KV-STREAMS FOR EFFICIENT COMPACTION IN AGENTIC REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Scaling the horizon of agentic LLMs is bottlenecked by the need to fit ever longer context traces in GPU memory. Context compaction has been the most popular mechanism to alleviate this issue, keeping GPU memory constant for a given trace. Unfortunately, most compaction strategies rely on prefilling the LLM context many times over, hindering training throughput. To alleviate this bottleneck and enable efficient trainable compaction, we propose *KV-streams*, a plug-and-play strategy compatible with *any* compaction strategy that substantially increases throughput while showing no evidence of hindering performance. KV-streams enable scalable compaction by streaming the KV cache forward rather than flushing it after each compaction. We show that KV-streams enable three different compaction strategies, achieving a 2.6 to $5\times$ wall-clock speedup in training. Beyond efficiency, we find that the streamed KV cache can act as a recurrent state, carrying forward information that has long since disappeared from the context. Specifically, in a controlled setting we show that, contrary to prior work, RL alone is all that is needed for this behavior to emerge. Overall, we show KV-streams to be an efficient and lightweight plug-and-play addition to any post-training pipeline.

1 INTRODUCTION

Reinforcement learning on long-horizon agentic tasks requires generating many concurrent rollouts, each having key-value (KV) caches growing linearly with length. Under a fixed GPU memory budget, the longest-running rollouts quickly become the bottleneck, monopolizing memory that shorter rollouts could otherwise use. Reducing the memory footprint of long traces can therefore substantially increase training throughput, enabling more optimizer steps and larger batch sizes (Kwon et al., 2023; Chen et al., 2026).

To stop memory from growing, a variety of *compaction* techniques can be used. They remove a portion of the context once a fixed token budget is exceeded (e.g. replacing older context with a summary) (Wu et al., 2026b; Li et al., 2026b). Compaction lets a trace extend indefinitely (in principle) without growing its memory footprint, but increases training compute. This is due to the retained context being prefilled each time following a compaction, so the model performs repeated forward passes over the same tokens (Cim et al., 2026).

In this work, we propose *KV-streams* to alleviate the computational bottleneck of compaction. Specifically, KV-streams evicts KV entries directly in the inference engine yielding the memory benefits of compaction while maintaining a single continuous stream of KVs, thus avoiding any repeated prefill cost. On the trainer side, KV-streams reproduce the eviction pattern with a custom attention mask, so no tokens are re-processed. Through this simple plugin, KV-streams substantially increase both training and inference throughput on the four compaction strategies we tested, and are compatible with others. Figure 1 summarizes those four strategies and the mechanism behind the speedup.

We evaluate KV-streams combined with various compaction strategies on long-horizon agentic tasks, using `Qwen3-4B-Instruct-2507` and `Qwen3.5-4B`, obtaining a $5\times$ speedup on TextWorld (Côté et al., 2019) and a $1.3\times$ speedup on ALFWorld (Shridhar et al., 2021). Interestingly, while early in training both KV-streams and re-prefill compaction can hinder performance compared to the use of full context, with sufficient training, KV-streams can close this gap and even surpass the performance of full context. Further, we find that models trained with KV-streams also

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.

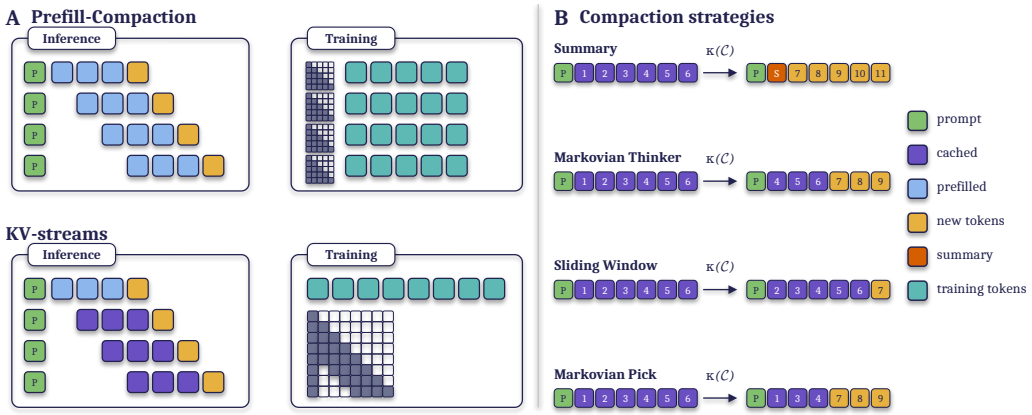


Figure 1: Compaction strategies and their training costs. **(A)** re-prefill compaction recomputes retained tokens and creates separate training traces. KV-streams preserves retained KVs and trains on a continuous trace with an attention mask that reproduces eviction. **(B)** context transformations under each strategy (P: prompt; S: summary).

generalize better to unseen text-based games (Cui et al., 2025; Hausknecht et al., 2020; Wang et al., 2022; Jansen et al., 2024). We then carry these results over to agentic software-engineering tasks, where KV-streams reach peak performance $3\times$ faster than full context while maintaining comparable performance.

Prior work (Kontonis et al., 2026) showed that preserving KVs across compactions lets the KV cache act as a "pseudo-recurrent" state, but found that this behaviour requires a preliminary SFT stage in their training setup. We test whether that requirement holds in our approach. Empirically, the additional wall-clock time spent on SFT does not improve performance, and synthetic experiments show that pseudo-recurrence can emerge in the KV cache without this preliminary SFT stage. This is a significant improvement for RL post training pipelines, with up to a $5\times$ speedup.

Overall, KV-streams is a simple plug-and-play addition to any compaction strategy that greatly reduces training wall-clock, matches or exceeds the performance of re-prefill compaction, generalizes effectively, and requires no expensive SFT stage for initialization. We make our inference and training code public at `https://anonymous.4open.science/r/anonymous-kv-eviction-8877/README.md` for a subset of the experiments.

2 Compaction

Long-running LLM agents commonly manage context growth by summarizing earlier interactions or offloading information to external memory, keeping the active context within a fixed budget (Packer et al., 2024; Wu et al., 2026b; Lu et al., 2025). It is necessary to scale LLM context windows by preventing “context rot” and limiting growing KV-memory. During training, limiting these effects is quite important. For instance, a single long rollout can monopolize the inference server’s memory, throttling throughput. On top of that, agentic settings are specially affected by this hindrance as the environment’s feedback adds to the growing context. For instance, in software engineering, reading or writing a file can consume a substantial amount of context that may not be necessary at later stages of the task. Reducing this context is therefore important for scaling long-context agentic RL. While prior works have attempted to alleviate these issues Aghajohari et al., 2026; Wu et al., 2026a they either focus on non-agentic settings, which simplify the problem, they do not study its interaction with RL, or most detrimentally, they re-prefill the context incurring a high compute cost. Our goal is to fill these gaps and eliminate any redundant compute cost.

To illustrate where this added cost comes from, we formalize compaction and show how KV-streams can be applied to any strategy that meets this definition. We note that our experiments are in agentic settings, which require *turn*-wise compaction (see Appendix F), but we use a generalized entry-level view to describe the process, which generalizes to tokens/turns. KV-streams is readily amenable to other long-context use cases as well.

A compaction operator κ maps a context $\mathcal{C}$ to a shorter one $\mathcal{C}' = \kappa(\mathcal{C})$. It fires whenever a trigger T is met, either a fixed entry budget B that caps $|\mathcal{C}|$ or the model's own decision to compact (Li et al., 2026a). When triggered, κ composes two operators, $\kappa = \text{gen} \circ \text{del}$. The *delete* operator removes a span of at least $b \geq 1$ tokens, so the context always shrinks, and *generate* emits a bridge s in its place, possibly empty, such as a summary (Li et al., 2026b; Wu et al., 2026b).

Prefill-compaction. Typical compaction pipelines continue the compacted context in a fresh trace, which resets the kept tokens' cached KVs along with their positional encodings. Concretely, a retained token that held position i and cache (k_i, v_i) in $\mathcal{C}$ is re-emitted into the new trace $\mathcal{C}'$ with reset positional encodings. Recomputing it under this new position and context gives a different cache entry, $(k'_i, v'_i) \neq (k_i, v_i)$, so both its position and its cached KV change. This reset is what forces the kept entries to be prefilled again rather than reused.

