SGRPO: OPTIMIZING STATE-SPACE LANGUAGE MODELS WITH STATE-ISOLATED GROUP-RELATIVE POLICY

Anonymous authors
Paper under double-blind review

ABSTRACT

Group-relative policy optimization (GRPO) assigns each sampled response an advantage by standardizing its reward against a group of rollouts generated from the same prompt. The derivation of this estimator requires the rollouts in a group to be independent samples from the current policy—a condition Transformer policies satisfy, since each rollout re-encodes the prompt and retains no state between generations. State-space policies such as Mamba break this condition: because they carry state recurrently, decoding a group's rollouts sequentially against one shared generation cache, an efficiency choice that avoids re-encoding the prompt, lets each layer's recurrent state and causal-convolution buffer carry forward from one rollout to the next. Each rollout after the first therefore begins not from the clean post-prompt state h_0 produced by the prompt alone, but from the state left behind by the preceding rollout. Under a non-degenerate reward map, this carry-over gives the estimator an order-dependent bias that, through the shared group baseline, reaches even the uncontaminated first rollout. The correction, state isolation, restores each layer's recurrent state and convolution buffer to their post-prompt values before every rollout, reinstating exact independence at zero additional FLOPs and leaving the group-relative objective unchanged. SGRPO packages this correction with token-level normalization and future-KL token weighting. The correction is worth adopting for its exactness and negligible cost alone, regardless of its effect on downstream accuracy.

1 INTRODUCTION

Reinforcement learning (RL) has become a central paradigm for post-training large language models (LLMs), driving substantial gains in mathematical reasoning, code generation, and other tasks whose outcomes can be verified automatically (Shao et al., 2024; DeepSeek-AI, 2025). Among the algorithms used in this setting, group-relative policy optimization (GRPO) (Shao et al., 2024) has emerged as a de facto standard: rather than training a separate value network, it samples a group of rollouts for each prompt and scores every rollout against the statistics of its own group. The resulting policy is simple, critic-free, and memory-efficient, and a line of subsequent work has since sharpened it along three largely orthogonal axes: loss normalization and sampling filtering (Yu et al., 2025), clipping behavior under off-policy updates (Xi et al., 2026), and token-level credit assignment (Ma et al., 2026). This line of work refines *what* the rollouts are scored with and further it leaves untouched the question of *how* the rollouts themselves are generated, which is where the present paper intervenes. In parallel, selective state-space models (SSMs) such as Mamba (Gu & Dao, 2024; Dao & Gu, 2024) have become a serious alternative to attention, and hybrid SSM–attention models now reach competitive accuracy at substantially lower inference cost (Lieber et al., 2024; NVIDIA, 2025). The two trends are converging, and group-relative RL is increasingly applied to policies whose backbone is partly or entirely recurrent.

However, every method in the GRPO family was designed and validated on Transformer policies, and its advantage estimator relies on an assumption that Transformers satisfy by construction: the G rollouts of a group must be independent and identically distributed (i.i.d.) draws from $\pi_\theta(\cdot \mid q)$. A Transformer re-encodes the prompt into a fresh key–value cache for every generation, so no rollout can influence another. An SSM, in contrast, compresses the prompt into a fixed-size recurrent

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

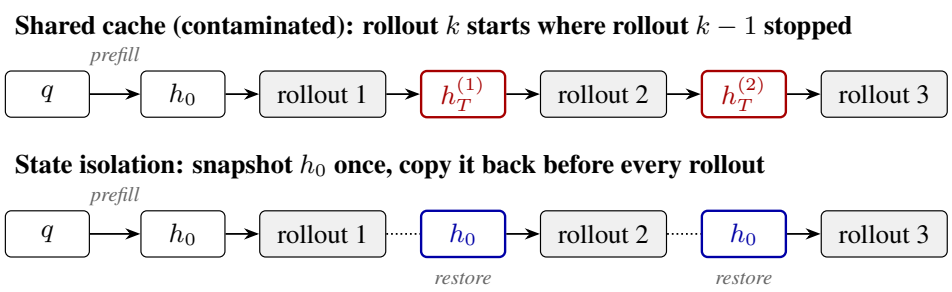


Figure 1: Shared-cache generation versus state isolation for one group ($G = 3$ shown). h_0 denotes the full post-prompt cache (recurrent state and convolution buffer of every layer). The prompt is encoded once in both cases; isolation adds only the copies.

state that is updated in place at every decoded token. When the rollouts of a group are decoded sequentially against one shared cache—the natural choice whenever the prompt should be encoded only once or memory rules out decoding the whole group in parallel—each rollout begins from the terminal state of its predecessor rather than from the post-prompt state h_0 (Figure 1). The group then ceases to be a sample and becomes a Markov chain, and the order in which rollouts are generated enters the advantages. Nothing in the loss or in the training logs signals the failure. On `mamba-130m-hf` we find that, across three seeds, the state from which the second rollout starts retains a cosine similarity of only 0.641 ± 0.008 to h_0, and that contaminated rollouts terminate early roughly four times as often as clean ones.

We propose *state isolation*, which snapshots the recurrent state and the convolution buffer of every layer once after the prompt has been encoded, and copies the snapshot back into the cache before each rollout. Because the restore is an exact tensor copy, every rollout starts from the same h_0 and the group is again an i.i.d. sample, without approximation, without additional forward passes, and without any change to the objective that consumes the rollouts. Through an empirical and theoretical analysis, we show that shared-cache generation displaces the starting state at every position in the group and alters generation behavior, and that under a readout-injectivity assumption this displacement is exactly what breaks the i.i.d. condition and injects an order-indexed shift into every advantage through the group baseline. Building on the correction, **we define** SGRPO, a critic-free objective that computes group-relative advantages from isolated rollouts, normalizes the loss at the token level following DAPO, and weights tokens by FIPO's future-KL signal. In summary, the principal contributions of our work include:

- **Diagnosis.** A formal characterization of hidden-state contamination in group-relative RL on SSM policies, identifying when shared-cache generation violates the i.i.d. assumption and how the resulting shift propagates to every advantage (Section 4);
- **Evidence.** Direct multi-seed measurements of state drift and of a behavioral side effect on Mamba, together with a length sweep and a comparison showing that residual-stream activations are not a valid proxy for the recurrent state (Section 4);
- **Correction.** State isolation, an exact fix with an audited set of restored cache fields and a measured cost of 2.81 MiB and under 3 ms per snapshot–restore pair, and SGRPO, the objective that builds on it (Sections 5 and 6).

