launched, and no interim result was ever reported as a conclusion. It is still a data-dependent choice of n made on the same outcome, so the two p-values above are not the p-values of a fixed-n design and we do not use them to reject anything. We report them because a reader is otherwise left to judge 14 of 18 by eye, and we report their provenance because a test whose n was chosen after an interim look is worth less than one whose n was not.

C.4 Paired per-item comparison, Dense against MaskLift on GSM8K

Section 4.1 compares arms by their conditional means. Because every arm is scored on the same 1319 GSM8K test problems in the same order, the same comparison can be made *paired*, by seed index and by question, and this section reports it. The pairing changes what the uncertainty statement is about. A Wilson interval on one arm's accuracy describes how far that one number would move if the benchmark were resampled; it is not a statement about a difference between two arms, and two arms scored on the same questions have correlated errors that a pair of marginal intervals ignores.

What was paired. Dense and MaskLift at 30% retention, seeds 0 to 4, GSM8K test. One MaskLift run (seed 3) finished below the untrained checkpoint and is a collapse by the convention of Section C.2; no Dense run did. Table 7 reports the four non-collapsed seed pairs and, separately, all five, because dropping a pair conditions on the outcome. The 72.8 against 72.3 comparison in Table 5 is four MaskLift survivors against five Dense runs and is therefore not seed-matched; over the matched pairs Dense averages 71.6.

What the pairing shows. Over the 4 non-collapsed pairs the two arms disagree on 23.6% of the 5276 item-comparisons: 11.2% are questions only Dense answers correctly and 12.4% only MaskLift. The pooled paired difference is $+1.21$ points in MaskLift's favour, with a normal-approximation 95% interval, over question sampling with the seed pairs held fixed, of $[-0.10, +2.52]$ points. That interval is narrow because pairing removes the question-difficulty component of the variation, not because the two arms behave alike on individual questions. Taken one pair at a time the same difference is $-11.07, -3.41, +14.71, +4.62$ points: a standard deviation of 11.05 points over 4 pairs, several times the width of the question-sampling interval and wider than the ± 2.4 point Wilson half-width a single run carries. Including the collapsed pair moves the pooled difference to -10.75 points. We report intervals on differences and no hypothesis tests, for the reasons in Section C.2.

What this does and does not establish. On this fixed benchmark, with these five seeds, the two arms' conditional means are close and the seed-matched difference is small next to its own spread across seeds. That is not equivalence. We fixed no equivalence margin in advance and we run no equivalence analysis, so the statement the evidence supports is that the conditional means are close and equivalence is not established. The intervals above are over the sampling of GSM8K's test questions and, separately, over our seeds. Neither is an interval over the population of problems a deployed model would meet, and nothing here licenses a claim that the two arms would stay close on a different benchmark or a different task.

Table 7: Paired per-item comparison of Dense and MaskLift on the GSM8K test split, $n = 1319$ questions, identical and identically ordered for every run. "Both" through "neither" count questions by the pair of outcomes. The paired difference is (only MaskLift) minus (only Dense), divided by the number of item-comparisons in the row. Every interval treats the question as the sampling unit, with the seeds held fixed; in the pooled rows the per-question difference is averaged over seed pairs first. Seed 3's MaskLift run is a collapse (Section C.2), so the comparison is pooled twice, with and without it. No hypothesis test is reported.

	Score (%)		Questions				
Seed	Dense	MaskLift	both	Dense only	MaskLift only	neither	Paired difference (points), 95% interval
0	81.0	69.9	851	217	71	180	-11.07 $[-13.59, -8.55]$
1	74.7	71.3	812	173	128	206	-3.41 $[-5.99, -0.83]$
2	60.0	74.8	700	92	286	241	$+14.71$ $[+11.82, +17.60]$
4	70.7	75.4	825	108	169	217	$+4.62$ $[+2.15, +7.10]$
Pooled, 4 pairs	71.6	72.8	3188	590	654	844	$+1.21$ $[-0.10, +2.52]$
3 (collapsed)	75.1	16.5	195	796	23	305	-58.61 $[-62.86, -54.35]$
Pooled, all 5	72.3	61.6	3383	1386	677	1149	-10.75 $[-12.10, -9.40]$

C.5 What a seed does and does not fix

Training is bit-reproducible on a fixed machine. A 200-step run logs its group reward every 10 steps and again at the last step, so each trace carries 21 logged rewards. The GSM8K dense arm was trained once for the original table and again on the same card, ten days and one code change later, and the two reproduced every logged reward in the training trace: 63 of the 63 steps the two runs of each of 3 seeds logged in common, with no exception. Their evaluation scores did not match, and that is the point. The retrained runs score $79.5, 73.5, 60.9$ where the originals score $81.0, 74.7, 60.0$ on the same problems, differences of about a point in both directions. Training is therefore bit-identical across ten days and a code change on this card, and the whole of the difference between the two scorings belongs to the evaluator that produced them (Appendix C), not to drift in training. The seed is doing real work: over 354 pairs of GSM8K runs differing only in seed, the two traces agree at 0–8 of those 21 steps, median 3, which is about what coincidence gives when the group reward takes nine discrete values. It is not reproducible across machines. Our runs were executed on two cards (Appendix C), and two early configurations that were run with the same seed on both landed at 22.3 against 34.5 and at 79.8 against 83.4, their two runs disagreeing on 717 and on 217 of the 1000 validation problems. The first gap is larger than several of the differences this paper does discuss.

Re-running HotpotQA turned that anecdote into a measurement. Its four trained arms were originally executed on machine B while the clip-and-reject arm of the same column ran on machine A, which confounded card with arm in exactly the row whose collapse count Section 4.1 discusses. All four were re-run on machine A, and the column the paper reports is that re-run. The originals were kept rather than deleted: the same configuration executed on two cards is the cleanest estimate of the cross-card effect this paper can make. Over 13 such pairs the two cards disagree by a median of 1.4 points and by as much as 4.4 (Table 8). The smallest disagreement, 0.1, is the untrained checkpoint: identical weights, no training at all, and the two cards still do not produce the same score. At the arm level the shift reaches 3.2 points, on *naiveeviction*, and it is not a constant offset that cancels in a comparison: the gap between MaskLift and the uncorrected arm on this task is 1.8 points read off machine B and 5.2 read off machine A. This is the concrete reason the main text reads only large separations as results. The seed therefore fixes the data order and the initial state, not the arithmetic, and the run-to-run spread we report contains hardware variation as well as sampling variation. We label the two cards A and B and use those labels throughout; Table 10 gives the card for every run. We did not balance them across arms: of the 18 seeds per arm in Table 6, the no-transfer arm ran 14 on A and 4 on B, and MaskLift 13 and 5. The imbalance is small and its direction is not consistent (B produced the worse mean for one arm and the better mean for the other), but it is uncontrolled, and three seeds (2, 4 and 16) had their two runs on different cards, two of them the largest positive paired differences in the table, so a reader should treat the comparison as paired by seed index rather than by execution environment. In Table 5 the four original GSM8K arms other than clip + reject are balanced exactly, each running seeds 0–3 on A and seed 4 on B, while the published objective added in revision ran its three seeds on A. GovReport ran entirely on one card; TL;DR did not, for the reason given below. The clip-and-reject cells are not balanced against the arms beside them on GSM8K: of its three runs there two are on B and one on A, while every other arm in that column

ran four of its five seeds on A. Card is therefore confounded with arm in that row, and it is the row whose collapse count Section 4.1 discusses. The same confound existed on HotpotQA in the opposite direction and has been removed by re-running that column onto one card, as described above.

One confound was created rather than removed. The arms added in revision, the published objective and the two arms under SnapKV's scorer, were all run on machine A. On GSM8K, HotpotQA and GovReport that matches the arms they are compared with. On TL;DR it does not: every other TL;DR arm of Table 5 ran entirely on machine B, so every TL;DR comparison between an arm added in revision and an arm of the original table is confounded with the card. The measurement above is what bounds it, and it is not small enough to ignore on a task whose arms span about five points.

Table 8: The same configurations executed on both cards. HotpotQA's four trained arms were run on machine B, then re-run on machine A so that the whole column would sit on one card; the paper reports the machine A column and these are the two sets read against each other. Scores are on the held-out split, 13 seed-matched pairs in total including the untrained checkpoint. The seed fixes the data order and the initial state, not the arithmetic.

Arm	seeds	machine B	machine A	shift
Dense GRPO	3	60.7	61.5	+0.8
Naive eviction	3	58.6	55.4	-3.2
Replay only	3	60.5	59.4	-1.1
MaskLift	3	60.4	60.6	+0.2

The seeds are also not independent across tables. The transfer-term comparison and the replay-only column of Table 5 are the same configuration, so four of the five seeds behind that column are among the 18; the columns differ in split, the table reporting the held-out test set and the comparison the validation set on which λ was chosen.

Table 9: Which card ran which run. Machine A is the 141 GB H200 NVL and machine B the 80 GB H100; both are shared. Top block: counts per cell of Table 5, written nA/nB; the assignment was made by a queue that admitted a job to whichever card had free memory, not by design. Middle block: the two ablations of Section 4.4, which an earlier version of this table left out because it was scoped to the main budget. Bottom block: the seed indices of Table 6, machine A then machine B. Read from the training logs, so it records which machine executed each run rather than which directory holds its result. It is reported so that the hardware confound described above can be located rather than only counted. By the strict test, an arm whose runs share no card with any arm it is compared with, 1 cell qualifies (TL;DR/Published objective). Two further comparisons are card-*asymmetric* rather than disjoint and are read in the text as though they were not: every 50% MaskLift run is on machine B while the 30% arm it is compared with is mostly machine A, and both $\lambda = 0.1$ runs are on machine B while $\lambda = 10^{-2}$ is mostly machine A.

	Dense	Naive	Clip + rej.	Replay only	MaskLift	Pub. obj.
GSM8K	4A/1B	4A/1B	1A/2B	4A/1B	4A/1B	3A
TL;DR	3B	3B	3B	3B	3B	3A
HotpotQA	3A	3A	3A	3A	3A	3A
GovReport	3A	3A	3A	3A	3A	3A
Section 4.4, retention ablation						
GSM8K, 50%	–	1A/2B	–	–	3B	–
Section 4.4, λ sweep: 0: 14A/4B, 10^{-3}: 2A/1B, 10^{-2}: 13A/5B, 10^{-1}: 2B						
Table 6, seed indices: machine A, then machine B						
Replay only	0–5, 9–15, 17			6–8, 16		
MaskLift	0, 1, 3, 5, 9–17			2, 4, 6–8		

Two runs, clip-and-reject seed 2 on GSM8K and naive eviction at 50% retention seed 2, have a result file on both machines, because a finished result is copied to the other machine so that its queue skips the job rather than repeating it. The table attributes each to the machine whose training log holds the completed run, which is the machine that executed it; the copy on the other machine is a file, not a run.

C.6 EVERY RUN'S FINAL SCORE

Table 10 gives the final held-out score of every training run behind Table 5, and Table 11 every run behind Table 6. Nothing is trimmed. Each row lists the runs in seed order, the card each ran on (Appendix C.5), the collapse count, the mean and s.d. of the runs that did not collapse, and the mean and s.d. over *all* runs including the collapsed ones.

