TRASURE: EXPLOITING CROSS-TRAJECTORY RECURRENCE TO ACCELERATE INFORMATION-SEEKING RL ROLLOUTS OVER SHARED CORPORA

Anonymous authors
Paper under double-blind review

ABSTRACT

Training agents with reinforcement learning (RL) to answer questions over shared corpora, such as enterprise knowledge bases, incurs substantial rollout cost. We identify **cross-trajectory recurrence** in this setting: trajectories for the same or different queries retrieve the same external items under different contexts, causing repeated prefill that group-based sampling further amplifies. However, this recurrence remains unexplored for rollout acceleration. We present TraSURE, which exploits this recurrence to accelerate RL rollouts over shared corpora. ***1*** For prefill, we use position-independent caching (PIC) to share recurring-item KV states across trajectories and avoid redundant prefills. ***2*** Building on this caching layer, we design *demand-aware rollout scheduling* to coordinate cache admission and retention using current and future item demand. ***3*** Beyond prefill, we use a next-outcome posterior to select historical continuations as drafts for *posterior-guided speculative decoding*. Across six models and four information-seeking workloads over shared corpora, TraSURE delivers up to $6.0\times$ speedup over standard rollouts, with all task scores within 1 pp of full-causal counterparts.

1 INTRODUCTION

Complex real-world tasks require large language model (LLM) and multimodal LLM (MLLM) agents to seek and reason over information beyond their parameters (Lewis et al., 2020; Li et al., 2024; Liu et al., 2024). Often, this information resides in *shared corpora* private to an organization and is repeatedly queried by agents, such as enterprise knowledge bases and multimodal collections. These agents exhibit *agentic behavior*: they decide what to seek, query these corpora, and use retrieved evidence for multi-turn reasoning and actions (Yao et al., 2023; Trivedi et al., 2023). Enterprise agents answer questions from internal knowledge sources (Choi et al., 2025; Brehme et al., 2025); for example, they may consult an employee handbook, onboarding slides, and HR training videos about probationary leave or new-hire allowances. *Agentic reinforcement learning (RL)* trains information-seeking behavior using outcome rewards on complete multi-turn interactions, without step-level supervision (Jin et al., 2025; Song et al., 2025; Chen et al., 2025; Chang et al., 2026).

In shared-corpus workloads, a query-to-answer interaction is a *trajectory*; generating one is a *rollout*. During a rollout, the agent retrieves corpus passages, pages, or multimodal objects, which we call *external items*. ***1*** These items add many additional context tokens; queries often require several, while RL exploration may access even more, making rollouts especially prefill-intensive. ***2*** On-policy training repeats this cost because every model update requires fresh trajectories. ***3*** Group-based methods such as GRPO (Shao et al., 2024) and DAPO (Yu et al., 2025) multiply it by sampling several trajectories per query for relative rewards. Together, these three factors make rollout generation a recurring RL bottleneck. On MultiHop-RAG, a multi-hop QA benchmark over a shared knowledge base (Tang & Yang, 2024), rollouts occupy 62% of an RL iteration (Fig. 1). Prefill accounts for approximately 82% of baseline rollout time. Yet, concentrated item demand creates an opportunity for computation reuse across trajectories. In this workload, the top 10% of items account for 65% of gold-evidence occurrences and 54% of actual rollout accesses (Fig. 1, inset). We call such repeated access **cross-trajectory recurrence**: *different trajectories can access the same*

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.

external item under different contexts. This recurrence occurs among trajectories for the same query and across trajectories for different queries, enabling item-level reuse beyond shared prefixes.