2 Related Work

Policy optimization for language models. Nearly all RL post-training of LLMs builds on the policy-gradient theorem (Sutton et al., 1999). TRPO (Schulman et al., 2015) stabilized policy-gradient updates by constraining each step to a KL trust region, and PPO (Schulman et al., 2017) later replaced this constraint with a clipped surrogate objective that remains the workhorse of the field, typically paired with a learned critic and generalized advantage estimation (Schulman et al., 2016). GRPO (Shao et al., 2024) dispensed with the critic altogether by normalizing each rollout's reward against the statistics of a group sampled for the same prompt, and this estimator has since

been adopted at scale (DeepSeek-AI, 2025). A growing body of work refines individual components of this recipe. DAPO (Yu et al., 2025) replaces the sequence-level loss average with token-level normalization, filters groups whose rewards are all identical, and removes the KL penalty. BAPO (Xi et al., 2026) adapts the clipping bounds so that positive- and negative-advantage tokens contribute in balance during off-policy training, while FIPO (Ma et al., 2026) reweights tokens by a discounted future-KL signal to obtain denser credit assignment. What these methods share is a focus on how a set of collected rollouts is converted into a gradient. They take for granted that the rollouts themselves form a valid i.i.d. sample, an assumption that holds automatically for the Transformer policies on which they were developed, and it is precisely this assumption that we revisit for recurrent policies.

State-space language models. Structured state-space models (Gu et al., 2020; 2022) emerged as an alternative to attention for long sequences, processing their input through a linear recurrence over a fixed-size state and thereby decoding autoregressively with constant memory per token. Mamba (Gu & Dao, 2024) made this recurrence input-dependent through a selective scan, which allowed SSMs to match Transformers on language modeling, and Mamba-2 (Dao & Gu, 2024) subsequently connected the two families through structured state-space duality. Building on these advances, hybrid architectures such as Jamba (Lieber et al., 2024) and Nemotron-H (NVIDIA, 2025) interleave SSM and attention layers, retaining Transformer-level accuracy while substantially reducing inference cost. The efficiency of all these models rests on the same property: the fixed-size state summarizes everything decoded so far and is updated in place rather than recomputed. That property, however, is exactly what allows information to leak from one rollout into the next when a generation cache is shared, and the consequences of this leakage for group-relative RL have so far gone unexamined, even as such training is increasingly applied to recurrent and hybrid policies.

3 PRELIMINARIES

Group-relative advantage estimation. Given a prompt q, GRPO samples a group of rollouts $o_1, \dots, o_G$ from the current policy, scores each with a verifiable reward $r_i = r(o_i)$, and assigns the advantage

$$A_i = \frac{r_i - \bar{r}}{\mathrm{std}(r_{1:G}) + \epsilon}, \qquad \bar{r} = \frac{1}{G}\sum_{j=1}^{G} r_j. \tag{1}$$

The group mean $\bar{r}$ serves as a Monte Carlo estimate of the value $\mathbb{E}_{o\sim\pi_\theta(\cdot|q)}[r(o)]$, and this substitution is justified only when the o_i are i.i.d. draws from $\pi_\theta(\cdot \mid q)$.

Selective state-space models. Each Mamba layer maintains a recurrent state that evolves as

$$h_t = \bar{A}_t\, h_{t-1} + \bar{B}_t\, x_t, \qquad \bar{A}_t = \exp(\Delta_t A), \tag{2}$$

where the step size $\Delta_t > 0$ and the input matrix $\bar{B}_t$ depend on the current input (Gu & Dao, 2024). Because A is diagonal with negative entries, every $\bar{A}_t$ has entries in $(0, 1)$, so past inputs are discounted geometrically. The scan is preceded by a short causal convolution whose most recent $d_{\text{conv}} - 1$ inputs are also cached. We write h_0 for the complete post-prompt cache, that is, the recurrent state and the convolution buffer of every layer after the prompt has been processed. The distribution of a completion then depends on the cache from which decoding starts, $\pi_\theta(o \mid q, h) = \prod_t \pi_\theta\big(y_t \mid \phi_\theta(h, y_{<t})\big)$, where ϕ_θ denotes the deterministic state update.

Decoding a group. There are two ways to decode the G rollouts of a group. If they are decoded in parallel as G rows of a batch, each row holds its own copy of h_0 and the rollouts are independent by construction. If they are instead decoded one after another against a single cache object, as happens when memory rules out a G-wide batch, when a prefilled prompt state is reused to avoid re-encoding, or when the cache is passed by reference rather than copied, the state left by one rollout becomes the starting state of the next. We refer to the latter regime as *shared-cache generation*. We do not claim that any particular public framework adopts it. Our analysis applies whenever it occurs.