The all-run column is reported because a conditional mean invites the reading that it is the arm's expected outcome, and on GSM8K it is not: MaskLift's conditional 72.8 sits against an all-run 61.6, and the whole of that difference is one run.

Two comparisons this table does not support. The conditional s.d. within a column of Table 5 are computed on different numbers of runs (MaskLift on four GSM8K runs, dense on five, because dense did not collapse), so MaskLift's ± 2.7 against dense's ± 7.8 is not evidence that MaskLift is less seed-sensitive than dense; over all five runs per arm those s.d. are 25.3 and 7.8, in the opposite direction. And a collapse is a threshold crossing, not a uniform event: TL;DR's one naive collapse finished at 15.96 against a 15.98 floor, a margin of 0.02 points, while GSM8K's MaskLift collapse finished at 16.53 against a floor of 40.71. The rule is fixed in advance and applied identically, which is what makes the counts comparable across arms; it does not make the two failures the same size.

Table 10: Final score of every run behind Table 5, plus the 50% retention runs of Section 4.4, in seed order. "Card" gives the machine each seed ran on, in the same order, with A and B as in Appendix C.5. "Col." is the collapse count. The last two columns are the mean and s.d. of the non-collapsed runs and of all runs.

Arm	Final scores, by seed	Card	Col.	Non-collapsed	All runs
GSM8K, test (n=1319), 30% retention; floor 40.71					
Dense GRPO	80.97, 74.68, 60.05, 75.13, 70.74	AAAAB	0/5	72.3 ± 7.8	72.3 ± 7.8
Naive eviction	31.24, 39.50, 36.16, 40.03, 37.00	AAAAB	5/5	–	36.8 ± 3.5
Clip + reject	32.30, 33.74, 31.08	BBA	3/3	–	32.4 ± 1.3
Replay only (λ=0)	62.02, 71.49, 63.68, 74.30, 20.47	AAAAB	1/5	67.9 ± 6.0	58.4 ± 21.8
MaskLift	69.90, 71.27, 74.75, 16.53, 75.36	AAAAB	1/5	72.8 ± 2.7	61.6 ± 25.3
Published objective	58.15, 48.14, 58.45	AAA	0/3	54.9 ± 5.9	54.9 ± 5.9
TL;DR, test subsample (n=2000), 30% retention; floor 15.98					
Dense GRPO	25.50, 18.39, 18.94	BBB	0/3	20.9 ± 4.0	20.9 ± 4.0
Naive eviction	20.52, 15.96, 18.76	BBB	1/3	19.6 ± 1.2	18.4 ± 2.3
Clip + reject	16.14, 20.43, 22.52	BBB	0/3	19.7 ± 3.3	19.7 ± 3.3
Replay only (λ=0)	23.93, 22.40, 25.52	BBB	0/3	23.9 ± 1.6	23.9 ± 1.6
MaskLift	25.53, 25.19, 23.94	BBB	0/3	24.9 ± 0.8	24.9 ± 0.8
Published objective	17.37, 23.72, 23.10	AAA	0/3	21.4 ± 3.5	21.4 ± 3.5
HotpotQA, released validation subsample (n=2000), 30% retention; floor 32.93					
Dense GRPO	61.84, 60.66, 61.98	AAA	0/3	61.5 ± 0.7	61.5 ± 0.7
Naive eviction	57.71, 52.00, 56.52	AAA	0/3	55.4 ± 3.0	55.4 ± 3.0
Clip + reject	56.87, 59.00, 60.74	AAA	0/3	58.9 ± 1.9	58.9 ± 1.9
Replay only (λ=0)	57.93, 59.40, 60.76	AAA	0/3	59.4 ± 1.4	59.4 ± 1.4
MaskLift	60.76, 58.32, 62.65	AAA	0/3	60.6 ± 2.2	60.6 ± 2.2
Published objective	56.60, 56.06, 60.81	AAA	0/3	57.8 ± 2.6	57.8 ± 2.6
GovReport, test (n=973), 30% retention; floor 16.99					
Dense GRPO	19.27, 20.50, 18.92	AAA	0/3	19.6 ± 0.8	19.6 ± 0.8
Naive eviction	19.08, 19.81, 18.57	AAA	0/3	19.2 ± 0.6	19.2 ± 0.6
Clip + reject	18.31, 18.44, 18.58	AAA	0/3	18.4 ± 0.1	18.4 ± 0.1
Replay only (λ=0)	18.63, 18.40, 19.47	AAA	0/3	18.8 ± 0.6	18.8 ± 0.6
MaskLift	19.50, 19.75, 18.62	AAA	0/3	19.3 ± 0.6	19.3 ± 0.6
Published objective	16.96, 18.98, 19.23	AAA	1/3	19.1 ± 0.2	18.4 ± 1.2
GSM8K, test, 50% retention; floor 40.71					
Naive eviction	3.03, 55.50, 1.52	BBA	2/3	55.5	20.0 ± 30.7
MaskLift	66.41, 77.79, 72.25	BBB	0/3	72.2 ± 5.7	72.2 ± 5.7

Table 11: Every run behind Table 6: GSM8K validation (n=1000), floor 48.6. The two arms differ by the transfer term alone and are paired by seed index. * marks a seed whose two runs were executed on different cards, so that pair is matched by seed but not by hardware.

Seed	Replay only	Card	MaskLift	Card	Difference
0	67.0	A	78.6	A	+11.6
1	75.4	A	74.9	A	−0.5
2*	70.4	A	83.4	B	+13.0
3	78.8	A	17.8	A	−61.0
4*	34.5	A	80.8	B	+46.3
5	60.6	A	82.1	A	+21.5
6	70.8	B	83.7	B	+12.9
7	71.9	B	82.7	B	+10.8
8	43.6	B	62.0	B	+18.4
9	72.5	A	79.6	A	+7.1
10	60.6	A	81.7	A	+21.1
11	84.2	A	79.3	A	−4.9
12	80.1	A	82.6	A	+2.5
13	70.0	A	81.3	A	+11.3
14	76.8	A	83.0	A	+6.2
15	55.2	A	19.3	A	−35.9
16*	1.3	B	82.5	A	+81.2
17	66.9	A	82.1	A	+15.2

D SYSTEMS MEASUREMENTS

Every measurement in this section was taken on Qwen3-1.7B, the policy every training experiment in this paper uses, with the released implementation on an otherwise empty card. Every column of Table 12 is measured, on runs that do what a rollout does: prefill the full context densely, drop the evicted key positions from the cache, then decode. An earlier version of this section reported a benchmark that instead decoded from a context already cut to the budget and timed prefill and decode together. That benchmark credited eviction with a halved prefill, which eviction does not produce, and against the corrected measurement its throughput figures are of the opposite sign in three of the five configurations; they are withdrawn and are not reprinted here. Zhu et al. (2026) reported systems numbers collected with a codebase since established not to execute the method it described; those are not reprinted, and nothing here is carried over from them.

What eviction removes, analytically. The decode-phase KV cache holds, for each of L layers, a key and a value tensor of H_{kv} heads $\times$ d_{head} per token, so for a context of C tokens at batch B in `bfloat16` its size is

$$\text{KV bytes} \;=\; 2 \cdot L \cdot H_{kv} \cdot d_{\text{head}} \cdot C \cdot B \cdot 2. \tag{8}$$

For Qwen3-1.7B, the policy every training experiment in this paper uses ($L = 28$, $H_{kv} = 8$, $d_{\text{head}} = 128$), this gives the figures below. Retention r scales the post-eviction cache by r exactly, because eviction removes whole key positions. These numbers are *arithmetic*: they are what the reader can recompute from the model's published configuration, and at the five shapes we ran they agree with the measured cache of Table 12 to the digits printed, so the formula is checked rather than assumed. They are the reason the rollout phase is the one worth compressing: at a 131,072-token context and batch 8 the cache alone is 112 GiB dense, which no single card holds, against 33.6 GiB at 30% retention.

Two things this table does not say. It is the cache only: parameters, optimiser state, activations and the vocabulary-sized logits of both learner passes are elsewhere, and on our runs those dominate at the context lengths we actually train. And it is the *decode-phase* cache, after eviction; prefill builds the full cache first, so the prefill peak is unchanged by any retention ratio.

The capacity ceiling, measured, and why eviction does not move it. Zhu et al. (2026) reported a largest context and a largest batch that ran before an allocation failure, from the withdrawn codebase. We re-ran that measurement against the released code: for each configuration, increase the batch size until the allocator fails, for a dense rollout and for one that evicts the cache after prefill, on an idle card of 140 GiB.

Table 12: Rollout-phase measurements on Qwen3-1.7B, the policy every training experiment in this paper uses, on a single H200 NVL in BF16, 128 decoded tokens. Each row is a full episode run n times: dense prefill, eviction from the full cache, then decode. The KV columns are the cache tensors' real size after eviction, summed over layers, and they reproduce (8) to the digits shown. "decode peak" resets the memory counter after prefill; "episode peak" is the maximum over prefill and decode together, which is what the card must hold. Both are peaks over the model's weights, the cache and the activations of the phase; the prefill samples one position's logits (`logits_to_keep`), so neither column carries a full-vocabulary projection over the context, which is a property of a harness rather than of a cache. Throughput is the change in decode tokens per second from the dense arm to the evicted one; prefill is excluded, because both arms pay the same prefill. The two arms are run alternately and paired by repetition index, so the interval is a 95% paired bootstrap over pairs rather than a spread of one arm about the other's mean. Every repetition ran on an otherwise empty card: `systems2.py` records whether another process held memory while it timed, and 0 runs were discarded on that test rather than averaged in. Both corrections postdate a first round of these measurements, which prefilled without `logits_to_keep` and ran all dense repetitions before all evicted ones; its files are kept in the archive under `superseded_systems2/`.

			KV cache (GiB)		decode peak		episode peak		decode thr.		
context	**batch**	**retention**	dense	evicted	dense	evicted	dense	evicted	change	95% CI	n
4K	1	30%	0.44	0.13	4.28	3.96	4.46	4.46	-1.2	$[-4.2, +0.5]$	11
16K	1	30%	1.75	0.52	5.61	4.36	6.39	6.39	$+0.9$	$[+0.2, +1.7]$	11
16K	1	50%	1.75	0.88	5.61	4.72	6.39	6.39	-0.6	$[-3.1, +1.0]$	11
16K	8	50%	14.00	7.00	18.18	11.06	24.32	24.32	$+21.0$	$[+20.8, +21.3]$	11
64K	1	50%	7.00	3.50	10.96	7.39	14.10	14.10	-0.1	$[-0.6, +0.4]$	11

Table 13: Decode-phase KV cache from (8) (1 GiB $= 2^{30}$ bytes). Arithmetic from the model configuration, not measured.

context	**batch**	**dense**	30% **retention**	50%
4,096	1	0.44 GiB	0.13 GiB	0.22 GiB
16,384	1	1.75 GiB	0.52 GiB	0.88 GiB
16,384	8	14.00 GiB	4.20 GiB	7.00 GiB
65,536	8	56.00 GiB	16.80 GiB	28.00 GiB
131,072	8	112.00 GiB	33.60 GiB	56.00 GiB

The result is negative and it is the same negative result the throughput measurement gives. In 3 of 3 configurations the two arms reach the *identical* largest batch and fail at the identical next one (Table 14), even though the evicted arm's decode peak is smaller by a factor of up to 3.0. Eviction cannot raise the ceiling because the ceiling is not set where eviction acts: both arms prefill the same context at full length before either evicts, so the largest allocation of the episode is one they share. What eviction buys is decode-phase working set, which is what Section 4.3 reports and what Figure 4 draws; a reader looking for more sequences per card will not find them here.