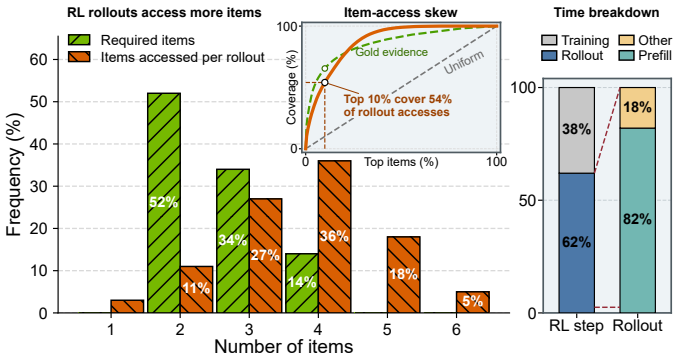

Figure 1: **Item access and rollout time in MultiHop-RAG.**

Cross-trajectory recurrence nevertheless remains unexploited by existing rollout accelerators: they either reduce trajectory-level work (Zhang et al., 2026; Zhai & Wang, 2026; Zhou et al., 2025) or reuse KV states along exact shared prefixes (Zheng et al., 2024; vLLM Project, 2026; Zhu et al., 2025), which rarely persist once trajectories diverge. Position-independent caching (PIC), developed for LLM serving, enables item-level reuse across different contexts, beyond exact-prefix matches (Hu et al., 2024). Yet, to our knowledge, PIC remains unexplored for accelerating group-based on-policy RL. This raises a central question: *How can we exploit recurring external items across diverging trajectories to accelerate group-based rollouts?*

We propose TraSURE, a method that exploits cross-trajectory recurrence to accelerate information-seeking RL rollouts over shared corpora. ***1*** During the RL rollout stage, its core, `PICache`, reuses item KV states across trajectories without per-request contextual recomputation. During policy optimization, we continue to train the agent's information-seeking behavior while using the same position-isolated mask as in rollouts; this maintains the training-inference consistency required by on-policy RL and **also adapts the policy to independently cached items.** ***2*** Building on `PICache`, we introduce *demand-aware rollout scheduling*, which uses current item demand to form reuse-friendly batches and a transition posterior to estimate future demand and retain items likely to be needed in subsequent rollout steps. This reduces redundant prefill computation both within the current batch and across later rollout steps. ***3*** Beyond prefill acceleration, TraSURE reduces autoregressive generation cost through *posterior-guided speculative decoding*, which uses a posterior over next outcomes to select historical continuations as speculative drafts. Our contributions are:

- We identify **cross-trajectory recurrence** in group-based on-policy RL over shared corpora and introduce `PICache`, which reuses item KV states with consistent position-isolated attention across rollout generation and policy optimization.
- We develop *demand-aware rollout scheduling*, which batches requests by marginal item-prefill work, coalesces concurrent misses into a single materialization, and coordinates cache admission and retention using current and posterior-estimated future demand.
- We propose *posterior-guided speculative decoding*, which uses transition-group posteriors to allocate a fixed draft-tree budget to historical continuations likely for the current request.
- Across six models and four shared-corpus workloads, the live-rollout results reach up to $6.0\times$ speedup over vanilla slime with SGLang radix caching, while all final task scores remain within 1 pp of independently trained full-causal counterparts.

2 Related Work

2.1 Reinforcement Learning for Information Seeking

Information-seeking agents acquire task-relevant knowledge by querying external resources. ReAct (Yao et al., 2023) interleaves reasoning with actions that gather information, while IR-CoT (Trivedi et al., 2023) uses intermediate reasoning to guide retrieval and subsequent reasoning. FLARE (Jiang et al., 2023) and Self-RAG (Asai et al., 2024) further study adaptive information acquisition during generation. SearchLVLMs (Li et al., 2024) and LLaVA-Plus (Liu et al., 2024) extend this paradigm to web-augmented visual QA and multimodal tool use.

Both supervised fine-tuning (SFT) and reinforcement learning (RL) learn such behaviors. FireAct (Chen et al., 2023a) learns reasoning-action sequences from demonstrations, whereas Search-R1 (Jin et al., 2025), R1-Searcher (Song et al., 2025), and ReSearch (Chen et al., 2025) learn multi-

turn search from outcome-level rewards. KARL (Chang et al., 2026) applies off-policy RL to enterprise knowledge agents. On-policy RL keeps updates aligned with current-policy behavior by collecting fresh trajectories before each update, as in Search-R1 and ReSearch (Jin et al., 2025; Chen et al., 2025), while group-based methods such as GRPO and DAPO require multiple trajectories per query (Shao et al., 2024; Yu et al., 2025). In multi-turn information seeking, these requirements compound: each policy update must wait for several fresh interaction trajectories, placing rollout generation on the sequential critical path.

2.2 On-policy Rollout Acceleration

Existing methods reduce on-policy rollout cost at two levels: by reducing trajectory-level generation or by accelerating the individual LLM requests within each trajectory.