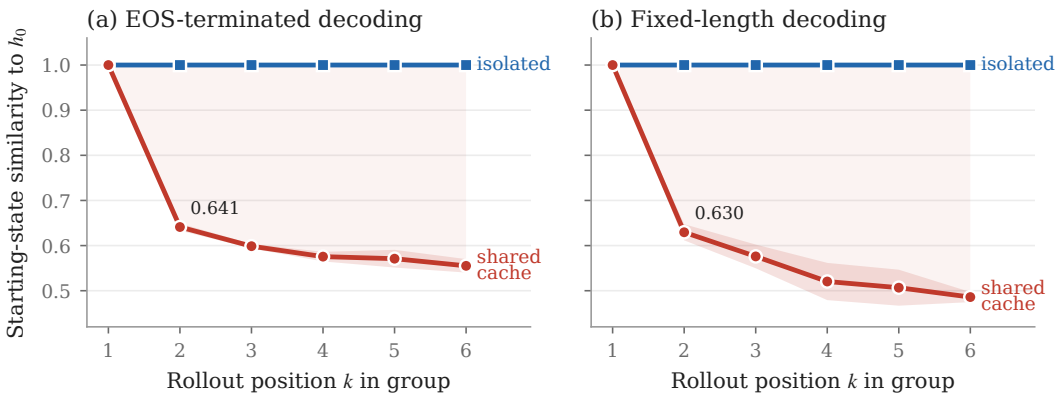


Figure 2: Starting-state similarity to h_0 by position in the group ($G = 6$, $T = 64$, 10 prompts, fp32) under (a) EOS-terminated and (b) fixed-length decoding. Lines show means over three seeds and shaded bands the sample standard deviation across seeds. The isolated arm equals 1 at every position in every seed.

4 Motivation: Hidden-State Contamination in Group Rollouts

This section establishes the problem before any correction is introduced. We first measure, directly in the cache tensors of `mamba-130m-hf`, how far shared-cache generation moves the starting state of each rollout and whether this movement changes generation behavior (Section 4.1). We then state formally why such movement biases the group-relative estimator (Section 4.2).

4.1 Empirical evidence

Experimental setup. All measurements are taken on `mamba-130m-hf` with prompts drawn from the GSM8K (Cobbe et al., 2021) test split, using ancestral sampling at temperature 1 on a single 6 GB laptop GPU. For each prompt we decode a group of $G = 6$ rollouts in two arms that run identical decoding code and differ in a single respect: the shared-cache arm decodes the group sequentially against one cache, whereas the isolated arm restores h_0 before every rollout (Section 5). Contamination is read directly from the cache rather than from model outputs. Our primary measure, the *starting-state similarity*, is the cosine similarity between the `recurrent_states` tensor from which rollout k starts and the one from which rollout 1 starts, which by construction is h_0, averaged over the 24 layers and over prompts. Because layer-level values within a prompt are strongly correlated, we summarize uncertainty by the sample standard deviation across seeds rather than by significance tests over individual layers. The stopping rule also requires care, since drift accumulates with every decoded token and therefore depends on how long each rollout runs. We consider two settings. Under *EOS-terminated decoding*, the standard setting in RL training, a rollout ends as soon as it emits the end-of-sequence (EOS) token, while under *fixed-length decoding* EOS is ignored and every rollout runs for exactly T tokens. We treat EOS-terminated decoding as the primary setting, as it matches training practice and yields the higher, and hence more conservative, similarity.

Shared-cache generation displaces the starting state. As shown in Figure 2, the state from which the second rollout starts retains a similarity of only 0.641 ± 0.008 to h_0 under EOS-terminated decoding, and the similarity continues to fall, in progressively smaller steps, to 0.555 ± 0.015 at the sixth rollout. Averaged over positions 2–6, it is 0.588 ± 0.005 under EOS-terminated decoding and 0.544 ± 0.027 under fixed-length decoding, so the effect does not hinge on how EOS is handled. The diminishing step sizes are consistent with the geometric discounting of Eq. (2), under which each rollout largely overwrites the history left by its predecessors while adding a further displacement of its own, although we offer this as an interpretation rather than a fitted model.

Contamination changes what the policy generates. The displaced state does not merely move the recurrent state; it alters the generated text. Under EOS-terminated decoding, shared-cache rollouts stop before the 64-token cap in 18.9% of cases, compared with 4.4% for isolated rollouts (Ta-

Seed	Shared cache, early EOS	Isolated, early EOS	Odds ratio	Fisher p
42	15/60 (25.0%)	5/60 (8.3%)	3.67	0.026
123	6/60 (10.0%)	1/60 (1.7%)	6.56	0.114
456	13/60 (21.7%)	2/60 (3.3%)	8.02	0.004
Pooled	34/180 (18.9%)	8/180 (4.4%)	5.01	2.4×10^{-5}

Table 1: Rollouts ending on EOS before the $T = 64$ cap under EOS-terminated decoding (10 prompts, $G = 6$).

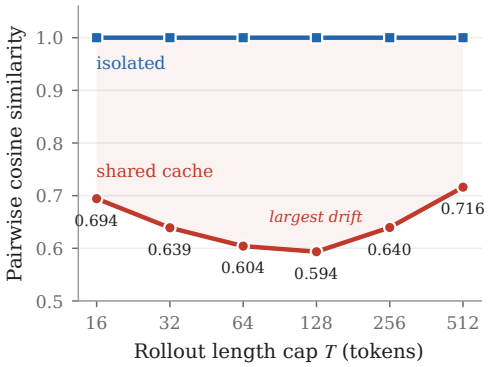


Figure 3: Rollout-length sweep over caps $T \in \{16, \ldots, 512\}$ (single seed). The measure is the mean *pairwise* cosine similarity among the six starting states of a group, averaged over layers and prompts, which differs from the similarity to h_0 in Figure 2. Decoding is EOS-terminated in bf16, so T is an upper bound on rollout length rather than an exact length.

ble 1; 180 rollouts per arm, odds ratio 5.0, Fisher exact $p = 2.4 \times 10^{-5}$), and their mean length falls from 62.7 to 53.8 tokens. The direction is consistent across all three seeds, although seed 123 alone does not reach significance ($p = 0.11$) and the pooled test does not stratify by seed. Because rollout length and final answers are coupled, this behavioral shift is a plausible route by which contamination reaches the reward, although we did not measure the reward shift directly in this experiment.

The effect persists across rollout lengths. Since $\bar{A}_t$ is contractive, one might expect longer rollouts to erase the initial condition and thereby weaken the effect. Figure 3 shows that contamination is present at every length cap we tested, but that its magnitude is not monotone in T: the mean pairwise similarity is 0.694 at $T = 16$, reaches its minimum of 0.594 at $T = 128$, and recovers to 0.716 at $T = 512$. A plausible explanation is that longer rollouts push the state further from h_0 while forgetting simultaneously pulls it toward a state determined by the most recent tokens, but we have not tested this account. The sweep uses a single seed, and because rollout lengths were not logged, a large cap does not guarantee a long rollout; repeating the $T = 64$ setting on an earlier revision of our code with the same seed gave 0.618 rather than 0.604, which indicates the scale of run-to-run variation independent of the seed.