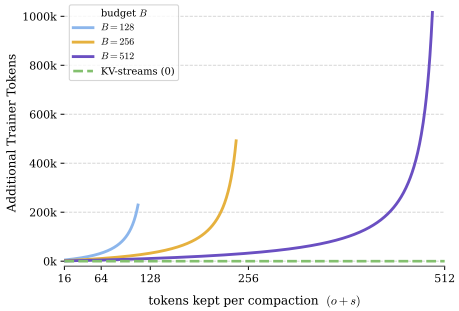


Figure 2: Extra tokens prefilled over a 32k-token rollout vs. tokens kept per compaction $o+s$ (overlap o plus summary s), for memory budgets B. As the retained tokens approach the budget, the repeated prefill grows by a factor of $\frac{N}{B-(o+s)}$, the number of times each kept token is re-processed by the trainer. KV-streams prefills nothing again (green, at 0).

This strategy incurs a heavy cost. Let r be the number of entries a compaction keeps, the retained overlap plus any generated summary. All r of them are prefilled into the fresh trace even though their KVs were already computed once (Cim et al., 2026). Under a token/turn budget B, the context holds r entries right after a compaction, so the model can generate $b = B - r$ new entries before it hits the budget and compacts again. A rollout of N entries therefore compacts N/b times, and each of those compactions prefills r entries a second time. Over the whole rollout the extra prefilled entries add up to

$$\Delta_{\text{re-prefill}} = \frac{\overbrace{N}^{\text{rollout entries}} \cdot \overbrace{r}^{\text{kept per compaction}}}{\underbrace{B - r}_{\text{new entries between compactions}}},$$

which diverges as the retained context approaches the budget, derived in App. H). In practice this can be mitigated by producing fewer compactions or reducing the number of entries kept per compaction (e.g. replacing all turns with a summary), yet this only mitigates the cost and does not remove it. Figure 2 illustrates this cost at a granular token level, showing how compacting more often makes the training grow exponentially with increased number of compactions.

KV-streams. To alleviate these issues we introduce KV-streams, a drop-in modification that skips the repeated prefill entirely. Rather than prefilling the kept tokens again, KV-streams carries the retained cache forward: the retained overlap and the summary s keep their existing entries (k_i, v_i) instead of being recomputed as (k'_i, v'_i), so $\Delta_{\text{re-prefill}} = 0$. In the trainer, a custom attention mask blocks attention to the deleted span, replicating this eviction in one forward pass. Figure 1 visualizes the mechanisms for KV-streams, showing how by directly evicting the KV-cache a compacted rollout avoids re-prefilling and does not require to be broken into multiple traces.

3 Compaction Strategies

KV-streams alleviates the cost of prefilling context, but different compaction strategies retain different amounts of context $r = o + s$. We study four that span this range, from keeping almost none of it to keeping nearly the full budget.

Summary. The standard strategy asks the model to summarize its prior context and then drops it (Wu et al., 2026b; Li et al., 2026b). It keeps no entries ($o = 0$) and replaces them with a summary of variable length s up to a preset limit.

Markovian Thinker. The Markovian thinker (Aghajohari et al., 2026) keeps the prefill cost down by deleting half the context at each compaction and replacing nothing, so $o = B/2$ and $s = 0$.

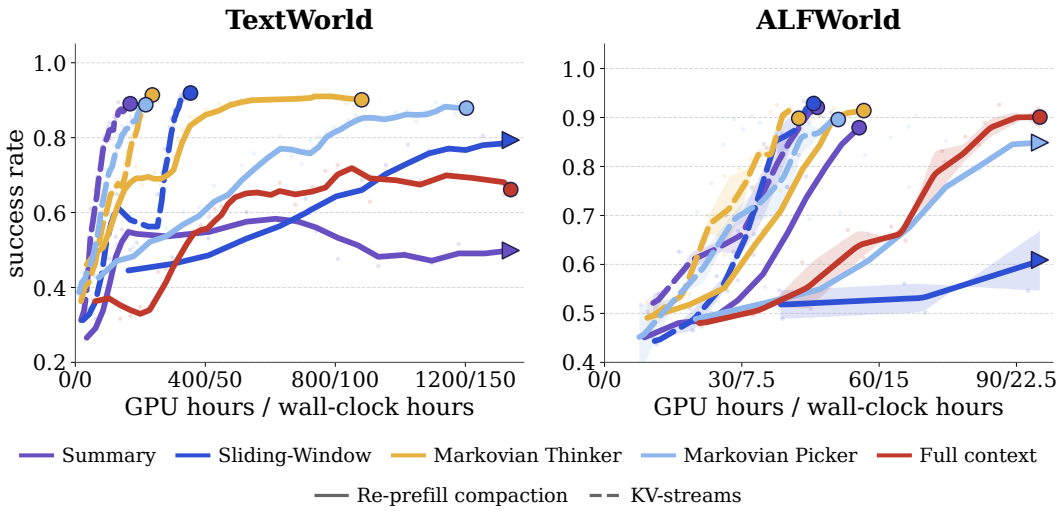


Figure 3: KV-streams reaches comparable or better success with fewer GPU hours on TextWorld (left) and ALFWorld (right). Axis ticks give GPU hours and wall-clock hours (8 GPUs on TextWorld, 4 on ALFWorld). Solid and dashed curves denote re-prefill compaction and KV-streams, with full context as a reference. Arrowheads mark truncation after exceeding the runtime of full-context completion. We provide complete curves appear in Figure 8.

Markovian Pick. It's possible that keeping the most recent half is not the most efficient strategy, rather a more efficient mechanism is to let the model pick what turns to remove and which to keep. Thus, at each compaction the model chooses b entries to retain deleting all others.

Sliding Window. To probe the high-retention regime where the repeated prefill is most expensive (Figure 2), we use sliding-window compaction (Xiao et al., 2024), which keeps almost the full window (o close to B, $s = 0$). It retains the most context and so incurs the largest prefill cost, exactly where KV-streams helps most.

4 Experimental Setup

Benchmarks and protocol. We evaluate KV-streams on two text-based games and on software-engineering tasks. Both settings test whether an agent can carry information forward once older context is compacted. Text-based games often require recalling details received early in the episode, so compaction can hurt performance if the agent fails to retain them. Software-engineering tasks involve reading and writing long files, and removing these files from context can discard information the agent later relies on. The text-based games use `Qwen3-4B-Instruct-2507` and the software-engineering tasks use `Qwen3.5-4B`. Across benchmarks we sample 8 rollouts per prompt and use a learning rate of 10^{-6}. Within each benchmark, all runs share the same optimization settings and differ only in the compaction strategy. We report evaluation score against cumulative GPU-hours, and we stop a run early only when its wall-clock time exceeds that of the full-context run.

TextWorld (Côté et al., 2019) provides procedurally generated games across four task families (coin_collector, cooking, simple, treasure_hunter) at three difficulty levels. We generate 60,000 synthetic games split uniformly across task families and difficulties, and retain 256 representative tasks for evaluation. Compaction triggers every 10 turns, and episodes run for up to 200 turns within a 32k-token budget. We train for 500 gradient steps with a batch size of 512 on an 8-GPU H100 node (4 inference, 4 trainer), and report mean `avg@1` on the evaluation tasks.

ALFWorld (Shridhar et al., 2021) is a household environment in which the agent completes a language-specified task by issuing text commands and receives the admissible commands at each step. Compaction triggers every 10 turns, and episodes run for up to 100 turns within a 16k-token budget. We train on the standard split for up to 200 gradient steps with a batch size of 128 on 4 A100 GPUs (1 inference, 3 trainer). Every 20 gradient steps we evaluate one rollout per game on the 70 seen and 67 unseen validation games, reporting the mean success rate over the two splits.