Trajectory-level Acceleration: AERO and Selective Rollout reduce cost by adaptively pruning or terminating rollouts, while APRIL defers incomplete rollouts to later iterations (Zhang et al., 2026; Zhai & Wang, 2026; Zhou et al., 2025). These methods alter which trajectories are completed within each policy update. In contrast, TraSURE leaves rollout selection and termination unchanged, accelerating the same on-policy samples through cross-trajectory computation reuse. **Prefill Computation Reuse:** Prefix caching in SGLang (Zheng et al., 2024) and vLLM (vLLM Project, 2026) reuses KV states along shared prefixes and is inherited by RL post-training frameworks such as slime (Zhu et al., 2025) and Vime (Vime Contributors, 2026). However, it cannot reuse the same external item under different preceding contexts or at different token positions. EPIC (Hu et al., 2024) relaxes these constraints through position-independent caching. It targets general LLM serving and does not address cross-trajectory position-independent reuse in group-based on-policy RL. Realizing such reuse also requires coordinating cache materialization and retention around recurring cross-trajectory demand. **Decode-side Reuse through Speculation:** Speculative decoding accelerates generation using drafts produced at lower cost, then verified by the current policy. HistoSpec in RhymeRL (He et al., 2025), SPEC-RL (Liu et al., 2025), and DAS (Shao et al., 2026) reuse continuations from historical trajectories as drafts, whereas EfficientRollout (Kim et al., 2026) derives a lightweight drafter from the current policy. These methods reuse computation from prior trajectories for decoding, but do not connect such reuse to recurring external items across trajectories.

Overall, existing methods reduce trajectory-level generation, reuse request-level prefill, or reuse historical continuations during decoding. They do not explicitly exploit item recurrence across queries and trajectories as a shared reuse opportunity spanning both prefill and decoding.

3 Preliminaries

We consider group-based on-policy RL, such as GRPO, for an information-seeking policy π_{θ_v} over shared corpora, where v indexes the policy version used by rollout workers. For each query q, the policy samples $\mathcal{T}(q) = \{\tau_0, \ldots, \tau_{G-1}\}$. All trajectories draw from a common collection of external items, although the retrieved subset may vary across queries and trajectories. For $i \in \{0, \ldots, G-1\}$, we write trajectory i as

$$\tau_i = (q, x_{i,0}, e_{i,0}, x_{i,1}, e_{i,1}, \ldots, x_{i,T}), \tag{1}$$

where $x_{i,t}$ is a policy-generated token span that may issue a retrieval action, and $e_{i,t}$ is the returned external item. We use *external item* to denote immutable retrieved content represented as a stable sequence of model-visible tokens or embeddings, such as a text passage, PDF page, image, audio segment, or video clip. The final span $x_{i,T}$ produces the final answer and has transition outcome `STOP`, so no item $e_{i,T}$ follows. A complete trajectory is formed through a sequence of requests to the policy model. Each request takes the preceding trajectory prefix as input and generates one span $x_{i,t}$. The returned item $e_{i,t}$, if any, extends the context for the next request. Generating a group of multi-turn trajectories repeatedly prefills growing contexts and decodes each new span; relative advantages can be computed and the policy updated only after all trajectories complete.

During rollout inference, *prefill* materializes key-value (KV) states for subsequent autoregressive *decode*. Under causal attention and positional encoding, an item's KV states depend on its preceding context and token positions. Conventional prefix caching therefore requires both to match across requests (Hu et al., 2024). KV-cache management retains these states under finite capacity; evicted

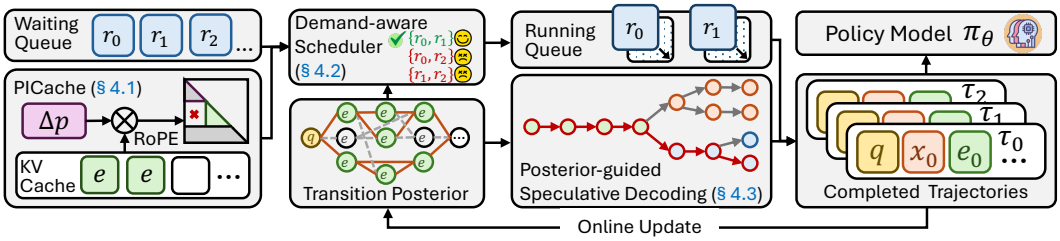

Figure 2: **System overview.** PICache removes repeated item prefill; completed trajectories update a transition posterior that guides admission and speculative decoding.

states must be recomputed on demand. A KV entry computed under θ_v is reusable only with a compatible policy version; an incompatible update invalidates it.