One measurement artefact is worth recording, not because it changed the answer but because it would have made the answer right for the wrong reason. Our first attempt returned logits for every prefill position rather than the last, which allocates a tensor proportional to batch times context times vocabulary; that tensor, not the KV cache, then set the ceiling, and the two arms failed at a batch where the cache was irrelevant. The numbers above come from a prefill that keeps one position's logits, which is what `capacity.py` does in the released code. The same artefact was present, unnoticed, in the episode-peak column of Table 12 until this revision: that measurement has been repeated with the same correction, its superseded files are kept beside these, and the episode peaks it reports are 18% to 56% lower than the ones an earlier draft printed, without changing the conclusion, which was and is that the two arms share them exactly.

Protocol for the step-time comparison. The only quantity we profile inside the RL loop is the step-time cost of the transfer term, and it is measured by running two arms back to back on an otherwise empty card: replay-only, which takes one learner forward pass, and MaskLift with transfer coefficient 10^{-2} in the `masked2dense` direction, which takes two. Each arm runs 30 GRPO steps on GSM8K at the settings of Table 1, repeated 2 times (`_steptime.sh` in the supplementary code). The per-step figure for a repetition is taken from that run's own elapsed-time checkpoints over steps 10 to

Table 14: Largest batch that completed before an allocation failure, Qwen3-1.7B on an idle 140 GiB card, prefill then 16 decoded tokens. *dense* prefills and decodes at full cache; *evicted* drops the cache to the retention budget after prefill, as a rollout does. Peak is the *decode-phase* peak: the released code resets the counter after prefill and after eviction, so it is the working set the decode loop holds and not the episode's largest allocation. The two arms reach the same ceiling in every configuration even so, because the episode's largest allocation is the prefill they share.

		dense		evicted	
context	**retention**	largest batch	peak (GiB)	largest batch	peak (GiB)
16K	30%	48	89.4	48	29.6
16K	50%	48	89.4	48	46.7
64K	50%	12	89.4	12	46.6

29, so the opening steps, which pay for compilation and allocator growth, are excluded. The two arms diverge after the first update and skipped different numbers of degenerate groups over that window, so this is not a comparison of identical rollout work. Nothing else in the loop is instrumented: this revision records no rollout throughput, no separate timing of the dense pass and no learner memory peak, and reports none.

What these measurements do and do not establish. We state this explicitly because Zhu et al. (2026) overclaimed here and was withdrawn for it.

1. **Claimed: the decode KV cache scales with the retention budget.** At a 16K context and batch 1, halving the budget halves the measured post-eviction cache, 1.75 to 0.88 GiB, and the measurement matches (8) in all 5 configurations. The same statement at the budgets, contexts and batch sizes we did not run is arithmetic only (Table 13).
2. **Claimed: a measured decode peak, and a small measured decode speed-up.** Over 11 repetitions per row on an empty card, the decode-phase peak falls from 5.61 to 4.72 GiB at 16K, batch 1, 50%. Decode throughput moves by between -1.2% and $+21.0\%$ across the five configurations. 2 of the 5 have an interval that excludes zero, and only 1 of those is a gain of any size: the batch-8 row, $+21.0\%$ with a paired interval of $[+20.8, +21.3]$. The second, 16K at batch 1 and 30% retention, is $+0.9\%$ with an interval of $[+0.2, +1.7]$: resolved and negligible, a distinction the paired interval can make and the earlier unpaired spread could not. The other 3 straddle zero and we give them no sign; at 64K and batch 1, for instance, the change is -0.1% with an interval of $[-0.6, +0.4]$, that is, nothing. That configuration is also the one that most clearly separates share from batch: its cache is a larger fraction of the dense decode peak than any other single-sequence row, and it still shows no gain, so what the batch-8 row has is not a large share alone but a per-step cache read large enough to be the step's cost.
3. **Claimed: the transfer term's step-time cost in one configuration.** Run back to back on an otherwise empty card, replay-only takes 20.45 s per GRPO step and MaskLift takes 21.10 s, a difference of $+3.2\%$, over 2 repetitions of 30 steps on GSM8K. It is one configuration on one card and is not a general bound. Zhu et al. (2026) asserted an overhead below 10% with no measurement at all; this replaces that assertion rather than extending it.
4. **Not claimed: no end-to-end speed-up.** The throughput figures above are a decode microbenchmark on a serving-shaped workload. We report no end-to-end training-time or quality-per-second comparison between arms, so nothing here establishes that any task score in this paper was reached faster. The benchmark of Zhu et al. (2026) did imply one, by timing prefill together with decode with an evicted arm that prefilled half as many tokens.
5. **Not claimed: no reduction in the memory the episode needs.** The prompt is prefilled with full attention before eviction, so no retention ratio can reduce the prefill peak, and the prefill peak is the episode's high-water mark in every configuration we ran, by between 4.3% and 33.8% over the decode peak: the episode peak is *identical* between the two arms in all 5 configurations of Table 12. What eviction buys is decode-phase working set, not the memory a card must have to run the episode at all. No memory claim in this paper should be read as contradicting that.

6. **Not claimed: no learner-peak reduction, and no learner-peak measurement.** MaskLift's learner realises the mask additively over the full tensors, so there is no physical learner-side memory saving to report (Section K). This revision does not instrument learner memory at all, so we state no learner peak, reduced or unchanged; Appendix E accounts for the transfer term's logit memory from tensor shapes instead.
7. **Claimed narrowly: a batch ceiling, not a context ceiling.** We ran an out-of-memory sweep over batch size at two fixed contexts and report the largest batch that completed in each (Table 14); it is identical for the dense and the evicted arm. We swept no context, so we report no largest context, and the 131K row of Table 13 remains arithmetic about a cache rather than a configuration that ran.
8. **Not claimed: no task quality at any of these lengths.** Every task-quality number in this paper comes from a configuration at or below 4K of trained context (Section J.2). The contexts in this section are serving configurations and carry no information about task quality.
9. **The eviction spike is an implementation note, not a measurement.** `evict_cache_` rebuilds each layer's key and value tensors with `index_select`, which allocates the new tensor before the old one is freed, so a transient above the dense cache exists. The loop runs one layer at a time and keys and values in turn, so that transient is one layer's retained keys or values rather than a fraction of the whole cache, and we did not measure it. We mention it because it is one reason the learner simulates the mask additively instead of slicing, and an implementation of Sparse-RL doing the same would not incur it either.

E Analytical Accounting of the Dense-Logit Memory in the Transfer Term

The transfer term requires the vocabulary distribution of both passes at every generation position, so it materialises a logit tensor of shape $[\texttt{batch}, \texttt{seq}, \texttt{vocab}]$. This section quantifies that cost analytically. **It is an accounting exercise from tensor shapes, not a measurement**; we did not measure the learner peak on the hardware this revision runs on, and claim only that eviction cannot reduce it.

Formula. Let $N = \texttt{batch} \times \texttt{seq}$ be the number of positions in the learner's forward pass, V the vocabulary size, and b the bytes per element. N counts *prompt and response* positions, not the response positions that are scored: our learner calls the model on the whole sequence and then slices, so the full $[\texttt{batch}, P+R, V]$ allocation stays resident. One logit tensor occupies

$$\text{Bytes} \;=\; N \cdot V \cdot b. \tag{9}$$

For a Qwen-class tokenizer with $V = 151{,}936$ (≈ 152K) in BF16 ($b = 2$), one sequence position costs $Vb = 303{,}872$ bytes ≈ 0.29 MiB, so

$$\text{Bytes} \;\approx\; 0.29\ \text{MiB} \times N, \qquad \text{i.e.} \qquad 0.29\ \text{GiB per } 1024 \text{ sequence positions.}$$

Worked examples. Table 15 evaluates (9) at representative shapes. The KL divergence is computed between two distributions, so if the teacher's masked logits and the student's dense logits are both resident at the same time the figure doubles; and if the KL is accumulated in fp32 for numerical stability, the fp32 buffer costs twice its BF16 counterpart. The rightmost column gives the resident total under the least favourable of these conventions (both logit tensors held, one fp32 copy of the current buffer). The released learner is not cheaper than that last column, and an earlier version of this paragraph said it was, on two counts. Its `response_logits` does not pass `logits_to_keep` and returns a view of the full $[1, P+R, V]$ tensor, so the BF16 term is $(P+R)/R$ times what the scored positions alone would cost: $5.7\times$ on GSM8K at its prompt budget, $9\times$ on TL;DR and GovReport and $49\times$ on HotpotQA, with the caveat that these are the budgets rather than the realised lengths (GSM8K's one-shot prompts average 208 tokens, Table 18, which makes its realised factor $1.8\times$). And `kl_chunk` does not bound the fp32 term: `backward` runs once, after the loop over the group's K trajectories, so every chunk's fp32 log-softmax output is retained for the backward pass and the fp32 term is $O(KRV)$ regardless of the chunk size. Chunking without a per-chunk backward or checkpoint frees nothing.

Table 15: Analytical dense-logit memory from (9) for $V = 151{,}936$, BF16 (1 GiB $= 2^{30}$ bytes). N is the number of positions in the forward pass, `batch` $\times$ (`prompt` + `response`), not the number of scored positions. The last four rows are this paper's own training shapes at $N = K(P + R)$, using each task's prompt *budget*; they are group totals rather than a peak, since the BF16 tensors are freed between trajectories while the fp32 buffers are not. "One tensor" is a single $[\texttt{batch}, \texttt{seq}, V]$ logit tensor; "both tensors" holds teacher and student logits simultaneously; "+ fp32 buffer" adds one fp32 copy of the same shape. Analytical accounting, not a measurement.

batch $\times$ seq	**N positions**	**One tensor**	**Both tensors**	**+ fp32 buffer**
$1 \times 1{,}024$	1,024	0.29 GiB	0.58 GiB	1.16 GiB
$4 \times 1{,}024$	4,096	1.16 GiB	2.32 GiB	4.64 GiB
$1 \times 8{,}192$	8,192	2.32 GiB	4.64 GiB	9.27 GiB
$8 \times 2{,}048$	16,384	4.64 GiB	9.27 GiB	18.55 GiB
$8 \times 4{,}096$	32,768	9.27 GiB	18.55 GiB	37.09 GiB
this paper's training shapes, $N = K(P + R)$ at the prompt budget				
GSM8K, $8 \times 1{,}456$	11,648	3.30 GiB	6.59 GiB	13.18 GiB
TL;DR, 8×864	6,912	1.96 GiB	3.91 GiB	7.82 GiB
HotpotQA, $4 \times 3{,}136$	12,544	3.55 GiB	7.10 GiB	14.20 GiB
GovReport, $4 \times 4{,}608$	18,432	5.22 GiB	10.43 GiB	20.87 GiB

Consequence and mitigations. Equation (9) is linear in `batch` $\times$ `seq` and carries a large constant, so at long sequence lengths or large batches the logit tensor becomes the dominant learner-side allocation. Our implementation applies neither of the two mitigations below: both passes' logits are materialised in full over $P + R$ positions, and the chunking at 128 positions bounds neither the logit tensors nor the `float32` buffers, because every chunk's softmax output stays saved until the single backward at the end of the group. The two mitigations that would remove the dependence on N are:

- **Chunked / streaming KL over the sequence.** Partition the scored positions into chunks of C positions, compute the teacher and student logits and the KL contribution for one chunk, accumulate the scalar (and the gradient with respect to the chunk's hidden states), and release the chunk's logits before proceeding. This requires running `backward` on, or checkpointing, each chunk: a Python loop by itself frees nothing, because autograd keeps every chunk's saved tensors alive until the backward pass. Resident logit memory becomes $O(C \cdot V \cdot b)$ instead of $O(N \cdot V \cdot b)$; with $C = 1024$ this is 0.29 GiB per resident tensor regardless of N.