Residual-stream activations are not a valid proxy. Setting `output_hidden_states=True` in HuggingFace Transformers (Wolf et al., 2020) returns residual-stream activations ($d_{\text{model}} = 768$ per token) rather than the recurrent SSM state (1536×16 per layer), and it is tempting to measure contamination on the former. We initially expected this proxy to hide the drift; a side-by-side comparison on one seed shows the opposite (Appendix C). Under contamination, the residual-stream similarity falls further than the recurrent-state similarity (0.29 versus 0.58, averaged over positions 2–6), while both remain exactly 1 under isolation. The proxy is therefore invalid because it measures a structurally different quantity, not because it conceals the effect, and in this comparison it overstates the divergence.

4.2 From State Drift to Biased Advantages

Under shared-cache generation, rollout 1 is drawn from $\pi_\theta(\cdot \mid q, h_0)$ and each later rollout from $\pi_\theta(\cdot \mid q, h^{(k)})$, where $h^{(k)}$ is the cache left by rollout $k-1$ and is therefore a deterministic function of $(h_0, o_{1:k-1})$. Let $\mu_\theta(h) := \mathbb{E}_{o \sim \pi_\theta(\cdot \mid q,h)}[r(o)]$ be the reward mean from a starting cache h, and let $\delta_k := \mu_\theta(h^{(k)}) - \mu_\theta(h_0)$ be the resulting order-indexed shift, with $\delta_1 \equiv 0$. Two assumptions are needed, and neither follows from the architecture alone.

Assumption 1 (Readout injectivity)**.** *On caches reachable from h_0, distinct caches induce distinct completion distributions $\pi_\theta(\cdot \mid q, h)$.*

Assumption 2 (Reward sensitivity)**.** *At the parameters θ under consideration, $\Pr(\delta_k \neq 0) > 0$ for each $k \geq 2$.*

Assumption 1 is not implied by the invertibility of $\bar{A}_t$, but Lemma 1 (Appendix A) shows that its failure would be non-generic for a real-analytic readout. Assumption 2 fails only when the reward is insensitive to the starting state, as in degenerate groups whose rollouts all receive the same reward, which DAPO's dynamic-sampling filter already discards.

Theorem 1 (Contamination breaks the i.i.d. assumption)**.** *Fix θ and assume shared-cache generation. (i) Under Assumption 1, the rollouts are i.i.d. draws from $\pi_\theta(\cdot \mid q, h_0)$ if and only if $h^{(k)} = h_0$ almost surely for every k. (ii) $\mathbb{E}[r_k \mid o_{1:k-1}] = \mu_\theta(h_0) + \delta_k$, and the numerator of every advantage in Eq.* (1) *decomposes as $r_i - \bar{r} = (\delta_i - \bar{\delta}) + (\eta_i - \bar{\eta})$ with $\mathbb{E}[\eta_i \mid o_{1:i-1}] = 0$. Under Assumption 2, the order-indexed terms $\delta_i - \bar{\delta}$ are not almost surely all zero, and they reach A_1 through $\bar{\delta}$ even though $\delta_1 = 0$.*

The proof is given in Appendix A. The condition $h^{(k)} \neq h_0$ in part (i) is not derived but measured, and Figure 2 shows that it holds at every position and in every seed. Taken together, the theorem and the measurements say that under shared-cache generation the rollouts of a group are no longer a sample from the policy being optimized: each rollout's conditional reward mean depends on its position and on the rollouts that preceded it, and through the shared baseline this dependence reaches every advantage, including that of the first rollout. The theorem does not fix the sign of $\mathbb{E}[\delta_k]$, and Appendix A further quantifies the error against the clean-state reward mean and discusses why the shift does not average out over training.

5 Methodology

5.1 State isolation

The analysis of Section 4.2 identifies the starting cache of each rollout as the sole source of contamination, which suggests a correction that operates on the cache itself and leaves the objective untouched. State isolation encodes the prompt once, stores a detached snapshot of the resulting cache h_0, and copies this snapshot back into the cache, in place, before each rollout is decoded (Algorithm 1, lines 6–10). Because the restore is an exact tensor copy, every rollout starts from exactly h_0, so the rollouts of a group are i.i.d. draws from $\pi_\theta(\cdot \mid q, h_0)$ and $\delta_k \equiv 0$ for every k; unlike Theorem 1, this guarantee requires no assumption on the readout or on the reward. Both components of every layer's cache must be restored. The recurrent state carries the long-range history, whereas the convolution buffer holds the last $d_{\text{conv}} - 1$ inputs to the short convolution, so that restoring the recurrent state alone would still condition the first few tokens of rollout k on the tail of rollout $k-1$.

A restore that omits a single carried field would leave a channel for contamination open. We therefore audited every field of the per-layer cache in the pinned library version (`transformers` 5.13.0 (Wolf et al., 2020); Appendix B) and found that only `recurrent_states` and `conv_states` carry information across decoding steps; the remaining fields are allocation metadata or initialization flags that are set once during prompt encoding and never reset. Isolation is also distinct from batching across groups: in the batched implementation we measure, the batch dimension indexes independent groups while the rollouts within each group are still decoded against a shared cache, so batching of this kind leaves the contamination intact. For hybrid SSM–attention models, our implementation additionally truncates the attention key–value cache to the prompt length, and for pure Transformers it reduces to a no-op; the hybrid path is implemented but has not been measured.