For software engineering we train on 1,028 SWE-rebench (Badertdinov et al., 2026) and 443 ScaleSWE (Zhao et al., 2026) tasks. We remove tasks that the base model solves on every at-

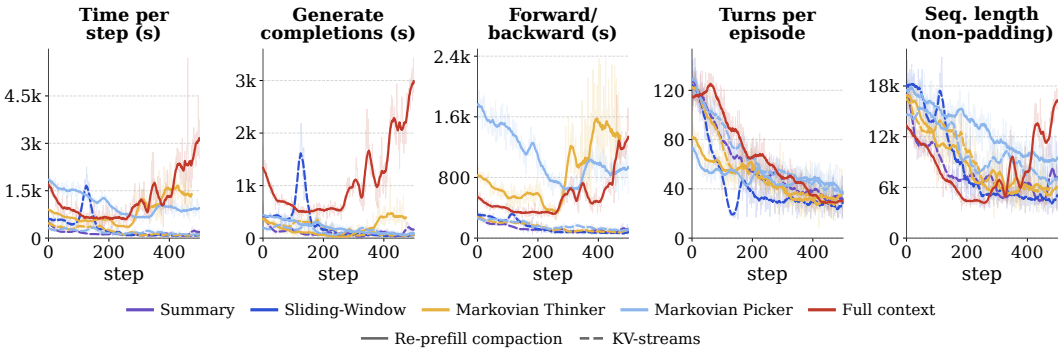


Figure 4: Smoothed per-step timing and rollout statistics on TextWorld for runs that do not exceed the full-context wall-time. KV-streams (dashed) reduces generation and forward/backward time relative to re-prefill compaction (solid). Rollout statistics show mean turns per episode and sequence length. We find KV-streams to be substantially faster during forward/backward and generation.

tempt and tasks it never solves, measured by pass@3, so that every remaining task carries a training signal. Compaction triggers every 30 turns, and episodes run for up to 100 turns within a 64k-token budget. We compare the two strongest KV-streams configurations, Markovian Thinker and Sliding Window, against full context. All methods train with a batch size of 256 on an 8-GPU GB200 node, and we evaluate on SWE-bench Verified (OpenAI, 2024).

Due to limited compute (as some experiments can take over a week to execute), we only run multi-seed experiments for AlfWorld, running single seed experiments for both software-engineering tasks and TextWorld. Appendix B lists the concurrency settings and hyperparameters for each benchmark.

Inference engine implementation. Our aim is to show the speedup at as close to production scale as possible, so we build on asynchronous RL frameworks that decouple the trainer and inference GPUs. For the text-based games, we run a forked vLLM inference engine and a forked prime-RL (Team et al., 2025) trainer[1], both modified to support KV-streams. Our vLLM configuration uses 16-token KV-cache blocks (Kwon et al., 2023). Only complete blocks are added to the prefix cache.[2] When a request does not end on a multiple of 16, the trailing tokens stay uncommitted until a later request completes the group. This breaks the stream, since the next request may evict tokens that the uncommitted tokens would have attended to, and we initially found it to substantially hinder performance. We fix it by padding each completion to a multiple of 16 tokens. The padding must be prefilled by both the inference engine and the trainer, which inflates the total token count. We account for this in Figure 3 and describe it in Appendix E. Despite this inefficiency, KV-streams still improves throughput. On the trainer side we use asynchronous RL with a maximum off-policy lag of 3, and we modify prime-RL to apply in-flight weight updates *without* flushing the KV cache (Piché et al., 2026). This change is important for scaling KV-streams, since flushing on every weight update would force a prefill of entire, possibly long, traces and undo the memory savings. For the software-engineering experiments, we instead use a forked Slime trainer with a forked SGLang inference engine, which commits individual tokens with a block size of 1 and therefore needs no padding.

5 Experimental Results

5.1 Text-Based Games

KV-streams reduces experiment wall-time. Figure 3 shows success rate against GPU-hours on TextWorld and ALFWorld for every compaction strategy under re-prefill compaction and under KV-streams. On TextWorld, KV-streams reaches the final performance of re-prefill compaction with between 3.7× and 11.3× fewer GPU-hours, while performing better or similarly. Re-prefill compaction is also expensive in absolute terms. Summary and Sliding-Window with re-prefill compaction consume more GPU-hours than training with full context, even though full context plateaus at a success rate of 0.64. KV-streams removes this overhead, so compaction becomes cheaper than full context as well as more accurate. On ALFWorld the traces are shorter, thus the

[1]Forked from `https://github.com/PrimeIntellect-ai/prime-rl`. Anonymized code: `https://anonymous.4open.science/r/anonymous-kv-eviction-8877/README.md`.

[2]`https://docs.vllm.ai/en/v0.9.1/design/v1/prefix_caching.html`

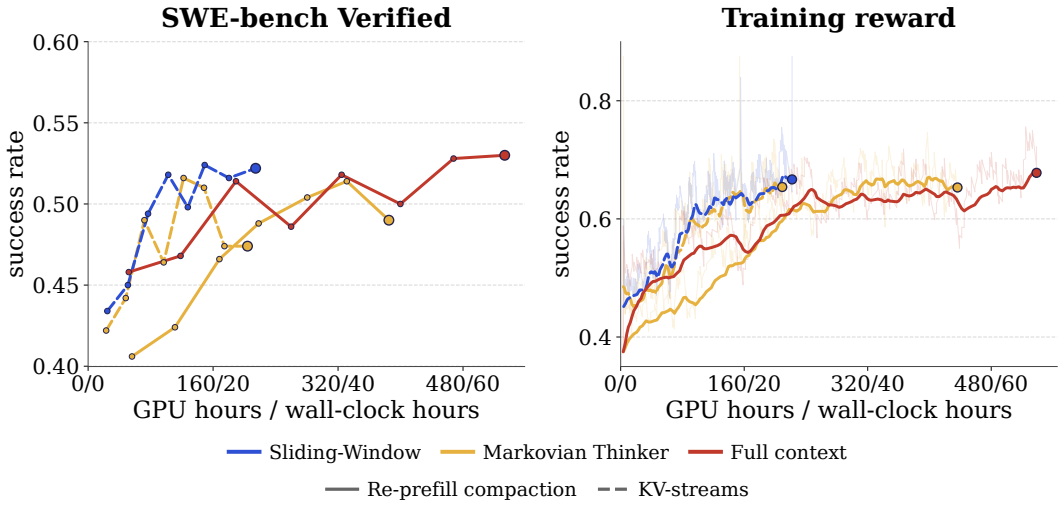


Figure 5: Comparison of KV-streams against full context and re-prefill Markovian Thinker on software-engineering tasks. **Left:** SWE-bench Verified success rate at each evaluation checkpoint. **Right:** training reward, smoothed with an exponential moving average over 8 gradient steps for every run (raw trace faint). Axis ticks give GPU hours and wall-clock hours on 8 GPUs, and every run is shown up to 160 gradient steps. KV-streams reaches peak performance about $3\times$ faster than full context and re-prefill compaction.

re-prefilling overhead is smaller. Regardless KV-streams still achieves an increased throughput between $1.3\times$ and $3.0\times$. KV-streams again reaches the final success rate of re-prefill compaction without incurring a performance loss across the four strategies. The full training curves for these runs, including the ones truncated in Figure 3, are in Appendix C.

Where does the speedup come from? Figure 4 breaks the time per step into generation and forward/backward. As expected, the bottleneck for re-prefill compaction is the forward and backward pass, where a large retained context r forces repeated recomputation. KV-streams instead reduces time in both the forward/backward pass and generation. While we find that initially re-prefill methods in TextWorld lead to shorter episodes and thus faster sampling, this gain diminishes as training progresses and is erased by the added training time.

Performance transfer. We aim to evaluate whether the speedups from KV-streams come at the cost of worse transfer to other tasks. We evaluate on unseen text-based games (Cui et al., 2025; Hausknecht et al., 2020; Wang et al., 2022; Jansen et al., 2024) using the final checkpoint from the *TextWorld* runs, with the same budget and context settings as training ($B = 10$ turns and a 32k-token context limit). Figure 6A reports the score for each game averaged over three seeds. For every game, a version of KV-streams performs on par with the best strategy. A version of KV-streams also improves upon the base model in every game, showing that learning on TextWorld transfers to other text-based games. No single compaction strategy performs best across all games (Figure 6A), consistent with broader findings that memory-method performance depends on the task (Wang et al., 2026). Regardless, we find no loss in transfer for KV-streams compared to full context and re-prefill compaction.