- **fp32 only per chunk.** Where the KL accumulation needs fp32, upcast one chunk at a time rather than the whole tensor, which bounds the fp32 buffer at $C \cdot V \cdot 4$ bytes, but only together with the first mitigation's per-chunk backward: a loop that merely upcasts one chunk at a time still leaves every chunk's buffer saved until the backward pass.

What we can and cannot say about the learner peak. The earlier claim of "zero additional peak memory overhead" is withdrawn, and we do not replace it with a measured one: this revision does not instrument learner memory, so we report no learner peak, changed or unchanged. What (9) establishes is the size of the term, not where it sits relative to the high-water mark: the term is linear in `batch` $\times$ `seq` with a large constant, and at larger `batch` $\times$ `seq` it must be managed by chunking. Whether it lands above or below the peak set by activations, gradients and optimiser states in any given configuration is a question we did not measure.

F Where Each Approach Acts on the Mismatch

Table 16 summarises where importance reweighting, both rejection rules we ran, and mask replay each intervene on the rollout/learner mismatch, and the condition under which each stops working. It is a qualitative summary; the quantitative statement about the full-trajectory ratio product is Theorem 1, with its scope in Section A.5.

Table 16: Where each approach acts on the rollout/learner mismatch. MaskLift's variance claim is scoped: the importance-ratio contribution vanishes *only under exact replay* of the recorded mask; the other sources of gradient variance are unaffected and we do not characterise them, and stability is not implied (Appendix A).

Approach	Off-policy ratio; data used	Failure condition
Importance reweighting	Estimated, clipped; all data	Ratio misestimation
Rejection by rank (our simplification, Appendix C.1)	Filtered, still estimated; 75% of data (a fixed quarter of each group dropped)	Rejection bias
Rejection by support test (Sparse-RL as published, Luo et al., 2026)	Unclipped ξ outside a clipped staleness ratio; data kept is whatever passes the test, and at 30% retention it vetoes 81.8% of GSM8K trajectories, 43.5% on HotpotQA, 19.8% on TL;DR and 13.2% on GovReport (Table 3)	Near-total veto where the mismatch is severe: 457 of GSM8K's scored steps produce no update at all
Router replay (R3)	Routing decisions matched exactly, residual numerics remain; all data	Routing not recorded per token and layer
Rollout-precision match (QaRL)	Reduced, not removed: quantization error is continuous; all data	Residual precision gap
Retention masking (MaskLift)	Identically 1 under exact replay; all data	Mask not reproducible across workers/heads

G THE REGIME MAP

How much the sparse/dense mismatch costs is not a constant of the method; it moves with the retention budget and with the prompt. Figure 5 and Table 17 are the evidence for that, and for the claim that mask replay is flat across the same cells. This sweep, the identity measurements of Section J.4 and the wrong-mask control of Section I.2 all run on Qwen3-1.7B, the policy every training experiment uses. Zhu et al. (2026) measured them on a larger checkpoint without saying so.

Two readings the figure does not support. First, the bottom two retention rows are not three independent eviction rules. Our implementation protects a head window and a tail window and lets the rule choose only among the rest, so at a 10% budget the protected positions consume the budget entirely and all three rules return the identical index set. The 10% row is therefore digit-for-digit equal across the three panels by construction, not by coincidence. At 30% at most one position is left free, and there the proximity rule and the recent-window rule always agree, also by construction: the proximity score is symmetric about the midpoint of the free region, so its two end positions tie exactly and the tie is broken toward the larger index, which is the position the recent-window rule takes. Those two panels therefore agree at 30% while *random* does not. Only the 50%, 70% and 90% rows compare three genuinely different rules. Second, the two rows do not separate everywhere: the two mildest unreplayed cells fall below the largest replayed one, all three lying inside $[0.019, 0.023]$. In the benign corner the unreplayed learner is already as accurate as the replayed one, which is the same statement as the regime claim itself and not a counterexample to it.

What the shot axis varies. The columns are labelled by few-shot count because that is the knob we turned, not because it isolates one property of the prompt. The prompt builder keeps the leading instruction and the final question and inserts the first n worked examples between them, so going from 0 to 8 shots lengthens the prompt, adds content, and moves the question from the start of the prompt to its end at once; the protected windows, being fractions of the prompt's positions, grow with it. What the sweep establishes is that the unreplayed deviation falls as shots are added and that replay is flat across the same cells. It does not separate length from content from question position, and we do not read it as doing so.

G.1 THE RETENTION RULE AND ITS EFFECTIVE BUDGET

The rule is not SnapKV. SnapKV (Li et al., 2024) scores prefix keys by the attention weight they receive from an observation window at the end of the prompt, pools those scores along the sequence, and takes a top-k separately for every attention head of every layer. The rule used in all of

Table 17: The proximity-rule slice of Figure 5 in numbers: mean deviation of the importance ratio from 1 for an unreplayed (dense) learner, 32 sequences per cell. The full sweep is 60 cells (4 shot counts × 3 eviction rules × 5 budgets); the 20 cells of the position-proximity rule (Appendix G.1) are tabulated here because that is the rule used in every training experiment. Columns are few-shot count, i.e. prompt length; rows are the retention budget. Under mask replay every cell falls between 0.013 and 0.023, so only the unreplayed learner is tabulated.

	few-shot examples in the prompt			
retention	0	1	2	8
prompt tokens	98	208	317	1047
0.9	0.081	0.053	0.032	0.019
0.7	0.199	0.103	0.042	0.028
0.5	0.281	0.160	0.066	0.039
0.3	0.321	0.232	0.181	0.059
0.1	0.327	0.359	0.279	0.179

our training runs and in every cell of the sweep uses no attention weights. For a prompt of n tokens at retention r it sets $\mathrm{keep} = \lfloor rn \rfloor$, $\mathrm{head} = \max(\lfloor 0.10\, n \rfloor, 4)$ and $\mathrm{tail} = \max(\lfloor 0.20\, n \rfloor, 4)$, protects $[0, \mathrm{head})$ and $[n - \mathrm{tail}, n)$, and fills whatever budget remains from the middle in decreasing order of the position-proximity score

$$s(p) \;=\; \frac{1}{p - \mathrm{head} + 2} \;+\; \frac{1}{(n - \mathrm{tail}) - p + 1}.$$

If $\mathrm{head} + \mathrm{tail} \geq \mathrm{keep}$ the two windows are re-cut to $\mathrm{head} = \lfloor \mathrm{keep}/2 \rfloor$ and $\mathrm{tail} = \mathrm{keep} - \mathrm{head}$, which exhausts the budget and leaves the score nothing to choose. Three consequences matter for reading this paper. The rule is a function of n and r alone: it never reads the prompt, the model or any attention score, so two prompts of the same length receive the identical mask. The score is symmetric about the midpoint of the free region and falls as one moves inward from either end, so it grows the head and tail windows inward and the retained set is always a prefix block plus a suffix block, never an isolated interior position. And eviction is applied to the decode cache uniformly across heads and layers, so one sequence-level index set is the mask that was actually executed. We cite SnapKV as the family the rule belongs to, namely dropping whole key positions from the prefix while keeping a recent window. No number in this subsection, in Table 17 or in Table 5 measures SnapKV's scorer: every run reported there uses the position rule above. Appendix H reports separate experiments that do run that scorer, per layer and per KV head, and gives both the replay identity at that granularity and the task scores.

What the budget leaves free. Table 18 gives the arithmetic at the prompt lengths this paper uses. At $r = 0.3$, the budget behind every run in Table 5, the protected windows take the whole budget or all but one position at every prompt length these experiments touch: enumerating n from 50 to 4096, 2430 lengths leave 0 free positions, 1617 leave exactly 1, and none leave more. The retained set in the main experiments is therefore fixed by the head and tail windows rather than by the score. Where 0 positions are free the cell is degenerate in a stronger sense: *snapkv*, *recent* and *random* return the identical index set, so the eviction rule is not a variable there at all, and at $r = 0.1$ that holds at every length in the same range. GovReport prompts are cut at 4096 tokens and most of them reach that cap (note § to Table 1), where $r = 0.3$ leaves nothing free. GSM8K trains at 1-shot, on prompts averaging 207.5 tokens in our sweep, where the score picks 1 position out of the 62 retained. The released artifacts record mean prompt lengths for the regime sweep only, so we state the arithmetic as a function of n rather than per task; the conclusion that $r = 0.3$ leaves at most one free position does not depend on the length a particular prompt happens to have.

The re-cut also means the protected fractions are not always 10% and 20%. Whenever $\lfloor 0.10\, n \rfloor + \lfloor 0.20\, n \rfloor \geq \lfloor 0.30\, n \rfloor$, which holds for every multiple of 10 in the range above and for many other lengths, the budget is split evenly instead, so at $r = 0.3$ and $n = 4096$ the retained set is the first 614 and the last 614 positions, that is 15% and 15%, not 10% and 20%.

Table 18: Budget arithmetic for the retention rule described above. n is the prompt length in tokens, $\text{keep} = \lfloor rn \rfloor$, *head* and *tail* are the protected windows after the re-cut, and *free* is the number of positions the proximity score actually chooses, that is $\text{keep} - (\text{head} + \text{tail})$. Rows with $\text{free} = 0$ are marked *degenerate*: the two windows consume the whole budget, and *snapkv*, *recent* and *random* return the identical index set there. The first four blocks are the mean prompt lengths of the regime sweep (Table 17), rounded to the nearest token; the 1-shot block is also the configuration GSM8K trains in. The last block is GovReport's 4096-token cap, which most of its prompts reach. Head and tail are $15\%/15\%$ rather than $10\%/20\%$ wherever the re-cut fires.

setting	n	r	**keep**	**head**	**tail**	**free**
regime sweep, 0-shot	98	0.1	9	4	5	0 (degenerate)
		0.3	29	9	19	1
		0.5	49	9	19	21
		0.7	68	9	19	40
		0.9	88	9	19	60
regime sweep and GSM8K training, 1-shot	208	0.1	20	10	10	0 (degenerate)
		0.3	62	20	41	1
		0.5	104	20	41	43
		0.7	145	20	41	84
		0.9	187	20	41	126
regime sweep, 2-shot	317	0.1	31	15	16	0 (degenerate)
		0.3	95	31	63	1
		0.5	158	31	63	64
		0.7	221	31	63	127
		0.9	285	31	63	191
regime sweep, 8-shot	1047	0.1	104	52	52	0 (degenerate)
		0.3	314	104	209	1
		0.5	523	104	209	210
		0.7	732	104	209	419
		0.9	942	104	209	629
GovReport, at its truncation cap	4096	0.1	409	204	205	0 (degenerate)
		0.3	1228	614	614	0 (degenerate)
		0.5	2048	409	819	820
		0.7	2867	409	819	1639
		0.9	3686	409	819	2458

H SnapKV's Own Scorer, Run Per Layer and Per Head

Appendix G.1 describes the position-proximity rule that every run of Table 5 uses, and records that it is not SnapKV. This section reports the experiment that closes that gap: SnapKV's own scorer (Li et al., 2024), run at the granularity it was published at. Two separate questions are at stake and different measurements answer them. Whether exact replay survives per-head, per-layer selection is an implementation question, and the identity measurement answers it. Whether the correction is still needed once the eviction rule reads the model rather than only the prompt's length is a task question, and the training runs answer that one.