5.2 The SGRPO objective

SGRPO applies a critic-free clipped surrogate (Schulman et al., 2017) to isolated rollouts. It computes group-relative advantages A_i^{ISO} (Eq. (1)) from isolated rewards, normalizes the loss over all real tokens in the group following DAPO (Yu et al., 2025), and weights each token by the future-KL signal of FIPO (Ma et al., 2026). With importance ratio $\rho_{i,t} = \pi_\theta(y_{i,t})/\pi_{\theta_{\text{old}}}(y_{i,t})$ and padding

Algorithm 1: SGRPO

Input: Initialized SSM policy π_θ, training dataset $\mathcal{D}$, reward function R, group size G, maximum rollout length T, number of inner epochs μ, clipping range ε, future-KL decay γ and weight bounds $[1, w_{\max}]$

for *step* $s = 1 \dots S$ **do**
  Procedure Sample with state isolation;
    Update the old policy $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$;
    Sample the s-th batch $\mathcal{D}_s$ from $\mathcal{D}$;
    for *each prompt* $q \in \mathcal{D}_s$ **do**
      Encode q once to obtain the post-prompt cache h_0;
      Snapshot $\hat{h} \leftarrow h_0$ (recurrent states and convolution buffers, all layers);
      for $k = 1 \dots G$ **do**
        Restore the cache in place: $h \leftarrow \hat{h}$;
        Sample $o_k \sim \pi_{\theta_{\text{old}}}(\cdot \mid q, h)$ for at most T tokens and store $\log \pi_{\theta_{\text{old}}}(o_k)$;
      end
      Compute $r_k = R(o_k)$; **if** $\mathrm{std}(r_{1:G}) = 0$ **then** discard the group;
      Compute A_k^{ISO} for each o_k by Eq. (1);
    end
  Procedure Update the policy π_θ;
    for *epoch* $= 1 \dots \mu$ **do**
      Compute the future-KL weights $w_{i,t}$ without gradient;
      Update π_θ by maximizing $\mathcal{J}^{\text{SGRPO}}(\theta)$ in Eq. (3);
    end
end

mask $M_{i,t}$, the objective is

$$\mathcal{J}^{\text{SGRPO}}(\theta) = \frac{1}{\sum_{i,t} M_{i,t}} \sum_{i,t} M_{i,t} \, \min\Big(\rho_{i,t}\, w_{i,t} A_i^{\text{ISO}},\ \mathrm{clip}(\rho_{i,t}, 1-\varepsilon, 1+\varepsilon)\, w_{i,t} A_i^{\text{ISO}}\Big), \qquad (3)$$

where $w_{i,t} = \mathrm{clip}\left(\exp(\sum_{k \geq t} \gamma^{k-t} M_{i,k}(\log \pi_\theta(y_{i,k}) - \log \pi_{\theta_{\text{old}}}(y_{i,k}))), 1, 1.2\right)$ is computed without gradient, with $\gamma = 2^{-1/30}$ and $\varepsilon = 0.2$. Since $\theta = \theta_{\text{old}}$ on the first pass over a freshly collected batch, $w \equiv 1$ at that point, so the future-KL weighting becomes active only from the second inner epoch onward. The only new component of SGRPO is state isolation; the objective is included so that the correction is attached to a concrete, reproducible training procedure, and its empirical evaluation is left to future work (Section 7).

6 Experiments

We verify that state isolation removes the contamination exactly, measure its cost, and report a preliminary training-time comparison. All settings follow Section 4.1.

Exactness. In the isolated arm of Figure 2, the starting-state similarity to h_0 equals 1 at every position, under both decoding settings and with zero variance across seeds. This outcome is guaranteed by construction (Section 5.1), so the measurement serves to verify the implementation, where it would expose an omitted cache field or an aliased tensor. In an independent single-seed run, the individual isolated values averaged $1 - 1.7 \times 10^{-9}$ with a standard deviation of 7.6×10^{-8}, which reflects floating-point rounding in the cosine computation.

Cost. Isolation adds only memory copies and no floating-point operations, and the prompt is still encoded once per group. For `mamba-130m-hf` (24 layers, $d_{\text{inner}} = 1536$, $d_{\text{state}} = 16$, $d_{\text{conv}} = 4$, fp32), one snapshot occupies $24 \times 1536 \times (16 + 4) \times 4 = 2{,}949{,}120$ bytes, or 2.81 MiB, which matches the measured increase in peak memory exactly, with zero variance over 500 measurements. Over 500 timed calls after warm-up on an RTX 4050 Laptop GPU (single seed), a snapshot takes

1.98 ± 1.59 ms (median 1.50) and a restore 0.87 ± 0.68 ms (median 0.65), so a snapshot–restore pair costs 2.85 ms and a group requires one snapshot and at most G restores (Appendix C).

Preliminary training diagnostics (single seed). We also ran one short training comparison between shared-cache and isolated rollouts, whose results are suggestive rather than established (100 steps, $G = 4$, $T = 64$, learning rate 10^{-6}, 50 prompts, EOS-terminated decoding). Because the untrained model never produced a correct final answer under the binary reward, every group was degenerate and no update occurred, so this run alone uses a diagnostic graded reward, $0.7 \cdot$exact match$+0.3 \cdot$(fraction of the reference rationale's intermediate values present in the output). Per-coordinate gradient variance, estimated on a fixed random subset of 262,144 coordinates, was 3.98×10^{-6} under shared-cache generation and 2.36×10^{-7} under isolation, a ratio of $16.8\times$. This comparison is weak: only 4 of the 24 shared-cache groups were non-degenerate, all from a single prompt, whereas the isolated estimate averages three prompts (11 samples; per-prompt 1.2, 4.0, and 1.9×10^{-7}). Gradient norms were 17.1 ± 17.8 and 6.7 ± 1.1, respectively. Over the final 20 training steps, the mean graded reward was 0.0067 with a shared cache and 0.0111 with isolation, a ratio of $1.66\times$ between two very small values, and 74 versus 62 of the 100 steps were skipped as degenerate. We did not detect order dependence in the rewards themselves, as the correlation between rollout position and reward was -0.05 and $+0.07$ in the two arms; rewards this sparse cannot resolve the shifts δ_k of Theorem 1. The evidentiary status of every result in the paper, including which results rest on a single seed, is summarized in Appendix D.