5.2 Software Engineering Agents

KV-streams scales to software-engineering tasks. Using the training set described in Section 4, we train for 160 gradient steps and evaluate on SWE-bench Verified, a standard software-engineering benchmark of 500 Python bug fixes. Figure 5 shows the evaluation results (left) and the training reward (right). Full context reaches the highest peak, 53.0% against 52.4% for Sliding-Window and 51.6% for Markovian Thinker with KV-streams. The gap is under one point for Sliding-Window and 1.4 points for Markovian Thinker, being within margin of error. Our results therefore remain consistent with the text-based games, finding no significant degradation in performance compared to full context. As before, KV-streams substantially reduces wall-clock time, providing a 3x speedup, reaching peak performance in 20 hours compared to more than 60 hours for full context. The gap

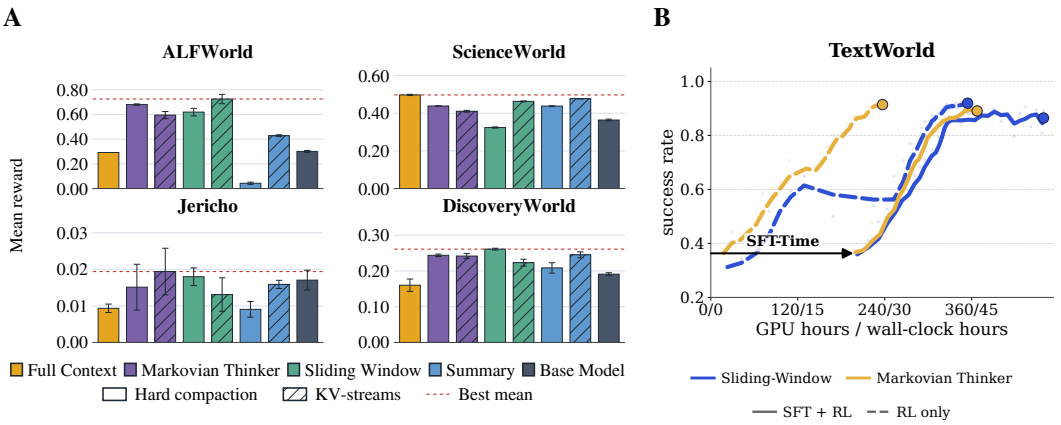


Figure 6: (A) Transfer of TextWorld-trained checkpoints to other environments. Bars show mean reward and whiskers show the standard deviation across evaluation seeds. (B) An SFT warm-start does not pay for itself once its cost is counted. TextWorld success rate against GPU hours, as in Figure 3, for Sliding-Window and Markovian Thinker with KV-streams, with an SFT warm-start followed by RL (SFT + RL, solid) and RL only (dashed). SFT + RL curves start after the GPU hours spent on SFT (SFT-Time arrow). The SFT loss is in Appendix D.

persists when comparing Markovian Thinker under re-prefill compaction and under KV-streams, where KV-streams reaches peak performance in about 16 hours versus about 40 hours for re-prefill.

6 Is SFT Necessary for Recurrent KVs?

Since retained KVs attended to tokens that are no longer in the context, prior work has shown they can act as a psudo-recurrent state (Kontonis et al., 2026), carrying forward information that has since been removed from the context. Because the model was not pretrained with this in mind, prior work relies on supervised fine-tuning to make the behaviour emerge as a first post training stage, before RL fine tuning can be done. We ask if this two-stage post training is necessary, and test whether KV-streams can enable direct RL fine tuning without an SFT stage. We test this by comparing cost savings of skipping an SFT warm start on TexWorld, and we analyze the mechanisms giving rise to this pseudo recurrent state in a controlled synthetic experiment

6.1 Prior SFT on TextWorld

If SFT were a necessary precursor for this recurrent behaviour, we would expect performance to be better with prior SFT. To analyze this and remove the confounder of SFTing from a better model, we collect 10k trajectories from `Qwen3-4B-Instruct-2507` with a 32k context limit (matching RL training), then perform SFT on those traces using randomly sampled eviction attention masks. For instance, sometimes we do sliding window with $B = 10$ and a stride equal to 1, other times we use Markovian Thinker truncation masking half the context ($B = N/2$).

Figure 6 B compares RL from the base model against an SFT warm-start followed by RL (Vattikonda et al., 2025). The model fits the full-context loss (Appendix D shows the SFT training curve), yet once SFT compute is counted, RL alone is more compute efficient and shows no observable drop in performance. We generally find that SFT does not justify it's cost heavily slowing down Markovian Thinker while when using sliding window both strategies reach peak performance in the same amount of time. These results suggest that prior SFT is not an important pre-requesite for enabling KV-streams, substantially simplifying the effort needed to enable it.

6.2 Recalling Evicted Content

The prior experiment shows empirically that KV-streams does not depend on prior SFT. Here we test this systematically with a controlled synthetic task. The model is first assigned an object (a fruit) and then asked to decode a sequence of numbers, with a budget of k tokens. Next we evict the turn containing the assignment and ask the model which object it was assigned, choosing from a list of candidates. After eviction the assignment no longer appears in the model's input, so the model cannot reference it and instructing the model to count numbers ensures it does not leak the object in its own

context. Thus, it can recover the object only if the tokens it decoded while the assignment was still visible encoded that information in their KV entries, which KV-streams carries forward. We vary the number of decoded tokens $k \in \{16, 32, 64, 128, 256, 512\}$ to test whether recall scales with the capacity of the KV-cache. Two further evaluations modify this base setting. The first asks whether a model trained to recall fruits generalizes to a new class of objects (actors). The second asks whether training can extend recall to an object the base model assigns zero probability to, by asking for a TV when the prompt lists only fruits as candidates.

RL enables information leakage through KVs. In the first task the model must retrieve the evicted assignment from a list of five candidate fruits, one of which is the assignment. Because the list always contains the answer, random sampling under RL produces some successes from the start, and the model can bootstrap from them. Figure 7A illustrates the setup and reports the results. As the KV-cache budget grows, recall rises to 100% and matches SFT. It becomes unreliable only at the smallest budgets of 16 and 32 tokens. KV-streams therefore learns to carry the assignment forward through RL alone, without an SFT stage. While this result is sufficient to show that information can be leaked through KVs alone, it is restricted in the setting that it could overfit to the specific use case making things less reliable, as well as the support between the model with evicted context and unevicted context always had overlap, i.e., the both full context and evicted had sufficient probability over the assigned fruit.

KV-streams generalizes across contexts. To test whether the information carried in the KV-cache is overfit to the training context, we modify the task. We take the model trained to recall fruits and ask it to recall an actor instead. Figure 7B shows the results. RL and SFT perform similarly at larger token budgets, and RL trails at smaller ones. Given a sufficient KV-token budget, KV-streams trained with SFT and with RL generalizes beyond the training domain.

KV-streams enables recall of objects with zero initial support. Finally, we test whether RL can teach the model to recall an object it would never sample once the assignment turn is evicted. We assign one of six objects at random, five fruits and a television, but at recall we list only the five fruits as options. The base model therefore places almost no probability on the television and would not sample it on its own. A model with full context, by contrast, can always recall that it was assigned the television. This lets us test whether information carried in the KV-cache can align the compacted model's distribution with the full-context one, bringing in a token that would otherwise never be sampled. SFT has no difficulty here by construction, since it forces the target response. Figure 7C shows that RL learns this behaviour as well, and as the KV-token budget grows it recalls the television 100% of the time. Even in this hardest case, RL alone aligns the compacted distribution with that of full context.

Overall, our empirical and synthetic experiments show that RL alone is enough for the model to carry information forward through the KV-cache, given sufficient KV-cache capacity, and that no expensive SFT stage is needed. This broadens the applicability of KV-streams. It becomes a plug-and-play mechanism that can be added at post-training without any additional pipeline stage.

7 Related Work

Many methods keep long-context generation affordable by capping how much each new token can cost. Some rewrite the model itself, as in recurrent and state-space architectures, which hold cost flat but only after the model is retrained to use them (Gu & Dao, 2024; Yang et al., 2025; Fu et al., 2025). Transformer-XL carries hidden states between segments (Dai et al., 2019), while Compressive Transformer compresses older memories for subsequent attention (Rae et al., 2020). GTrXL adapts Transformer-XL for stable reinforcement learning in memory-dependent environments (Parisotto et al., 2020).