A note on the strategy names in the released code. The released `-strategy snapkv` selects the position-proximity rule of Appendix G.1, and `-strategy snapkv_true` selects the scorer described here. That is an unfortunate pair of names and we have left them as they are, because they are the names the runs were launched with and the ones that appear in the result filenames; renaming them in the release would make the released code disagree with the artifacts it is supposed to reproduce.

What the scorer is, and how it differs from our rule. SnapKV takes an observation window at the end of the prompt, reads the attention each prefix key receives from the queries in that window, pools those scores along the sequence with a max filter, and keeps the top-scoring keys. Every step of that is per head and per layer, because the attention matrix is. Our implementation uses a thirty-two-token observation window and a width-seven max-pooling kernel, one of the two configurations the original reports, and takes the top-k separately for every KV head of every layer; query heads are pooled into

their KV group before selection, because eviction acts on the cache and a grouped-query attention group shares one cached key. Three differences from the position rule matter for reading this paper. First, SnapKV reads the model: the mask is a function of the prompt's content and of the current weights, so two prompts of equal length receive different masks, and the same prompt receives a different mask after a gradient step. The position rule is a function of n and r alone. Second, SnapKV can retain an isolated interior position; the position rule cannot, because its score is monotone inward from two protected windows and the retained set is always a prefix block plus a suffix block. Third, SnapKV selects a different key set in every head of every layer, whereas our rule executes one index set for the whole sequence. The first and third differences are what make the replay question non-trivial, and the third is the failure mode Appendix J.4 lists for an implementation that stores a single sequence-level mask and broadcasts it.

Three implementation facts make exact replay possible at this granularity. They are worth stating because none of them is a property of the objective; each is a property of how the cache and the position encoding happen to be built, and an implementation that violated any of them would need a different mechanism.

- **Equal k per head keeps the cache rectangular.** The budget is the same integer for every head of every layer, so after eviction every head holds the same number of keys. Eviction is then a gather along the sequence axis with a per-head index tensor, and the evicted cache is an ordinary dense tensor the decoder consumes unchanged: no ragged structure, no padding, no per-head length bookkeeping. This is physical eviction and not masking, so the decode-time saving is real and the recorded index set is exactly what was executed.
- **Rotary position information is written at prefill, so dropping keys does not renumber them.** It is applied to the key representations during prefill, before the scorer runs. A surviving key therefore carries its original position in itself, and removing its neighbours does not shift it. The recorded index set names the same key positions at rollout and at replay, with no re-derivation step between them. Had the position encoding been applied after eviction, the survivors would have been renumbered and the recorded indices would no longer identify the same keys.
- **Per-layer masks are installed by forward pre-hook.** The `attention_mask=` argument is one tensor broadcast to every layer, so it cannot express a selection that differs by layer, and a per-head, per-layer mask is not expressible through that interface at all. The learner instead installs a forward pre-hook on each decoder layer holding that layer's recorded per-head key set, which adds the corresponding $-\infty$ bias to the pre-softmax attention logits of that layer only. The arithmetic is the additive-bias replay used everywhere else in this paper, under the same eager attention implementation; only the installation path differs. Because it is a different code path from the sequence-level mask, it needs its own identity measurement rather than inheriting the one in Appendix J.4.

One case the ratio leaves undefined, and what we do with it. SnapKV retains its observation window outright and chooses the rest of the budget from the prefix before it. Expressed as a fraction of the prompt, the budget can therefore be smaller than the window, which leaves the scorer nothing to choose. The published implementation cannot reach this case because its capacity is an absolute token count set above the window. Ours can, and in that case the window shrinks to the budget, so the last $\lfloor rn \rfloor$ positions are retained. We chose that over retaining the whole window, which would hold thirty-two of a fifty-eight-token prompt and hand the arm a larger cache than its column claims. Tokenised at the prompt format of Table 1, the case arises for 3 of the first 1200 TL;DR training prompts and for none of HotpotQA's or GovReport's, so it is a corner of one task and not a regime. It is worth recording for two reasons. An earlier version of the code refused this case rather than defining it, which stopped two TL;DR runs whose data order reached such a prompt while their siblings finished; both were re-run under the rule above and every cell of Table 20 carries its full complement of seeds. The selections the two versions return are identical wherever the earlier one returned at all, so no completed run is affected by the change. And the rollout and the learner reach the rule through the same function, so the branch does not disturb the property this paper rests on: the replayed mask is still the one generation ran under.

The identity holds at per-head granularity. Over 32 sequence-and-budget cells, which are the same sequences at each of the two retention budgets of Table 19, the replayed learner's mean deviation

of the importance ratio from one is 5.6×10^{-6} in `float32`, against 0.288 for the same learner on the same sequences without replay. The worst single *token* is 1.4×10^{-4}, about $26\times$ that mean; the worst single *cell* mean is 1.5×10^{-5}, about twice the position rule's worst cell 7.5×10^{-6} (Appendix J.4). The two statistics are not interchangeable and an earlier version of this paragraph compared the first against the second. Either way what replay leaves behind here is `float32` round-off rather than nothing, and it is coarser round-off than the sequence-level mask leaves. In `bfloat16` the mean is 2.9×10^{-2} against 0.282, and there the claim has to be confined to the mean: the worst replayed *token* is 0.761, larger than the average *unreplayed* cell, and the worst replayed cell mean is 0.055, so a reader who takes the mean as a per-sequence guarantee at this precision would be misled. The `bfloat16` figures are the ones a training loop realises, since our runs compute in `bfloat16`. What is new in this measurement is only that the property survives a rule selecting different keys in every head of every layer, which is the case Appendix J.4 flags as a hazard.

Ratio deviation at matched budget, against the position rule. Table 19 places the two rules side by side at matched retention. For an unreplayed learner, SnapKV produces a *larger* deviation than the position rule at both budgets. Under replay its deviation is small and flat across both, but not as small as the position rule's: the replayed SnapKV figure sits above 0.023, the top of the replayed band across all 60 cells of the regime sweep. Both columns are measured in `bfloat16`. The two rules are measured on different sequence sets of different size, drawn the same way from one-shot GSM8K prompts, so the comparison is between two samples and not two measurements of one sample.

Table 19: Mean deviation of the importance ratio from 1 at matched retention, on one-shot GSM8K prompts drawn the same way as the proximity-rule slice of Table 17 but on a different and smaller set of sequences. *position rule, dense* is the corresponding cell of Table 17; *SnapKV, dense* is the same measurement with SnapKV's per-layer, per-head scorer in place of the position rule; *SnapKV, replay* is the replayed learner under that scorer. Larger is further from on-policy. All three columns are `bfloat16`.

retention	position rule, dense	SnapKV, dense	SnapKV, replay
30%	0.232	0.345	0.029
50%	0.160	0.220	0.029

Task quality under eviction by SnapKV's attention scorer. Both arms train at the retention budget every main-table run uses, on the tasks of Table 20, against each task's untrained checkpoint as the collapse floor. The uncorrected learner generates under eviction by that scorer and scores densely, exactly as the naive arm of Table 5 does under the position rule; MaskLift generates under the same eviction and replays the recorded per-head mask in the learner. Both arms and every floor in this table were scored by the shared task evaluator at the training caps. That is the evaluator that produced the collapse floors and the clip-and-reject column, and not the one that produced the GSM8K dense and MaskLift columns of Table 5, which is the seam Appendix C discloses; comparisons inside this table are therefore like for like, and comparisons from this table to the GSM8K column of the main table are not.

Two things hold across all four tasks and neither of them is a stability claim. MaskLift finishes above the uncorrected learner on every one, from the same rollouts under the same mask, and it does not collapse on any. The uncorrected learner collapses on two of the four, TL;DR and GovReport, where every one of its runs finishes below the untrained checkpoint; on the other two it survives and lands below MaskLift, by 27.1 points on GSM8K and by 1.5 on HotpotQA.

Which tasks break is not the same under the two eviction rules, and that is worth stating because it is the kind of thing a reader would otherwise assume. Under the position rule the uncorrected learner collapsed on GSM8K, in 5 of 5 runs, and survived on GovReport (Table 5); under SnapKV's scorer it survives on GSM8K and collapses on GovReport. We do not offer that as an ordering of collapse rates between rules. Three seeds per arm cannot support one, and no claim in this paper rests on one compressed arm collapsing less often than another. What the table does contribute is the part that does not move: a scorer that reads the model rather than only the prompt's length does not remove the cost of training a learner on a policy other than the one that generated. The cost is paid in collapse on two tasks and in quality on the other two, and the correction removes it on all four.

Table 20: Training under eviction by SnapKV's own per-layer, per-head scorer, at 30% retention. "Collapsed" counts runs finishing below that task's untrained checkpoint, the threshold used throughout this paper, and the mean and s.d. describe the runs that did not collapse, with "–" marking a cell with no surviving run. The last column is the untrained checkpoint's own score, which is that threshold. Every cell here was scored by the shared task evaluator at the training caps (Appendix C), including the floors, so the comparisons within this table are on one protocol. Metrics are each task's own, as in Table 5, so columns are comparable down a row and not across rows.

	naive eviction, dense learner				**MaskLift**			
Task	n	collapsed	mean	s.d.	collapsed	mean	s.d.	**floor**
GSM8K	3	0	49.0	3.4	0	76.1	2.0	40.7
TL;DR	3	3	–	–	0	20.6	3.8	16.0
HotpotQA	3	0	59.7	1.3	0	61.2	1.2	32.9
GovReport	3	3	–	–	0	20.1	1.2	17.0

What this does not establish. One model (Qwen3-1.7B), one retention budget for the training runs and the two budgets of Table 19 for the deviation measurement, and 3 seeds per training arm. Nothing was swept for either arm: λ was carried over unchanged from the GSM8K selection of Appendix J, and the learning rate, group size, step count and generation caps are those of every other arm. The comparison is therefore matched in eviction rule, budget and optimiser settings, and in nothing else. The deviation measurement and the training runs are also not the same kind of quantity: one is measured on fixed sequences at a fixed checkpoint and the other is the outcome of a training run, and we have no mechanism connecting them, so the table of deviations does not predict the table of outcomes and we do not use it that way. Finally, nothing here speaks to H2O, StreamingLLM, token merging or quantization. What generalises from this section is the implementation claim: a mask recorded per layer and per head replays to `float32` round-off, and can be installed in a learner whose attention interface has no per-layer mask argument at all. The task numbers are one eviction rule under one set of settings.