7 Limitations and Future Work

Limitations. Our study was conducted under a limited compute budget, a single 6 GB GPU, and its scope reflects that constraint. All measurements concern one 130M-parameter pure SSM with rollouts of at most 512 tokens on GSM8K prompts, whereas GRPO-family methods are typically applied to models with billions of parameters and far longer rollouts. Theorem 1 does not depend on scale, but the magnitude of δ_k does, and we have not measured it at scale or on hybrid architectures. For the same reason we could not verify the theoretical conditions over long training horizons: the sign-consistency condition for persistence (Appendix A) requires tracking δ_k throughout a full training run, and readout injectivity (Assumption 1) is supported only by a genericity argument. We report no downstream accuracy, and our only training-time evidence comes from a single seed with a diagnostic reward. The rollout-length sweep, the residual-stream comparison, and the timing measurements are likewise single-seed. Finally, the practical reach of the failure depends on how often shared-cache generation occurs, and we have not surveyed which RL frameworks, if any, decode SSM groups in this way; our correctness results hold regardless.

Future work. With additional compute, we plan to evaluate SGRPO against GRPO, DAPO, and BAPO across multiple seeds on GSM8K and MATH (Hendrycks et al., 2021), to extend the measurements to larger and hybrid SSM–attention models, and to log δ_k over complete training runs in order to test the persistence condition directly. We also intend to examine how current RL frameworks decode recurrent policies, to reseed every single-seed result reported here, and to adopt FIPO's dual-clip token mask, which our simplified future-KL weighting omits.

8 Conclusion

In this paper, we examine what happens to group-relative policy optimization when the policy is a state-space model and the rollouts of a group are decoded against a shared generation cache. Through direct measurements on Mamba, we show that every rollout after the first starts from a state displaced from the clean post-prompt state, at every position in the group, in every seed, and at every rollout length we tested, and that the displacement alters generation behavior, with contaminated rollouts terminating early roughly four times as often. We then establish formally that this displacement breaks the i.i.d. assumption on which the group-relative advantage estimator rests, and that the resulting order-indexed shift reaches every advantage in the group through the shared baseline. Building on this analysis, we propose state isolation, which restores the recurrent state and convolution buffer of every layer to their post-prompt values before each rollout, eliminating the contamination exactly at a measured cost of 2.81 MiB and a few milliseconds per group, and we

define SGRPO, which trains on the isolated rollouts with token-level normalization and future-KL weighting. In practical terms, any group-relative RL pipeline for SSM or hybrid policies should either decode each group in parallel from independent copies of the post-prompt cache or restore that cache before every rollout. We hope this work draws attention to architecture-dependent assumptions in RL post-training and helps extend group-relative methods reliably beyond Transformer policies.

AI USE STATEMENT

In this work we used generative AI tools (large language model assistants) for the following tasks, each reviewed and verified by the authors: *implementing methods* (drafting and editing the training, rollout, and measurement code); *formulating mathematical claims and writing proofs* (drafting and revising Theorem 1, Lemma 1, and Propositions 1 and 2, all checked line by line by the authors); *designing research methodology* (discussion of experiment design, including the comparison of decoding settings, and writing the algorithm description); *interpreting results* (statistical re-analysis of logged results, such as per-seed and pooled Fisher tests); *literature analysis and reference formatting* (verifying titles, authors, and identifiers of cited works); and *figure creation and paper drafting*. Every reported number is computed by code from result files produced on the authors' hardware. The authors take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

All experiments use the public `state-spaces/mamba-130m-hf` checkpoint and the GSM8K test split, with `torch` 2.6.0 and `transformers` 5.13.0. Seeds, group sizes, rollout lengths, decoding settings, and precision are stated with each experiment, and the state-isolation procedure is given in full in Algorithm 1 and Appendix B. We will release the code for all measurements, together with the logged results and the script that regenerates every figure.

REFERENCES

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

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645:633–638, 2025. arXiv:2501.12948.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling (COLM)*, 2024.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent memory with optimal polynomial projections. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations (ICLR)*, 2022.

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

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida,

Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Chiyu Ma, Shuo Yang, Kexin Huang, Jinda Lu, Haoming Meng, Shangshang Wang, Bolin Ding, Soroush Vosoughi, Guoyin Wang, and Jingren Zhou. FIPO: Eliciting deep reasoning with future-KL influenced policy optimization. *arXiv preprint arXiv:2603.19835*, 2026.

NVIDIA. Nemotron-H: A family of accurate and efficient hybrid Mamba-Transformer models. *arXiv preprint arXiv:2504.03624*, 2025.

John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

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, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.

Zhiheng Xi, Xin Guo, Yang Nan, Enyu Zhou, Junrui Shen, Wenxiang Chen, Jiaqi Liu, Jixuan Huang, Zhihao Zhang, Honglin Guo, Xun Deng, Zhikai Lei, Miao Zheng, Guoteng Wang, Shuo Zhang, Peng Sun, Rui Zheng, Hang Yan, Tao Gui, Qi Zhang, and Xuanjing Huang. BAPO: Stabilizing off-policy reinforcement learning for LLMs via balanced policy optimization with adaptive clipping. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2510.18927.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

A Theoretical details

Throughout this appendix we use the notation of Section 4.2: $h^{(k)}$ is the cache from which rollout k starts, $\mu_\theta(h)$ is the reward mean from starting cache h, and $\delta_k = \mu_\theta(h^{(k)}) - \mu_\theta(h_0)$.