Others leave the pretrained model untouched and simply hold the context to a fixed size, most often by summarizing old turns or keeping the most recent ones (Wu et al., 2026b; Li et al., 2026b). The second family is convenient but pays a hidden tax, since every time the budget is hit the window is torn down and rebuilt, and the same tokens are pushed through the model again (Cim et al., 2026; Aghajohari et al., 2026; Kontonis et al., 2026; Wu et al., 2026a). That repeated work is the prefill cost we study in Section 2, and it is what makes these runs jagged and starves the GPU. KV-streams targets this tax directly. We retain whatever the strategy chooses to keep, but rather than rebuild the window we drop the evicted keys and values inside the inference engine and mirror that drop in the

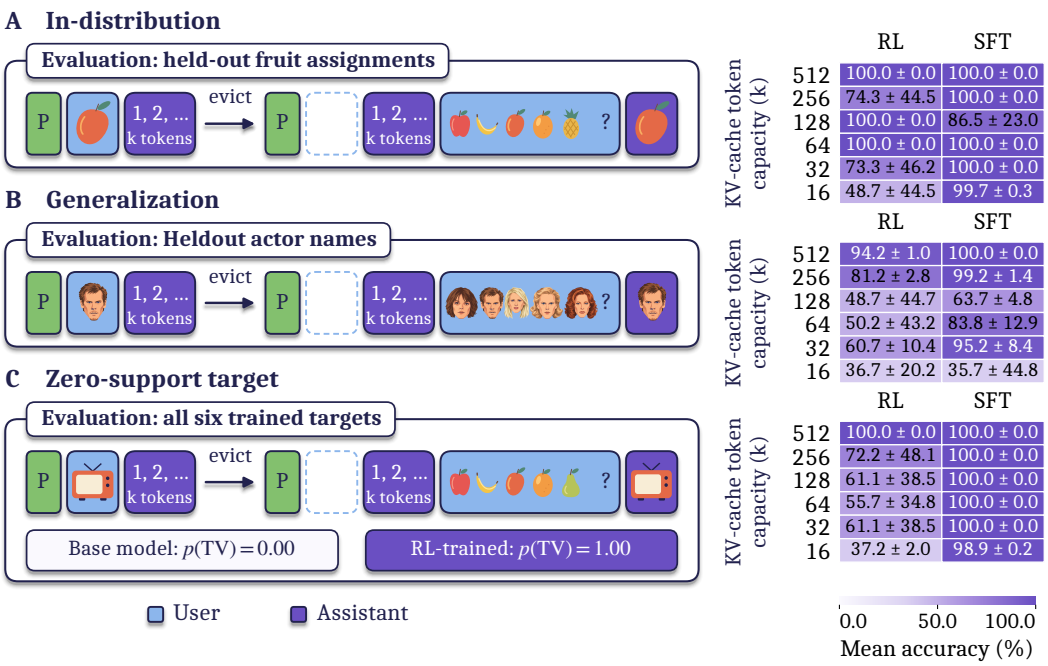


Figure 7: Post-eviction retrieval in-distribution (A), on new names (B), and with TV trained but unlisted (C). Heatmaps show RL and SFT accuracy (mean ± SD over three seeds) at the best learning rate per condition. Panel C covers all trained targets; its inset compares empirical TV-answer probabilities in the initial and final training batches of an RL run. Icons represent text targets.

trainer with an attention mask. The upshot is a smooth, steady cost per token that does not depend on which compaction rule is in play, close in spirit to schemes like attention sinks (Xiao et al., 2024) that sidestep rebuilding but without committing to one fixed retention pattern. Inference-time cache eviction is also studied by H_2O, which retains recent and high-attention tokens (Zhang et al., 2023), and TOVA, which interprets Transformers as multi-state RNNs and bounds their state through KV-cache compression (Oren et al., 2024). Our focus is on efficient agentic RL training that reproduces the inference-time eviction pattern.

A related line of work trains the model to manage its own context, so that the content surviving a compaction is itself learned, as a generated summary in some methods and as a compact memory state in others (Yan et al., 2026b;a; Zhou et al., 2026; Li et al., 2026b; Lu et al., 2025). These methods decide what to keep, and they all rebuild the window after each compaction, so the retained content is prefilled again and the training cost we study applies to them unchanged. KV-streams is orthogonal to this choice. It does not propose a new compaction rule and instead makes any rule cheaper to train, so these learned strategies could run on top of it.

AutoCompressors adapt pretrained language models to compress context into learned summary vectors used as soft prompts (Chevalier et al., 2023). For multi-turn agents, summarization-based context management can also be optimized jointly with tool use through RL (Lu et al., 2025).

8 Conclusion

We introduce KV-streams, a plug-and-play mechanism that is compatible with any compaction strategy. We show that by avoiding the repeated prefill cost, KV-streams substantially reduces experiment wall-time, providing up to a 3× speedup while often improving performance and at worst not hindering it. Further, we show that streamed KVs can act as a pseudo-recurrent state, carrying forward information that has long been removed from the context. Unlike prior work, we show that prior SFT is not a prerequisite for this effect to occur, which greatly simplifies the adoption of KV-streams. Overall, we show that KV-streams is a simple and effective way to substantially increase experiment throughput.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript drafting and editing, data analysis and plotting code, and the creation and refinement of figures, diagrams, and illustrative icons. We have reviewed all AI-assisted work and take responsibility for the final content, including text, claims, code, and visual artifacts.

REFERENCES

Milad Aghajohari, Kamran Chitsaz, Amirhossein Kazemnejad, Sarath Chandar, Alessandro Sordoni, Aaron Courville, and Siva Reddy. The markovian thinker: Architecture-agnostic linear scaling of reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=3As6AQ9ELI.

Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *Advances in Neural Information Processing Systems*, 38, 2026.

Kaiwen Chen, Xin Tan, Jingzong Li, and Hong Xu. Libra: Efficient resource management for agentic rl post-training, 2026. URL https://arxiv.org/abs/2606.03077.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Musa Cim, Burak Topcu, Chita Das, and Mahmut Taylan Kandemir. Parallel context compaction for long-horizon llm agent serving, 2026. URL https://arxiv.org/abs/2605.23296.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. In Tristan Cazenave, Abdallah Saffidine, and Nathan Sturtevant (eds.), *Computer Games*, pp. 41–75, Cham, 2019. Springer International Publishing. ISBN 978-3-030-24337-1.

Christopher Zhang Cui, Xingdi Yuan, Ziang Xiao, Prithviraj Ammanabrolu, and Marc-Alexandre Côté. Tales: Text adventure learning environment suite, 2025. URL https://arxiv.org/abs/2504.14128.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1285. URL https://aclanthology.org/P19-1285/.

Zichuan Fu, Wentao Song, Yejing Wang, Xian Wu, Yefeng Zheng, Yingying Zhang, Derong Xu, Xuetao Wei, Tong Xu, and Xiangyu Zhao. Sliding window attention training for efficient large language models, 2025. URL https://arxiv.org/abs/2502.18845.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Matthew Hausknecht, Prithviraj Ammanabrolu, Marc-Alexandre Côté, and Xingdi Yuan. Interactive fiction games: A colossal adventure. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 2020.

Peter Jansen, Marc-Alexandre Côté, Tushar Khot, Erin Bransom, Bhavana Dalvi Mishra, Bodhisattwa Prasad Majumder, Oyvind Tafjord, and Peter Clark. Discoveryworld: A virtual environment for developing and evaluating automated scientific discovery agents. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 10088–10116. Curran Associates, Inc., 2024. doi: 10.52202/079017-0324. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/13836f251823945316ae067350a5c366-Paper-Datasets_and_Benchmarks_Track.pdf.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Hassan Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. MEMENTO: Teaching LLMs to manage their context. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=YaYiQDVsi0.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Tianjian Li, Jingyu Zhang, William Jurayj, Xi Wang, Chuanyang Jin, Mehrdad Farajtabar, Eric Nalisnick, and Daniel Khashabi. Self-compacting language model agents, 2026a. URL https://arxiv.org/abs/2606.23525.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. Compactionrl: Reinforcement learning with context compaction for long-horizon agents, 2026b. URL https://arxiv.org/abs/2607.05378.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Scaling llm multi-turn rl with end-to-end summarization-based context management, 2025. URL https://arxiv.org/abs/2510.06727.

OpenAI. Introducing SWE-bench verified. OpenAI announcement, August 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18724–18741, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1043. URL https://aclanthology.org/2024.emnlp-main.1043/.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphaël Lopez Kaufman, Aidan Clark, Seb Noury, Matthew Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7487–7498. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/parisotto20a.html.

Alexandre Piché, Ehsan Kamalloo, Rafael Pardinas, Xiaoyin Chen, and Dzmitry Bahdanau. PipelineRL: Faster on-policy reinforcement learning for long sequence generation. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=A35ak14Cyp.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Prime Intellect Team, Mika Senghaas, Fares Obeid, Sami Jaghouar, William Brown, Jack Min Ong, Daniel Auras, Matej Sirovatka, Jannik Straube, Andrew Baker, Sebastian Müller, Justus Mattern, Manveer Basra, Aiman Ismail, Dominik Scherm, Cooper Miller, Ameen Patel, Simon Kirsten, Mario Sieg, Christian Reetz, Kemal Erdem, Vincent Weisser, and Johannes Hagemann. Intellect-3: Technical report, 2025. URL https://arxiv.org/abs/2512.16144.