One difference between the released file and the earliest of these runs. The learner builds its per-layer masks one layer at a time, inside the pre-hook. An earlier version built all of them before the forward pass, which is the same arithmetic but holds every layer's mask at once: at GovReport's prompt length that is 17.7 GiB of mask before the model, the optimiser state and the gradients are counted, and it does not fit. The GSM8K runs in Table 20 were trained with the earlier version and the rest with the released one. The selection each returns is the same function of the same inputs, and we re-measured the identity of Table 19 after the change rather than assuming it.

I Failure Modes and Limitations of Scope

I.1 The complete list of limitations

The main text carries a short version of this list. The full one is here, and the ordering is by how much each could change a reader's assessment, not by how comfortable it is to state.

- **Theory covers Step 1 only.** Exact replay removes the sparse/dense ratio's variance contribution, not total gradient variance, and is no stability result; Appendix A bounds the *full-trajectory-product* estimator, while practical PPO/GRPO applies the PPO $\min$ to per-token ratios, which introduces bias and, when the advantage is negative, leaves the per-token contribution unbounded in the ratio (Section A.5). **For Step 2 we prove nothing:** Remark 1 is a mechanism, not a bound on the transfer gap, which is reported as a measurement.
- **Mask replay is a precondition.** The identity fails silently if the learner recomputes eviction, or if resharding, head layout, dynamic batching or packing breaks the mask-to-sequence association; we have not characterised the resulting bias.
- **Runs collapse, and we cannot say why.** 12 of 66 held-out runs trained under eviction ended below the untrained checkpoint, in every such arm of Table 5 including both that replay the mask, though not in every eviction arm this paper reports: under SnapKV's own scorer MaskLift has none (Appendix H); the dense baseline has not collapsed in any of its 14. We report the counts rather than filtering them and have *not* identified the cause. That

asymmetry is suggestive but we do not claim compression causes collapse, and no claim that one compressed arm collapses less often than another is supported by our data.

- **Systems scope and cost.** Prefill precedes eviction, so eviction cannot reduce the prefill peak; the learner realises the mask additively, so it cannot reduce the learner peak either, and we measured neither. Savings are confined to post-eviction decode. The transfer term adds a forward pass and materialises $[\text{batch}, \text{seq}, |\mathcal{V}|]$ logits for both (Appendix E); measured back to back on an empty card it costs $+3.2\%$ of step time: 20.45 s against 21.10 s per GRPO step over 2 repetitions of 30 steps on GSM8K.
- **Our implementation caps trainable context.** We replay the mask as an additive bias on the pre-softmax logits under an eager attention implementation, so the learner materialises attention scores and cost grows with the *square* of length. GovReport is truncated to 4K for this reason, and it is the main obstacle to scale in our code. This is a limit of what we ran, not of exact replay: PyTorch's FlexAttention takes a user-supplied mask function over query/key index pairs, fuses it into a FlashAttention-style kernel and never materialises the score matrix, applying the mask pointwise inside every tile it visits while block sparsity decides only which tiles are visited. We did not implement this (Appendix K).
- **Discrete eviction only.** Recordable token selection (SnapKV, H2O, StreamingLLM) replays to floating-point round-off, which we measure for SnapKV at per-head, per-layer granularity (Appendix H) and assert for the rest, but quantization perturbs surviving entries continuously, so the identity covers only the discrete part (Appendix L); token merging is out of scope.
- λ **has no schedule.** A single fixed λ is selected once, on GSM8K validation, and reused unchanged. Past some value the transfer target dominates the reward and the benefit disappears; we offer no adaptive rule and no evidence that a value transfers across datasets.
- **Untested regimes and coverage.** The largest model trained here is Qwen3-1.7B; the longest trained context is 4K. The published objective of Luo et al. (2026) is run as written on all four datasets at one budget (Appendix C.1), at hyperparameters not swept per dataset; a per-dataset search of it, or of our own simplification beside it, could close part of the reported gap. Every training run uses the position-proximity eviction rule of Appendix G.1 except the arms of Appendix H, which use SnapKV's published scorer per layer and per KV head; no other scorer family (H2O, StreamingLLM, token merging) was trained at all. No memory-matched dense rollout and no learner-side sparse-kernel comparison; naive eviction is a diagnostic ablation, not a competing algorithm.

I.2 Dropped evidence under non-semantic masks

MaskLift aligns the learner with whatever mask the rollout used; it does not recover information that was never in the rollout's context. The KV-selection ablation is the relevant evidence, and the quantity it measures is the mean deviation of the sparse/dense ratio from 1, not a task score. The 4 configurations are 1 and 8 shots crossed with 30% and 50% retention, 32 sequences and 48 decoded tokens each. Across them exact replay holds the deviation between 0.016 and 0.020, while a same-size random mask gives between $1.00\times$ and $9.2\times$ as much: never less, and at 8 shots and 30% retention no more, with the largest factor at 1 shot and 50%. The measurement records one mean per configuration, so we cannot report a spread within a configuration, and four configurations do not support a distribution over configurations either. Two cautions follow.

First, no task score was measured under a wrong mask, so this ablation says nothing about how much task quality a non-semantic mask costs, and it does not show that per-prompt evidence is retained. A retention budget keeps some fraction of key tokens on every prompt, and a metric averaged over a test set can remain respectable while, on individual prompts, the specific span that a correct answer depends on has been evicted: a leading instruction, a document title, or the single supporting sentence in a multi-hop chain. We ran neither a per-prompt error analysis identifying which prompts lose their supporting evidence nor any task evaluation under a random mask, and we regard both as gaps in the present evaluation rather than resolved questions.

Second, the transfer term cannot compensate for evidence that was never sampled. It carries the masked policy's improvement into the dense one, but that improvement was itself learned from trajectories generated without the evicted content. When the evicted content is necessary, not merely helpful, to produce a correct response, no amount of alignment between rollout and learner recovers it, and the correct intervention is a better eviction policy rather than MaskLift.

J THE λ SWEEP

Selected on the GSM8K validation split; the test split was not consulted. The interior points carry few seeds, so this supports the existence and the bounds of a useful λ rather than a precise optimum.

Table 21: λ on GSM8K validation. Same convention as Table 5: collapses counted separately, mean and s.d. over the runs that did not collapse.

λ	runs	collapsed	mean	s.d.	card
0	18	3	70.7	7.9	14A/4B
10^{-3}	3	0	75.0	6.8	2A/1B
10^{-2}	18	2	80.0	5.3	13A/5B
0.1	2	0	68.3	2.8	2B

J.1 OVER-REGULARISATION BY THE TRANSFER TERM

λ controls a trade-off and is not self-limiting. Above some value the transfer target must compete with the reward signal; our sweep does not locate that value. At $\lambda = 0.1$ the two runs we have score 66.3 and 70.3, below the 10^{-2} mean and no better than no transfer term at all, but both sit inside the $\lambda = 0$ range, and two runs establish neither harm nor its absence (Figure 3). Both also ran on machine B while 10^{-2} is mostly machine A, so the gap is read against the cross-card band of Appendix C.5 as well as against the seed spread. We use a single fixed λ with no schedule and no automatic adaptation, selected on a held-out validation split and never on the reported test split; we do not know the safe range on datasets we did not sweep, and a practitioner adopting MaskLift should treat λ as a hyperparameter requiring validation rather than as a setting that transfers.

J.2 CONTEXT LENGTHS ACTUALLY TRAINED, VERSUS CAPACITY MEASURED

The longest context we *trained* on is GovReport, truncated to 4K tokens (Table 1); its natural prompts exceed 8K, which is why exact replay's context ceiling is the reason for the truncation rather than an incidental detail. Every task-quality number in this paper comes from a configuration at or below that length. A capacity ceiling is a different quantity from task quality. We measured a batch ceiling at 16K and 64K of context (Table 14), well beyond the trained length, but no task quality at any length beyond 4K. We have not validated MaskLift beyond 4K trained context, and the gap between 4K and those lengths is unevaluated for quality. Whether the transfer term still carries the improvement when the evicted fraction spans tens of thousands of tokens is an open question, and one our context ceiling currently prevents us from answering.

J.3 EXTREME COMPRESSION IS CHARACTERISED ON ONE DATASET

The retention sweep in Section 4.4 runs on GSM8K at two budgets, 30% and 50%. It cannot support a claim about graceful degradation in general, and the dataset it is run on matters more here than it usually would: how much the mismatch costs is governed by how much information eviction destroys, so the benign corner is the long, redundant prompt at a mild budget, not the short one; in the sweep, 8 shots at 90% retention gives 0.019 and 0 shots at 10% gives 0.359. Across our four datasets the governing variable is redundancy rather than length as such: TL;DR has the shortest prompt budget of the four and is nonetheless benign, while GSM8K's one-shot prompt is longer and is not. The same caveat applies to λ, which we sweep on one dataset and then hold fixed.

J.4 THE MASK-REPRODUCIBILITY PRECONDITION

What "identically one" means in finite precision. The identity is exact in exact arithmetic; what a measurement can show is that the residual is floating-point error and not a modelling gap. Over 18 configurations, three retention ratios $\times$ three eviction rules (proximity, recent, random) $\times$ two prompt lengths, the replayed learner's mean ratio deviation is 3.4×10^{-6} in `float32`, worst case 7.5×10^{-6}, against 0.118 for the same learner without replay. That is four orders of magnitude, and it does not vary with the eviction rule, which is the sense in which replay treats the rule as a black box.

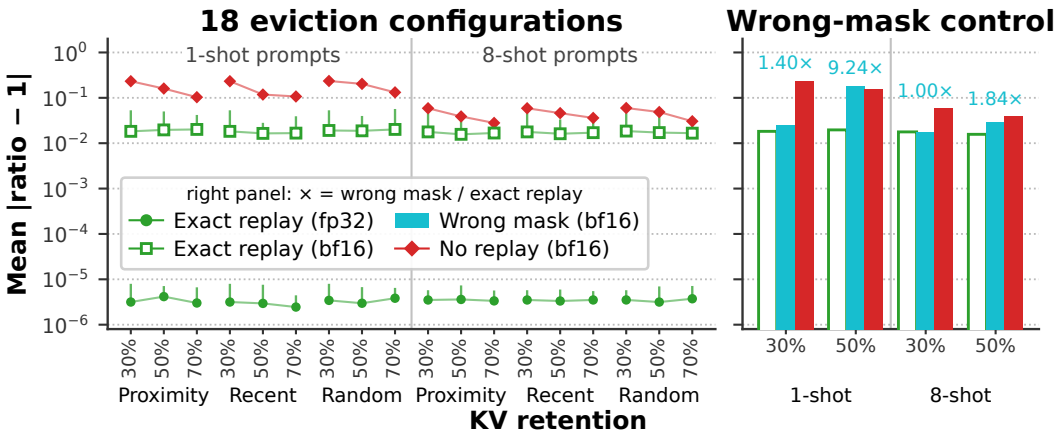


Figure 6: **Left:** replayed and unreplayed ratio deviation over the 18 eviction configurations. In `float32` exact replay sits four orders of magnitude below the unreplayed learner and is flat in the eviction rule; in `bfloat16` it sits at 1.8×10^{-2}, the floor a training loop in that precision realises. The two precisions are measured on different numbers of sequences (16 and 32), and the unreplayed series drawn is the `bfloat16` one, which is the pair Section 3 quotes; the `float32` unreplayed mean is 0.118. **Right:** the same-size wrong-mask control on a shared log scale; annotated factors are wrong mask over exact replay.