A.1 Proof of Theorem 1

Proof. (i) If $h^{(k)} = h_0$ almost surely for all k, each o_k is drawn from $\pi_\theta(\cdot \mid q, h_0)$ with fresh sampling randomness, so the rollouts are i.i.d. . Conversely, suppose $\Pr(h^{(k)} \neq h_0) > 0$ for some k. By Assumption 1, on this event the conditional law of o_k given $o_{1:k-1}$, namely $\pi_\theta(\cdot \mid q, h^{(k)})$, differs

from $\pi_\theta(\cdot \mid q, h_0)$. This is incompatible with o_k being independent of $o_{1:k-1}$ with law $\pi_\theta(\cdot \mid q, h_0)$, since independence would require the conditional law to equal $\pi_\theta(\cdot \mid q, h_0)$ almost surely. (ii) Since $h^{(k)}$ is determined by $o_{1:k-1}$, conditioning gives $\mathbb{E}[r_k \mid o_{1:k-1}] = \mu_\theta(h^{(k)}) = \mu_\theta(h_0)+\delta_k$. Defining $\eta_i := r_i - \mu_\theta(h^{(i)})$, we have $\mathbb{E}[\eta_i \mid o_{1:i-1}] = 0$ and $r_i = \mu_\theta(h_0) + \delta_i + \eta_i$. Substituting into Eq. (1), the common term $\mu_\theta(h_0)$ cancels in $r_i - \bar{r}$, leaving $(\delta_i - \bar{\delta}) + (\eta_i - \bar{\eta})$; the denominator $\mathrm{std}(r_{1:G})$ is computed from the same contaminated rewards. Under Assumption 2, some δ_k with $k \geq 2$ is nonzero with positive probability, so the vector $(\delta_i - \bar{\delta})_{i=1}^G$ is not almost surely zero: if $\bar{\delta} = 0$, then $\delta_k - \bar{\delta} = \delta_k$ for that k, and otherwise $\delta_1 - \bar{\delta} = -\bar{\delta} \neq 0$, which is the term through which the shift enters A_1. □

A.2 Readout injectivity is generic

Lemma 1 (Generic injectivity of the readout)**.** *Fix the prompt q and let $\mathcal{H} \subseteq \mathbb{R}^d$ be a connected open set of caches. Suppose that for some token v the log-probability $f(h) := \log \pi_\theta(v \mid q, h)$ is real-analytic on $\mathcal{H}$ and not constant. Then the set of pairs $Z = \{(h, h') \in \mathcal{H} \times \mathcal{H} : \pi_\theta(\cdot \mid q, h) = \pi_\theta(\cdot \mid q, h')\}$ has Lebesgue measure zero in $\mathcal{H} \times \mathcal{H}$.*

Proof. Let $g(h, h') = f(h) - f(h')$, which is real-analytic on the connected open set $\mathcal{H} \times \mathcal{H}$. Because f is not constant there are h_1, h_2 with $f(h_1) \neq f(h_2)$, so $g \not\equiv 0$. The zero set of a real-analytic function that is not identically zero on a connected open set has Lebesgue measure zero, a consequence of the identity theorem. Equal next-token distributions imply equal log-probabilities for v, so $Z \subseteq \{g = 0\}$ and Z is null. □

We work with a log-probability rather than raw logits because the softmax is invariant to a constant shift of the logits, so two different logit vectors can give the same distribution. The regularity condition holds at the level of the architecture: linear maps, SiLU, softplus, the exponential, RMS normalization with $\epsilon > 0$, and log-softmax are all real-analytic, so the map from the starting cache to the first-token log-probabilities is real-analytic. The lemma has two limits. Reachable caches may lie on a lower-dimensional subset of $\mathcal{H}$, on which an ambient null set carries no information, and the lemma idealizes away finite-precision arithmetic. It therefore does not prove Assumption 1 for the trained weights; it shows that a failure of the assumption would require a non-generic coincidence.

A.3 Error against the clean-state reward mean

Let $\sigma^2(h) := \mathrm{Var}(r \mid h)$ and compare each rollout's reward with the target $\mu_\theta(h_0)$ that the group mean is meant to estimate.

Proposition 1 (Mean squared error gap)**.** *For $k \geq 2$, $\mathrm{MSE}(r_k^{\mathrm{shared}}; \mu_\theta(h_0)) - \mathrm{MSE}(r_k^{\mathrm{iso}}; \mu_\theta(h_0)) = \mathbb{E}[\delta_k^2] + \big(\mathbb{E}[\sigma^2(h^{(k)})] - \sigma^2(h_0)\big)$. If reward noise is homoscedastic, the gap equals $\mathbb{E}[\delta_k^2]$, which is strictly positive under Assumption 2.*

Proof. Let $\mu_0 = \mu_\theta(h_0)$. Conditioning on the starting cache, $\mathbb{E}[(r_k - \mu_0)^2 \mid h^{(k)}] = \sigma^2(h^{(k)}) + \delta_k^2$, so the shared-cache MSE is $\mathbb{E}[\sigma^2(h^{(k)})] + \mathbb{E}[\delta_k^2]$. Under isolation $h^{(k)} = h_0$ and the MSE is $\sigma^2(h_0)$. Subtracting gives the stated gap, and under homoscedasticity the variance terms cancel. □

The homoscedastic case does not cover the binary rewards common in verifiable-reward RL. For a Bernoulli reward, $\sigma^2(h) = \mu(h)(1 - \mu(h))$, and writing $\mu(h^{(k)}) = \mu_0 + \delta_k$ gives $\sigma^2(h^{(k)}) = \mu_0(1 - \mu_0) + \delta_k(1 - 2\mu_0) - \delta_k^2$, so the gap reduces to $\mathbb{E}[\delta_k](1 - 2\mu_0)$, whose sign depends on the direction of the shift. For binary rewards, the harm of contamination therefore lies in the loss of the i.i.d. property established by Theorem 1, not necessarily in a larger error of each individual reward.

A.4 Persistence across training

Corollary 1 (Persistence)**.** *If Assumptions 1 and 2 hold at every step of a run, Theorem 1(ii) applies at every step. The pairing that generates δ_k (rollout k inherits the terminal cache of rollout $k-1$) is fixed by the sampling loop and depends neither on θ nor on the step index, so nothing in the procedure forces $\delta_k(\theta_t)$ to change sign across steps and thereby average out.*

Proposition 2 (No-cancellation condition)**.** *Let $\mathcal{T}$ be the set of training steps at which Assumptions 1 and 2 hold, and for $k \geq 2$ let $\bar{\delta}_k := |\mathcal{T}|^{-1} \sum_{t \in \mathcal{T}} \mathbb{E}[\delta_k(\theta_t)]$. If $\mathbb{E}[\delta_k(\theta_t)]$ is nonzero and has the same sign for every $t \in \mathcal{T}$, then $\bar{\delta}_k \neq 0$.*

The implication is immediate; its value lies in giving a checkable sufficient condition for Corollary 1. Corollary 1 is a structural argument rather than a convergence result, and we have not verified the sign condition, which would require logging δ_k throughout a full training run.