Dheeraj Vattikonda, Santhoshi Ravichandran, Emiliano Penaloza, Hadi Nekoei, Thibault de Chezelles, Megh Thakkar, Nicolas Gontier, Miguel Muñoz Mármol, Sahar Omidi Shayegan, Stefania Raimondo, Steve (Xue) Liu, Alexandre Drouin, Alexandre Piche, Alexandre Lacoste, and Massimo Caccia. How to train your llm web agent: A statistical diagnosis. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 53256–53282. Curran Associates, Inc., 2025. doi: 10.52202/085713-1774. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/4d239452d41ded07139801d7c966c629-Paper-Conference.pdf.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader?, 2022. URL https://arxiv.org/abs/2203.07540.

Yuyao Wang, Zhongjian Zhang, Mo Chi, Kaichi Yu, Yuhan Li, Miao Peng, Bing Tong, Chen Zhang, Yan Zhou, and Jia Li. Evomembench: Benchmarking agent memory from a self-evolving perspective. *arXiv preprint arXiv:2605.18421*, 2026.

Ian Wu, Yuxiao Qu, Amrith Setlur, and Aviral Kumar. Reasoning cache: Continual improvement over long horizons via short-horizon RL. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=IpeH08AJ9P.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. Resum: Unlocking long-horizon search intelligence via context summarization, 2026b. URL https://arxiv.org/abs/2509.13313.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Yuchen Yan, Liang Jiang, Jin Jiang, Shuaicheng Li, Zujie Wen, Zhiqiang Zhang, Jun Zhou, Jian Shao, Yueting Zhuang, and Yongliang Shen. Inftythink+: Effective and efficient infinite-horizon reasoning via reinforcement learning, 2026a. URL https://arxiv.org/abs/2602.06960.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Inftythink: Breaking the length limits of long-context reasoning in large language models. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=T1h5em349L.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

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 *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Jiale Zhao, Guoxin Chen, Fanzhe Meng, Minghao Li, Jie Chen, Hui Xu, Yongshuai Sun, Wayne Xin Zhao, Ruihua Song, Yuan Zhang, et al. Immersion in the github universe: Scaling coding agents to mastery. *arXiv preprint arXiv:2602.09892*, 2026.

	TextWorld	ALFWorld	SWE
Summary	48 (192)	64 (64)	–
Sliding-Window	48 (192)	64 (64)	70 (280)
Markovian Thinker	48 (192)	64 (64)	68 (272)
Markovian Picker	48 (192)	64 (64)	–
Full context	12 (48)	16 (16)	28 (112)

Table 2: Concurrent rollouts per inference GPU for each strategy, with the total across GPUs in parentheses. Settings are the same for re-prefill compaction and KV-streams.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=XY8AaxDSLb.

A Broader Impacts

KV-streams reduces repeated prefill during training, making experiments with long-running agents less expensive. The same efficiency gains may also make harmful or unauthorized automation cheaper. Agents that handle sensitive information still need access controls, monitoring, and secure data handling. We do not release new pretrained models or datasets, and using KV-streams does not remove the need for the safety policies, permissions, and privacy protections of the underlying models and environments.

B Further Details

All runs within each benchmark share the same hyperparameters (Table 1). Text-based games run on prime-RL with vLLM, while software-engineering tasks use a Megatron-based trainer with SGLang. For each strategy, we use the same rollout concurrency under re-prefill compaction and KV-streams (Table 2). Full context needs a lower concurrency because long traces occupy more memory. The async level limits how many policy versions a rollout can lag behind the trainer; we drop rollouts that exceed the off-policy step limit.

	TextWorld	ALFWorld	SWE
Model	`Qwen3-4B-Instruct-2507`	`Qwen3-4B-Instruct-2507`	`Qwen3.5-4B`
GPUs (inference / trainer)	4 / 4 H100	1 / 3 A100	4 / 4 GB200
Rollouts per prompt	8	8	8
Batch size (rollouts per step)	512	128	256
Gradient steps	500	200	200
Learning rate	10^{-6}	10^{-6}	10^{-6}
Weight decay	0.01	0.01	0.1
Adam (β_1, β_2)	$(0.9, 0.9)$	$(0.9, 0.9)$	$(0.9, 0.98)$
Gradient norm clipping (max norm)	1.0	1.0	1.0
KL coefficient	0.001	0.001	0.001
Sampling temperature	1.0	1.0	1.0
Max tokens per turn	1,024	1,024	8,192
Sequence budget (tokens)	32k	16k	64k
Max turns per episode	200	100	100
Compaction interval (turns)	10	10	30
KV block size (tokens)	16	16	1
Async level / off-policy steps	1 / 3	1 / 3	1 / 2
Evaluation interval (steps)	15	20	20
Evaluation set	256 games	70 seen + 67 unseen	SWE-bench Verified

Table 1: Hyperparameters shared by all runs within each benchmark. Compaction strategy and rollout concurrency vary between runs (Table 2).

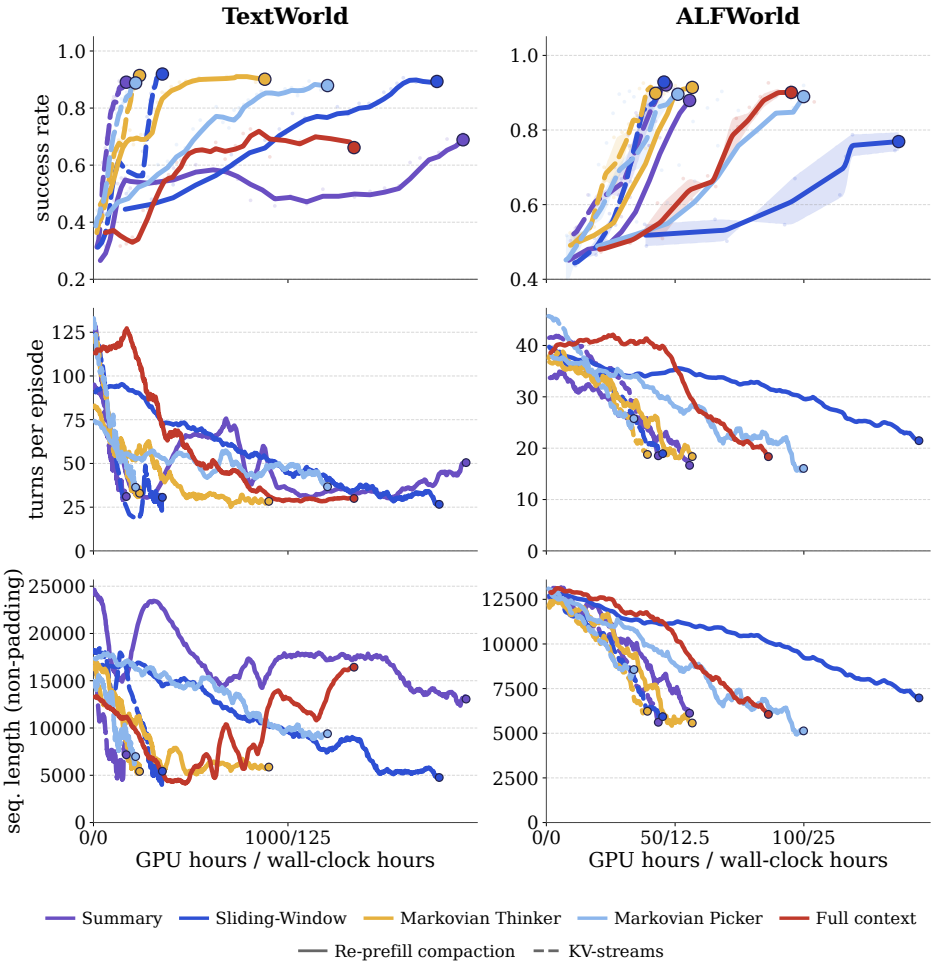


Figure 8: Training curves through each run's last checkpoint (dot). Summary and Sliding-Window with re-prefill compaction exceed the full-context compute budget. The lower rows show smoothed mean turns per episode and rollout length, excluding padding.