In `bfloat16` the same measurement gives 1.8×10^{-2}. This is worth stating plainly rather than relegating: our training runs compute in `bfloat16`, so the deviation actually realised during training is of that order, not of order 10^{-6}. It remains about $6.0\times$ below the unreplayed learner's deviation at the same precision (0.107), and it is a property of the arithmetic rather than of the method. But a reader who takes 3.4×10^{-6} to be the number their bf16 training loop will see would be misled, and we would rather they were not.

The same three measurements on a second checkpoint. Every *training* run in this paper is Qwen3-1.7B, and Section 6 states that as a scope condition. The three measurements in this section and the two that follow it are not training runs, and they were also taken on Qwen3-4B-Instruct-2507 before the policy was fixed. Over the same 18 configurations the replayed learner's mean ratio deviation there is 3.3×10^{-6} in `float32` and 1.5×10^{-2} in `bfloat16`, against 0.107 and 0.101 without replay; over the 60 regime cells the unreplayed deviation spans 0.017–0.320 while replay holds it at 0.011–0.020; and the same-size wrong mask costs between $1.04\times$ and $10.7\times$ exact replay over 4 configurations. Every one of those is within the spread of the 1.7B figures. This bounds the scope condition from one side and one side only: what does not depend on the model size is the *mechanism*, and nothing here says anything about what the task scores would do at a scale where the cache dominates the weights. The 4B result files ship with the archive.

Theorem 2 holds only under exact replay, and exact replay is an implementation obligation rather than a property of the objective. Concretely, the learner must reproduce, for each response y_i, the retention mask M_i at the granularity at which eviction actually occurred: per attention head, per layer, per query position, over *key* positions, and associated with the correct token positions of the correct sequence. The guarantee fails whenever that association is broken. Failure conditions we consider realistic:

- **Across workers.** If rollouts are generated by inference workers and consumed by separate training workers, the mask must be serialised and shipped alongside the token ids. A pipeline that transmits only (x, y_i, r_i) silently reverts to a dense learner, which is exactly the mismatched configuration our diagnostic ablation measures.
- **Across heads and layers.** Rules such as SnapKV select different key positions for different heads. Storing a single sequence-level mask and broadcasting it to all heads then replays a mask that no head actually used, so $\pi_\theta(\cdot \mid x, M) \neq \pi_{\text{sparse}}$ and the ratio is no longer identically one. Our own rule evicts uniformly across heads and layers (Appendix G.1),

so a sequence-level mask is the mask that was executed and this failure cannot arise in those runs. It is a hazard for a per-head rule, and we have now tested one: Appendix H runs SnapKV's scorer with a separate top-k per layer and per KV head, records the mask at that granularity, installs it in the learner by per-layer forward pre-hook, and measures the identity there in both precisions. The mask has to be recorded and replayed at the granularity eviction executed; recording it at a coarser one is the failure this bullet describes.

- **Dynamic batching and padding.** Continuous batching, sequence packing, left/right padding changes, and sorting by length all permute or re-index positions between rollout and learner. Any re-indexing that is not applied to the mask in the same way as to the tokens misaligns the mask by an offset, which is worse than no mask at all because it masks the wrong keys.
- **Eviction that continues during decoding.** Our rule fixes the retained set once, after prefill, so one mask per sequence *is* what executed. A rule that keeps evicting as it decodes, and H2O's accumulated-attention rule run online is the obvious case, makes the visible set a function of the generation step, and the recorded object is then one mask per step rather than one per sequence. Theorem 2 still covers it, since its hypothesis is stated per query position, but an implementation that records the final mask and replays it for every step replays a set no step used. We did not run such a rule.
- **Non-determinism in eviction.** If the eviction decision depends on quantities that are not reproducible at replay time and the mask is recomputed rather than recorded, the replayed mask may differ from the executed one. MaskLift requires the mask to be *recorded at rollout*, not recomputed in the learner.

When exact replay is violated, the resulting estimator is off-policy by an unmeasured amount and sits somewhere between MaskLift and the mismatched-learner ablation; MaskLift provides no guarantee in that regime and offers no mechanism for detecting the violation. We recommend implementations assert mask–sequence consistency (for example, by verifying that the learner's recomputed rollout log-probabilities match the recorded ones to numerical tolerance) rather than assume it.

K Discussion: Learner-Side Sparse Attention Kernels

A natural question is whether a learner-side sparse attention kernel (one that attends only over the retained KV entries, rather than over the full tensor with an additive $-\infty$ mask) would provide the same alignment while also reducing learner compute and memory. It would, subject to one condition, and we regard this as complementary to rather than in competition with the contribution of this paper.

The alignment property of Theorem 2 depends only on *which key positions are visible* at each query position and head, not on how that visibility is implemented. A sparse kernel that executes exactly the recorded mask therefore yields the same identity $\pi_\theta(\cdot \mid x, M) = \pi_{\text{sparse}}(\cdot \mid x)$ and the same conclusion, up to floating-point reassociation. In addition, it would avoid materialising the masked-out portion of the attention computation, cutting learner attention FLOPs and the memory held for those entries, benefits that additive-mask simulation does not provide. The condition is exactness, and block granularity does not by itself violate it: in a kernel of the FlexAttention kind the mask function is applied pointwise to the scores inside every tile that is visited, and block sparsity decides only which tiles are visited, so the executed visibility set is the recorded one and block size changes the speed-up rather than the mask. What breaks the premise of Theorem 2, in proportion to the mismatch, is a kernel that rounds the mask itself to tile granularity and so attends over a superset or subset of the recorded positions.

We do not claim this benefit, because we did not implement it. Our learner uses mask simulation over the full dense tensors, which is why the learner peak is unchanged rather than reduced: **no physical learner-side memory saving is claimed anywhere in this paper**. The contribution we do claim is the alignment mechanism (recording the discrete information set used at rollout, replaying that exact mask in the learner, and using the RL-improved masked policy as a teacher for the dense one it will be deployed as, recovering information lost under the aligned sparse policy), and that mechanism specifies what a sparse learner kernel would have to execute in order to be correct. The two are orthogonal and compose.

L Discussion: The Boundary with Quantization-Based Compression

MaskLift's exact-replay property applies to compression schemes whose effect on the forward pass is a *discrete* choice of which entries are visible. Eviction methods (SnapKV, H2O, Recent, Random) are of this kind: their entire effect is captured by a binary mask over key positions. Where the retained set is fixed once after prefill, as in the rule we ship, that mask is one vector per sequence and replay reproduces the rollout's visible set exactly; a rule that keeps evicting while it decodes is still covered, because Theorem 2 quantifies over every query position, but its mask has to be recorded per generation step rather than once.

Quantization is not of this kind. A method such as KIVI keeps every entry and perturbs its numerics, so the rollout policy differs from an unquantized dense evaluation by a continuous error that depends on the values themselves. Three consequences follow, and we state them as a boundary rather than as an extension.

1. **A combined eviction + quantization scheme has a replayable component.** Its eviction decisions are discrete and can be recorded and replayed exactly, which removes the mismatch attributable to eviction. This is worth doing.
2. **The quantization error itself is not reproduced by a binary mask.** A mask can only set attention weights to zero or leave them unchanged; it cannot reproduce a dequantized key or value tensor. Under exact mask replay of the eviction component alone, the learner's masked policy still differs from the quantized rollout policy by the quantization error, so the importance ratio is not identically one and Theorem 2 does not apply to the combined scheme.
3. **Thresholding does not fix this.** Converting the continuous quantization error into a binary keep/drop decision (for instance by masking entries whose quantization error exceeds a threshold, or by probabilistic masking) defines a *different, discrete* compression policy. The learner could then replay that new policy exactly, but the object being replayed is no longer the quantized rollout that was actually executed, so exact replay of the thresholded mask does not give exact alignment with the quantized generator. One would be aligning to the wrong policy, which is the mismatch we set out to remove.

Extending exact replay to quantization would require recording and replaying the quantization state itself (per-group scales and zero-points, or the dequantized tensors), which is a different mechanism with different storage and bandwidth costs. We do not study it here. The honest statement of scope is that MaskLift's guarantee covers discrete eviction, that it can be applied to the eviction component of a hybrid scheme, and that it does not cover the continuous component.

The quantized-rollout case has been attacked from the other side. QaRL (Gu et al., 2026) keeps the rollout quantized and moves the learner to it, running the training-side forward at the rollout's precision so the two agree approximately, and adds a sequence-level clipped objective to contain what disagreement is left. That is the same goal as ours, alignment of the compute path rather than correction of the estimator, reached by matching numerics instead of replaying a discrete record. The two are complementary in principle and we have not combined them: under a hybrid eviction-plus-quantization scheme the eviction component is replayable exactly and the quantization component is not, so QaRL's route is what would have to cover the residual. We report this as the shape of the boundary, not as a result: we ran no quantized rollouts.

M Results Across Model Scales and Additional Benchmarks

This section previously reported a table across four additional model scales and two additional benchmarks. Those numbers, reported by Zhu et al. (2026), came from an implementation since established not to execute the method it described, so they are removed rather than reprinted, and they are not restored here in any form. We did not repeat the scale study with the released implementation in time for this submission. The reader is owed the reason for the gap rather than a silent omission. Two facts bound what we can say about scale here. First, exact mask replay requires the learner to materialise attention scores, so its cost grows with the square of sequence length and with K; the

scales we could run within our compute budget are stated in Table 1 and nothing beyond them is claimed. Second, the mechanism the paper argues for (that the orientation of the transfer term is decided by which distribution the policy gradient enters) is not a statement about model size, and we have no evidence that it inverts at larger scale. Absence of a scale study is a gap in our evidence, not support for a scale-invariance claim.

N Why the Sequence-Level Token Saving Is Below the Per-Prompt Saving Target

A rollout at a nominal 50% retention target does not save 50% of the sequence's KV tokens, and the shortfall is not an implementation defect. This section shows that the sequence-level saving is bounded above by $1 - r$, the per-prompt *saving* target and is in general strictly below it, so the shortfall is a structural property of where eviction applies and not a sign that the retention budget was not met.

Where eviction does and does not apply. Three facts about the rollout pipeline fix the accounting.

1. **Prefill is dense.** The prompt is prefilled with full, uncompressed attention; eviction runs afterwards. This is why eviction cannot reduce the peak allocated memory of a rollout episode and why no saving whatsoever is realised during prefill.
2. **Eviction applies to the prompt KV.** The retention ratio is a budget over the key/value entries produced by the prompt: retaining 50% means half of the *prompt* entries survive.
3. **Generated tokens are appended uncompressed.** Each decoded token appends a full KV entry, which is not subject to the recorded eviction mask.

Derivation. Let P be the number of prompt tokens, G the number of generated tokens, and r the retention ratio applied to the prompt KV. Counting resident KV entries at the end of the sequence,

$$\text{dense} \;=\; P + G, \qquad \text{compressed} \;=\; rP + G,$$

so the sequence-level saving is

$$s \;=\; \frac{(P+G) - (rP+G)}{P+G} \;=\; \frac{(1-r)\,P}{P+G} \;=\; \frac{1-r}{1+g}, \qquad \text{where } g \;=\; \frac{G}{P} \;\geq\; 0. \tag{10}$$

At $r = 0.5$ this gives $s = 0.5/(1+g) \leq 50\%$, with equality only in the limit $g \to 0$. Because s is strictly decreasing in g, *any* non-zero generation length pushes the achieved saving below the retention target, and the size of the shortfall is determined entirely by the prompt-to-generation length ratio. Two further consequences are worth stating: the shortfall is identical for naive eviction, clip + reject, and MaskLift, since all three roll out under the same mask at the same retention ratio, and no implementation change to the learner can affect it, since it is a property of the rollout cache.