B Implementation details of state isolation

A restore is complete only if it covers every cache field that carries information from one decoding step to the next. We audited the per-layer cache of `mamba-130m-hf` in `transformers` 5.13.0, where it is implemented by `LinearAttentionCacheLayerMixin`. Table 2 lists every field of that class. Only the recurrent state and the convolution buffer carry information across steps, and state isolation snapshots and restores both; the remaining fields are fixed at allocation or set once during prompt encoding. Because cache classes differ across library versions, the audit must be repeated whenever the library is upgraded.

Field	Kind	Carries state across rollouts?
`recurrent_states`	tensor $[B, 1536, 16]$	Yes; snapshotted and restored
`conv_states`	tensor $[B, 1536, 4]$	Yes; snapshotted and restored
`is_recurrent_states_initialized`	Boolean	No; set once at prompt encoding
`is_conv_states_initialized`	Boolean	No; set once at prompt encoding
`dtype`, `device`, `batch_size`, `conv_kernel_size`	metadata	No; fixed at allocation

Table 2: Fields of the per-layer Mamba cache in `transformers` 5.13.0.

C Additional results

Starting-state similarity by position. Table 3 gives the values underlying Figure 2, and Table 4 gives the average over positions 2–6 for each seed. Under EOS-terminated decoding, the mean rollout length was 51.0, 59.1, and 51.3 tokens in the shared-cache arm and 61.5, 63.8, and 62.9 tokens in the isolated arm, for seeds 42, 123, and 456.

Position k	1	2	3	4	5	6
EOS-terminated, shared cache	1.000	$0.641 \pm .008$	$0.599 \pm .005$	$0.576 \pm .011$	$0.571 \pm .020$	$0.555 \pm .015$
Fixed-length, shared cache	1.000	$0.630 \pm .018$	$0.576 \pm .026$	$0.520 \pm .041$	$0.507 \pm .040$	$0.486 \pm .011$
Either setting, isolated	1.000	1.000	1.000	1.000	1.000	1.000

Table 3: Starting-state similarity to h_0 (mean $\pm$ sample standard deviation over three seeds; $G = 6$, $T = 64$, 10 prompts).

Decoding	Seed 42	Seed 123	Seed 456	Mean $\pm$ SD
EOS-terminated	0.5825	0.5919	0.5904	0.5883 ± 0.0050
Fixed-length	0.5221	0.5737	0.5354	0.5437 ± 0.0268

Table 4: Shared-cache similarity to h_0 averaged over positions 2–6, per seed. The isolated arm equals 1.000 in every cell.

Residual stream versus recurrent state. Table 5 compares the two measurements in the shared-cache arm (single seed, 10 prompts, $T \leq 64$, bf16). Both rows equal 1.000 at every position under isolation.

Rollout-length sweep. Table 6 lists the values underlying Figure 3 (single seed). The second row reports the spread of the 3,600 layer-by-pair values, which are correlated within a prompt; it

Position k	1	2	3	4	5	6
Recurrent state (`recurrent_states`)	1.000	0.647	0.612	0.570	0.555	0.526
Residual stream (`hidden_states`)	1.000	0.355	0.375	0.268	0.191	0.245

Table 5: Similarity to the rollout-1 value under each measurement, shared-cache arm.

describes dispersion rather than uncertainty in the mean. The isolated arm equals $1 - 10^{-9}$ at every T.

Length cap T	16	32	64	128	256	512
Mean pairwise similarity	0.694	0.639	0.604	0.594	0.640	0.716
Standard deviation	0.192	0.207	0.223	0.217	0.269	0.278

Table 6: Shared-cache pairwise starting-state similarity by rollout-length cap.

Cost. Table 7 reports 500 timed calls (50 repeats for each of 10 prompts, fp32, single seed) after five discarded warm-up calls per prompt, which remove a one-time allocation cost incurred when a new cache is first touched.

	Mean	SD	Median	Min	Max
Snapshot (ms)	1.979	1.594	1.499	0.674	14.956
Restore (ms)	0.870	0.681	0.653	0.304	5.148
Peak memory increase (MiB)	2.8125	0	–	–	–

Table 7: Cost of state isolation on an RTX 4050 Laptop GPU.

D EVIDENTIARY STATUS OF RESULTS

Table 8 summarizes the strength of the evidence behind each result. We distinguish results that follow from the mechanism and were confirmed by measurement, results replicated over three seeds, results from a single seed that require replication, properties verified by inspecting the implementation, and assumptions that remain unverified.

Result	Evidence
Shared cache displaces the starting state at every $k \geq 2$ (Fig. 2)	Three seeds
Displacement holds under both decoding settings	Three seeds
Contamination raises early termination (Tab. 1)	Three seeds (pooled test)
Contamination at every $T \in [16, 512]$, non-monotone in T (Fig. 3)	Single seed
Residual stream is not a valid proxy (Tab. 5)	Single seed
Isolation restores h_0 exactly	By construction; measured
Restored fields are the complete carried state (Tab. 2)	Implementation audit
Batching across groups leaves within-group contamination intact	Implementation audit
Memory cost of 2.81 MiB per snapshot	By construction; measured
Wall-clock cost of snapshot and restore (Tab. 7)	Single seed
Gradient-variance and reward differences in training (Sec. 6)	Single seed; suggestive
Readout injectivity (Asm. 1)	Assumed; failure non-generic
Sign consistency of δ_k during training (Prop. 2)	Not verified
Effect of isolation on downstream accuracy	Not evaluated

Table 8: Evidence behind each result reported in the paper.