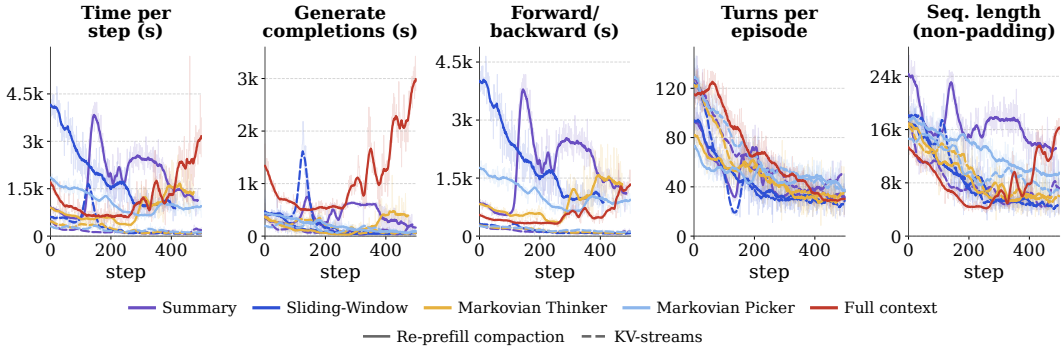


Figure 9: Per-step timing and rollout statistics, including Summary and Sliding-Window with re-prefill compaction.

C Full Training Curves

Summary and Sliding-Window with re-prefill compaction exceed the full-context compute budget and have much longer step times (Figures 8 and 9). Curves end at each run's last checkpoint, including runs stopped after exceeding the full-context wall-clock time. We also compare mean training reward and evaluation success rate by gradient step (Figure 10).

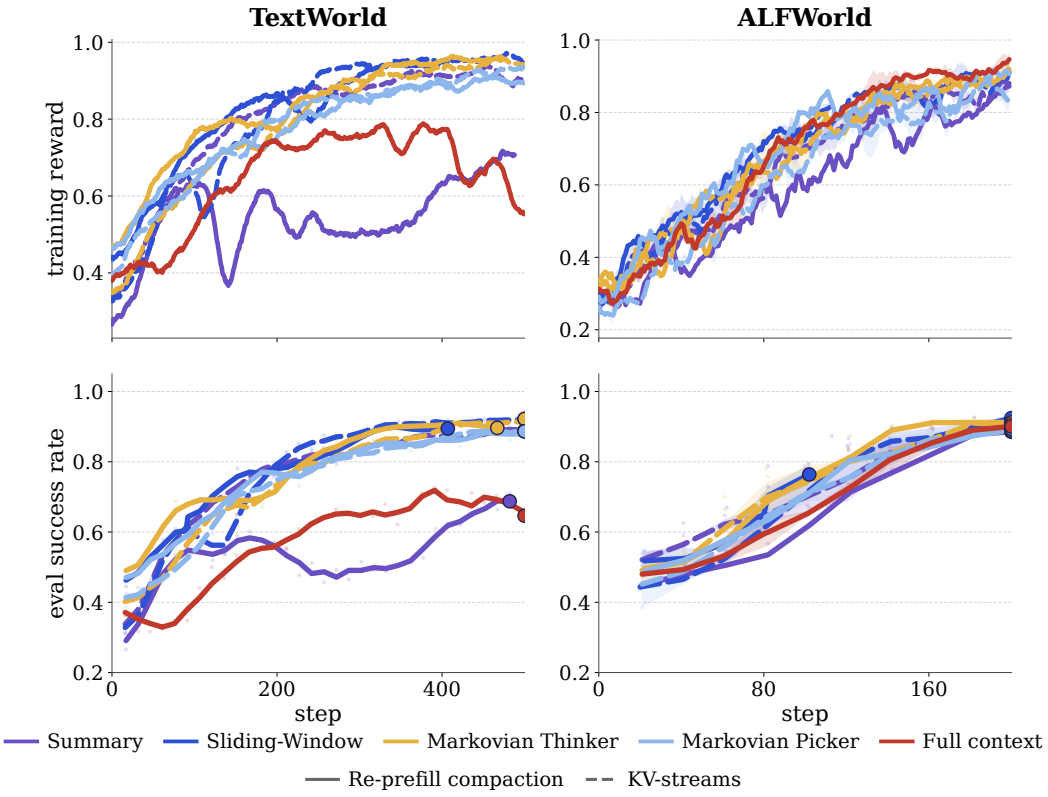


Figure 10: Training curves by gradient step on TextWorld (left) and ALFWorld (right). Top: smoothed mean training reward. Bottom: evaluation success rate, with faint points for individual evaluations. Multi-seed curves show the per-step mean and shaded seed range. Solid: re-prefill compaction; dashed: KV-streams.

D SFT Warm-Start Training Loss

For the warm-start experiment in Section 6.1, we train for 500 steps on 10k full-context trajectories from `Qwen3-4B-Instruct-2507`, using randomly sampled eviction attention masks. The loss falls quickly over the first 50 steps and then decreases more slowly (Figure 11).

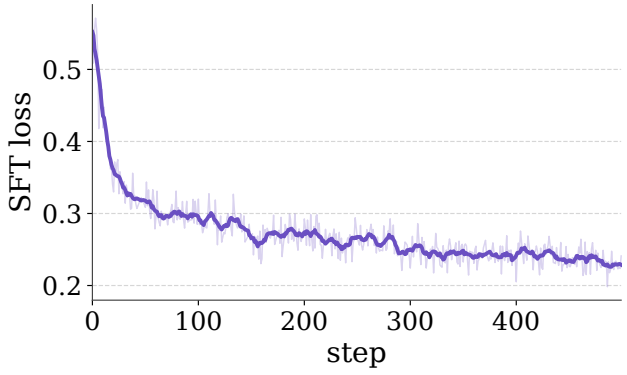


Figure 11: SFT warm-start loss by optimizer step. The faint line shows per-step loss; the bold line shows a centered 9-step rolling mean.

E vLLM PADDING

Our vLLM configuration commits only complete 16-token blocks to the prefix cache. If a turn ends partway through a block, that block must be recomputed on the next request. We therefore pad each turn to the next multiple of 16 so that KV-streams can reuse the full turn's KVs. This adds tokens to the rollout, which we count in Figure 3. SGLang supports a block size of 1 and does not need this padding.

F TURN-WISE COMPACTION

In the agentic experiments, we trigger compaction and remove context in whole turns so that message boundaries remain intact. The retained length r counts the tokens in the retained turns plus any generated summary (Section 2).

G CONTROLLED RETRIEVAL PROTOCOL

Assignment. We first assign the model one of five fruits: apple, banana, mango, orange, or pineapple. The assignment prompt lists all five candidates and asks the model to reply "Assignment acknowledged." without repeating the fruit. We evaluate on 200 held-out trials, balanced at 40 trials per fruit.

Counting sequence. We then ask the model to count upward in space-separated integers, such as "1 2 3 4 5 . . .", without naming or paraphrasing a fruit or answering the retrieval question. We vary the sequence budget from 16 to 512 tokens, using intermediate budgets of 32, 64, 128, and 256. Since an integer can occupy several tokens, the budget limits the number of decoded tokens rather than the final integer reached. The sequence remains in context for retrieval, so we check that it does not contain a candidate answer. All reported held-out trials pass this check.

Eviction and retrieval. Next we evict the assignment and acknowledgment as one turn, retaining the counting sequence and its KVs. The retrieval prompt asks the model to recall its assigned fruit from the five candidates, using the format "Recall: <fruit>". We score the parsed answer by exact match and check the traces to confirm that eviction occurred before the first answer token was generated. Because the assignment is no longer visible and the counting text contains no answer, correct recall requires information carried forward in the retained KVs.

Prompt generalization. To test whether recall depends on the wording of the retrieval instruction, we evaluate the five-fruit checkpoints with a changed prompt and no further training (Figure 12B). The assignments, candidates, sequence budget, and eviction procedure stay the same. In both prompts, `[choices]` is a shuffled, comma-separated list of the five fruits.

> *Training and in-distribution evaluation:* "Which fruit were you assigned from these five fruits: [choices]? Respond with exactly 'Recall: <fruit>', replacing <fruit> with the assigned fruit. Do not explain."
>
> *Prompt-generalization evaluation:* "What is your favourite object? Choose from these five options: [choices]. Respond with exactly 'Recall: <object>', replacing <object> with one option. Do not explain."