Inverting the formula. Rearranging (10) at $r = 0.5$ gives the generation share implied by an observed saving,

$$g \;=\; \frac{1-r}{s} - 1 \;=\; \frac{0.5}{s} - 1, \qquad \frac{G}{P+G} \;=\; \frac{g}{1+g} \;=\; 1 - 2s. \tag{11}$$

This is useful as a consistency check rather than as a result: a reported sequence-level saving and a reported prompt-to-generation ratio determine each other exactly at a given retention target, so a pair that does not satisfy (11) indicates that the two were measured under different conditions. We do not report a sequence-level token saving in this revision; our systems measurements are of the decode-phase cache and of decode throughput (Figure 4), so the identity is stated here for a reader checking a figure of that kind elsewhere, including the one Zhu et al. (2026) reported.

O Complete Training Procedure

Algorithm 1 states the full MaskLift training loop. Notation follows Section 3: for a prompt x the policy generates K responses $y_1, \ldots, y_K$, $\hat{A}_i$ is the group-relative advantage of response y_i, $|y_i|$ its

number of sampled tokens, and M_i is the *retention mask* recorded for y_i during rollout (in general a per-head, per-layer binary indicator over key positions, 1 = retained, 0 = evicted; our own rule evicts uniformly across heads and layers, so the recorded indicator is constant along those two axes). GRPO uses no value network. The loss is written as a minimisation objective and matches Eq. (5) term for term: $\mathcal{L}_{\text{GRPO}}$ below is the clipped surrogate less the reference KL, so $-\mathcal{L}_{\text{GRPO}}$ is $\mathcal{L}_{\text{PG}}$ of Eq. (4), and the transfer KL is added with coefficient λ exactly as there. The reference policy is evaluated *with the same retention mask* as the policy, so that the reference KL is taken between two distributions defined on the same information set. The dense pass of Track B runs sequentially after the masked pass of Track A.

Conventions the algorithm leaves implicit. *Behaviour policy.* No second copy of the model is kept. The stored $\log \pi_{\theta_{\text{old}}}(y_{i,t} \mid x, y_{i,<t}, M_i)$ are the log-probabilities the sampler emitted while generating, so θ_{old} is by construction the parameters at the start of the step, and $\theta_{\text{old}} = \theta_0$ at the first step. *Buffer lifetime.* Each step generates its own group, takes exactly one gradient step on it and discards it: one epoch, no accumulation, no reuse across steps, so the loop is strictly on-policy. The consequence for Eq. (4) is that $p_{i,t}$ departs from 1 only through the rollout/learner information mismatch and through finite-precision recomputation, never through parameter drift; we keep the clip anyway because the objective is GRPO's and because the identity is exact only in exact arithmetic. *Degenerate groups.* A group whose K rewards have zero spread gives every response the same advantage; we redraw the prompt up to six times, and if the group is still degenerate no optimiser step is taken and no ratio is recorded for that step. *Reference model.* $\beta = 0$ in every run, the $\beta > 0$ branch is never taken, and no reference model is constructed anywhere in the released code. *Lengths, EOS and padding.* y_i ends at and includes the first EOS token, or at the generation cap when none is emitted, and $|y_i|$ is that length. Each response is scored in its own forward pass over x followed by y_i, so no padding position is ever scored and no packing or re-indexing occurs between rollout and learner. The retention mask is applied to response queries only, so $y_{i,1}$, which the actor drew from the dense prefill, is scored densely by the learner as well.

The ratio diagnostic, exactly. Every reported ratio deviation is $d_i = \frac{1}{|y_i|} \sum_t |p_{i,t} - 1|$ for one response, and the per-step figure is the mean of d_i over the group's responses. It is computed in the Track A forward pass of that step, that is at $\theta = \theta_{\text{old}}$ and *before* that step's update, so it measures replay error at fixed behaviour parameters and not the policy drift that a later update introduces. For the clip-and-reject arm it is taken over all K responses, before rejection: the arm is not charged for the trajectories it discards, and the figure shows that its remaining ratios are where the uncorrected learner leaves them. The published objective is the one arm where the convention runs the other way, because its support test rejects before the diagnostic is recorded: its values are conditioned on having passed a threshold applied to the very quantity plotted, and the series is therefore not comparable with the others. On GSM8K that leaves it 3 values out of 63 logged step-rows, which is why Figure 2(b) draws it there as isolated points with no band. Figure 2(b) plots the median of that per-step value across seeds with an interquartile band, at every tenth step and at the last; steps at which every group was degenerate produce no value and leave a gap rather than a zero. The offline measurements in Table 17 and Figure 6 use the same d_i, averaged over sequences instead of over a group.

P Broader Societal Impacts

This work reduces the working memory a long-context rollout holds while it decodes, and it reduces the decode-phase cache in exact proportion to the retention budget (Table 13). It does not reduce what bounds context and batch on a fixed device: both arms prefill the full prompt before either evicts, so the episode's peak is unchanged and the saving is working set during decode (Appendix D, item 5). The likely effect is that long-context alignment experiments become feasible at smaller hardware budgets than before, which lowers the cost of alignment and safety tuning work for groups without large clusters and reduces the energy used per experiment at a fixed workload. We note that efficiency gains are frequently reinvested in larger workloads rather than realised as absolute savings, so we do not claim a net reduction in energy use.

The same reduction in cost applies regardless of the objective being optimised. A cheaper alignment pipeline is also a cheaper pipeline for tuning a model away from its safety behaviour or toward generating harmful content, and MaskLift is agnostic to the reward function it is given. This is a

Algorithm 1 MaskLift with GRPO

Require: Policy π_θ; reference policy π_{ref}; prompt dataset $\mathcal{D}$; group size K; learning rate α; clipping parameter ϵ; reference-KL coefficient β; transfer coefficient λ; any discrete eviction rule with a retention budget. *No value network is used.*

while training **do**
  $\theta_{\text{old}} \leftarrow \theta$ *// the behaviour parameters of this step; $\theta_{old} = \theta_0$ at the first step*
  Draw a prompt batch $x \sim \mathcal{D}$ *// our runs use one prompt per step; nothing below depends on that*
  // redraw, up to 6 times, while the group's K rewards have zero spread: such a group carries no advantage
  // Phase 1: sparse rollout with retention-mask recording
  for each prompt x drawn this step **do**
    for $i = 1$ to K **do**
      Generate $y_i \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$ with physical KV eviction
      Record the retention mask M_i over key positions, at the granularity eviction executed *// per head and layer in general; one vector per sequence for our rule*
      Store the rollout log-probabilities $\log \pi_{\theta_{\text{old}}}(y_i \mid x, M_i)$
      Compute the reward $r_i = r(x, y_i)$
    end for
    // Group-relative advantage over the K responses of this prompt
    $\hat{A}_i \leftarrow \dfrac{r_i - \text{mean}_{j \le K}\, r_j}{\text{std}_{j \le K}\, r_j + \delta}$ for $i = 1, \ldots, K$
    Collect $\{(x, y_i, M_i, \hat{A}_i, \log \pi_{\theta_{\text{old}}}(y_i \mid x, M_i))\}_{i=1}^{K}$ into this step's batch *// consumed by the single update below and then discarded; nothing is reused across steps*
  end for
  // Phase 2: dual-track optimisation, exactly one gradient step on this step's batch
  // Track A: masked (aligned) policy gradient
  Masked forward pass: $\pi_\theta(\cdot \mid x, M_i)$, replaying M_i exactly
  Per-token ratios $p_{i,t} \leftarrow \dfrac{\pi_\theta(y_{i,t} \mid x, y_{i,<t}, M_i)}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid x, y_{i,<t}, M_i)}$
  // clip per token, average over a response's tokens, then over the group: long responses are not up-weighted
  Clipped surrogate $\mathcal{L}_{\text{Surr}} \leftarrow \frac{1}{K} \sum_{i=1}^{K} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \min\big(p_{i,t}\hat{A}_i,\ \text{clip}(p_{i,t}, 1-\epsilon, 1+\epsilon)\, \hat{A}_i\big)$
  if $\beta > 0$ **then**
    Masked reference forward pass: $\pi_{\text{ref}}(\cdot \mid x, M_i)$ *// same retention mask*
    $\mathcal{K}_{\text{ref}} \leftarrow \mathbb{E}_{i,t}\big[D_{\text{KL}}\big(\pi_\theta(\cdot \mid x, y_{i,<t}, M_i) \,\|\, \pi_{\text{ref}}(\cdot \mid x, y_{i,<t}, M_i)\big)\big]$
  else
    $\mathcal{K}_{\text{ref}} \leftarrow 0$ *// every run in this paper: no reference model is loaded and no third forward pass is run*
  end if
  $\mathcal{L}_{\text{GRPO}} \leftarrow \mathcal{L}_{\text{Surr}} - \beta\, \mathcal{K}_{\text{ref}}$
  // Track B: dense student*, executed sequentially after Track A*
  if $\lambda > 0$ **then**
    Dense forward pass without any mask: $\pi_\theta(\cdot \mid x)$
    $\mathcal{L}_{\text{Transfer}} \leftarrow \mathbb{E}_{i,t}\big[D_{\text{KL}}\big(\text{sg}[\pi_\theta(\cdot \mid x, y_{i,<t}, M_i)] \,\|\, \pi_\theta(\cdot \mid x, y_{i,<t})\big)\big]$ *// masked pass is the TEACHER; chunk over t (Sec. E)*
  else
    $\mathcal{L}_{\text{Transfer}} \leftarrow 0$ *// the replay-only ablation: the second forward pass is not run, so it costs nothing*
  end if
  // Update: minimising this loss maximises the objective of Sec. 3.3
  $\mathcal{L}_{\text{total}} \leftarrow -\mathcal{L}_{\text{GRPO}} + \lambda\, \mathcal{L}_{\text{Transfer}}$
  $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\text{total}}$ *// AdamW, after global-norm clipping*
end while

property of efficiency work generally rather than of this method specifically, but it is worth stating plainly: we are lowering the cost of an optimisation procedure, not making its outcomes safer. A second, more specific consideration is that the method trains policies under a compressed context by construction. If the eviction policy systematically drops certain kinds of content, the resulting model may be worse on inputs that depend on it in ways the aggregate metrics in this paper would not reveal (Section I.2); auditing compressed-context models on distributional slices, rather than on averages alone, is a reasonable precaution.

A third consideration runs against the first. About one in five of the eviction-trained runs behind Table 5 and Appendix H (18 of 90) finished below the untrained checkpoint, in one case at 16.53 against a floor of 40.71. We have not identified the cause (Section I.1) and we have no detector that fires before a run is scored. The groups this section describes as newly able to afford these experiments are the least able to absorb a wasted one, so the cost we claim to lower should be read as including a per-run evaluation against the untrained checkpoint, and reruns when it fails.