Zero-support target (TV). To test recall beyond the listed candidates, we train on six objects: apple, banana, mango, orange, pear, and television (Figure 12C). Both the assignment and retrieval prompts list only the five fruits as candidate answers, but the assignment explicitly names television on TV trials. We use the same counting and eviction procedure, so a correct answer must recover the assigned object even when it is not listed. We evaluate on 240 trials, balanced at 40 per object.

Actor-name transfer. To test generalization beyond fruits, we take the five-fruit checkpoints and assign actor names instead: Kathryn Erbe, Kevin Bacon, Naomi Watts, Nicole Kidman, and Toni Collette. There is no further training. The model generates the counting sequence and then recalls the name after we evict the assignment turn. We evaluate both with the five names listed as candidates

and without a candidate list, using 200 trials per condition, balanced at 40 per name. As in the fruit task, we reject counting sequences that contain a candidate answer and score recall by exact match after eviction.

Training data and checkpoint selection. We train a separate model for each objective and sequence budget (16, 32, 64, 128, 256, or 512 tokens). The five-fruit task uses 1,000 training and 200 held-out assignments, while the TV task uses 1,200 training and 240 held-out assignments across all six objects. Both datasets contain 200 training and 40 held-out examples per target. RL, RFT, and SFT start from `Qwen3-4B-Instruct` and run for 100 updates. We evaluate their final checkpoints (Figure 12). Self-distillation starts from the corresponding SFT checkpoint and runs for another 100 updates. The prompt and actor-name evaluations reuse the five-fruit checkpoints without further training.

RL. We sample on-policy batches of 128 trajectories, with four rollouts per assignment, at temperature 1.0 and top-$p = 0.95$. The reward is one when the post-eviction answer matches the assignment and the counting text contains no candidate answer, and zero otherwise. The policy-gradient loss weights each generated response token's negative log-probability by the trajectory's advantage. We use a batch-wide leave-one-out baseline: each trajectory's reward is compared with the mean reward of the other 127 trajectories. If the entire batch is incorrect, the baseline is instead the chance accuracy ($1/5$ for five fruits and $1/6$ for hidden TV). No supervised target loss is added.

RFT. Rejection-sampling fine-tuning (RFT) uses the same online sampling and binary reward as RL, but minimizes negative log-likelihood only on generated response tokens in successful trajectories. Incorrect or contaminated trajectories contribute zero loss; there is no leave-one-out advantage weighting. Both RL and RFT use AdamW with a constant learning rate of 5×10^{-6}, weight decay 0.01, $(\beta_1, \beta_2) = (0.9, 0.95)$, and gradient-norm clipping at 1.0.

SFT. For SFT, we construct demonstrations containing the assignment acknowledgment, a deterministic sequence of ascending integers truncated to the specified token budget, and the correct response `Recall: <target>`. We minimize teacher-forced cross-entropy on assistant tokens, excluding system and user tokens. Training uses the eviction attention mask so that the answer cannot attend to the assignment turn. We use batches of 128, AdamW with zero weight decay, $(\beta_1, \beta_2) = (0.9, 0.95)$, and gradient-norm clipping at 1.0. The learning rate warms up for 20 updates to 10^{-5}, then follows a cosine schedule to 10^{-6}.

Self-distillation. For self-distillation, we reuse the SFT demonstrations and add a KL loss to the supervised cross-entropy, each with coefficient 1.0. We run the student with the eviction attention mask and obtain the teacher distribution from a full-context pass of the same current model, without gradients. The teacher can therefore attend to the assignment. The added loss approximates $D_{\mathrm{KL}}(p_{\mathrm{student}} \| p_{\mathrm{teacher}})$ by summing over the student's top 100 vocabulary tokens at each assistant position, without renormalizing their probability mass. The teacher is recomputed as training progresses. Optimizer and batch settings match SFT, except that the learning rate warms up for five updates to 5×10^{-7} and follows a cosine schedule to 10^{-7}.

In all four objectives, the eviction mask prevents answer tokens from attending to the assignment turn while allowing attention to the counting sequence's retained KVs. Training can therefore change how the model carries information forward through these states, but it does not alter a cache after generation.

Results across sequence budgets. We average accuracy over three seeds at the best learning rate for each condition (Figures 12 and 13). We select this rate by comparing mean accuracy only among configurations with three completed seeds, and use the same rate for all three. The RL and SFT results in Figure 7 use the same selection rule. For the TV task, accuracy is averaged over all six objects, not only TV trials.

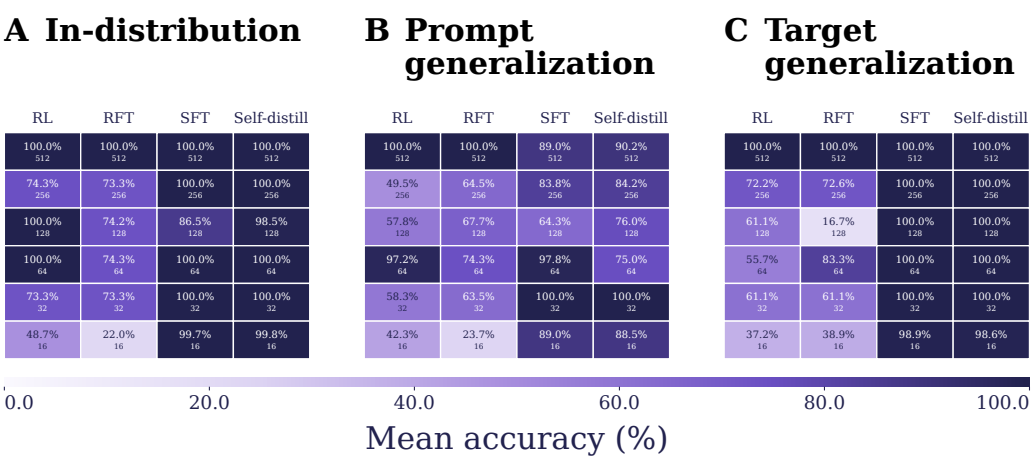


Figure 12: Mean post-eviction accuracy over three seeds with (A) the training prompt, (B) a changed prompt, and (C) six trained targets but only five listed answer options. Smaller cell labels give the counting length.

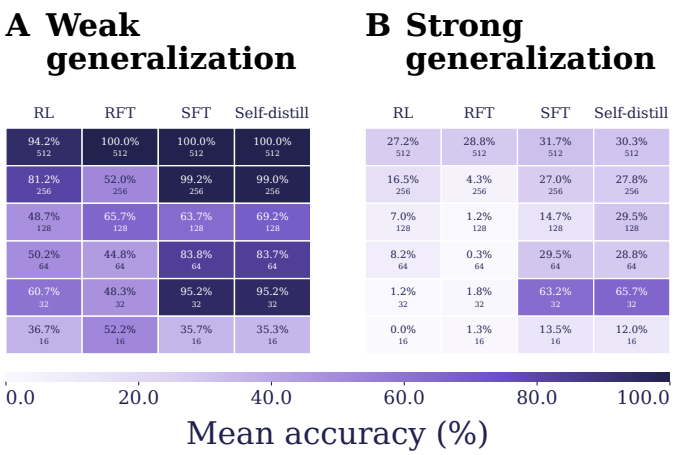


Figure 13: Mean retrieval accuracy over three seeds for new person names with (A) and without (B) candidate answers. Smaller cell labels give the counting length.

H Prefill Token Cost

Consider a rollout that generates N tokens with compaction operator κ and token budget B. Each compaction retains $r = o + s$ tokens: overlap o and any generated summary s. Re-prefill compaction copies these tokens into a new trace and prefills them again.

Number of compactions. After a compaction, the context holds r tokens and generates $b = B - r$ more before the next compaction. Generating N tokens therefore takes

$$m = \frac{N}{b} = \frac{N}{B - r}$$

compactions.

Prefill cost. Each compaction adds a forward pass over the r retained tokens, giving an extra token cost of

$$\Delta_{\text{re-prefill}} = m\,r = \frac{N\,r}{B - r} = \frac{N(o + s)}{B - (o + s)}.$$

This cost grows without bound as r approaches B and depends only on the total retained length, not its split between overlap and summary. Keeping only a summary of length $s = B/4$ gives $r = B/4$

and $\Delta_{\text{re-prefill}} = N/3$, regardless of B. KV-streams retains the KVs without prefilling them again, so $\Delta_{\text{re-prefill}} = 0$.