Model-free N-gram speculative decoding drafts from an available token-sequence multiset $\mathcal{C}$, where $N_{\mathcal{C}}(s)$ counts occurrences of s (He et al., 2024; Hu et al., 2025). For recent suffix h and candidate tree path $\pi(v)$, its conventional score is

$$P_{\text{token}}(\pi(v) \mid h; \mathcal{C}) := \frac{N_{\mathcal{C}}(h \,\|\, \pi(v))}{N_{\mathcal{C}}(h)}, \tag{2}$$

Matched continuations form a shared-prefix draft tree for parallel verification by the target policy, without changing the target-policy rollout distribution (Leviathan et al., 2023; Chen et al., 2023b).

4 Methodology

Fig. 2 provides an overview of TraSURE, implemented in the lightweight, extensible slime RL framework (Zhu et al., 2025), with the widely used, high-performance SGLang LLM serving engine (Zheng et al., 2024) powering its rollouts. TraSURE targets two dominant serving costs: repeated item prefill and autoregressive decode.

The system comprises three components, detailed in the following subsections. **PICache.** Position-independent caching makes each immutable item a reusable KV unit across requests with different preceding contexts and token positions. Position-isolated attention removes the item state's dependence on its preceding context, while shared radix-tree references and runtime RoPE rebasing let all occurrences use one physical item-KV footprint; applying the same attention semantics during rollout and policy optimization allows the policy to adapt during RL. **Demand-aware rollout scheduling.** Item-union admission batches requests by their marginal item work. A single-writer, multi-reader protocol prevents concurrent consumers from duplicating item prefill, while posterior-estimated future item demand guides admission and cache retention. **Posterior-guided speculative decoding.** It allocates a fixed draft-tree budget to continuations likely for the current request; the unchanged current-policy verifier preserves the rollout distribution.

The latter two components share an item-transition graph updated from historical traces and current sibling outcomes. For each transition group $g = (q, e^-)$, it maintains an online posterior over next items or termination. The posterior is an execution hint: estimation errors cause cache misses or suboptimal schedules, while current-policy verification rejects invalid drafts. Algorithm 1 in Appendix A gives the end-to-end rollout procedure.

4.1 Position-Independent Caching for Agentic RL Rollouts

We implement position-independent caching (PIC) in `PICache`, our shared item-KV layer. `PICache` reuses external items across trajectories under different preceding contexts and token positions. TraSURE applies the same PIC attention semantics during rollout generation and policy optimization, keeping the two phases aligned as the policy evolves.

Main Motivation: Cross-trajectory Item Reuse. Different trajectories often retrieve the same immutable item, but under different preceding contexts. Conventional prefix caches therefore cannot share its prefix-dependent KV states. Prior serving systems demonstrate that context reuse need not be limited to shared prefixes (Hu et al., 2024; Yao et al., 2025). This motivates us to ask

whether each recurring external item, rather than its entire request prefix, can serve as the reusable unit across RL trajectories.

RL-specific Opportunity: Policy Adaptation to PIC Semantics. In fixed-model serving, CacheBlend selectively recomputes cached tokens to recover context interactions lost through independent item encoding (Yao et al., 2025). *RL already trains the policy for information seeking; the same training loop also lets it adapt to independently cached items.* We apply position-isolated attention during both rollout generation and policy optimization, allowing the policy to learn under consistent attention semantics. Cached item states are then reused without per-request contextual recomputation; Section 5.2 evaluates policy quality.

Design: Position-isolated Attention. A complete RL trajectory comprises successive policy-model generation requests (Eq. (1)). At turn t, a request takes the available prefix r_t and autoregressively generates span x_t (trajectory index i omitted):

$$r_t = [q, x_0, e_0, \ldots, x_{t-1}, e_{t-1}], \quad x_t \sim \pi_{\theta_v}(\cdot \mid r_t). \tag{3}$$

Here brackets denote token concatenation. Prefill computes any input KV states not already cached. For the request in Eq. (3), attention operates on r_t during prefill and on $[r_t, x_{t,<j}]$ when predicting token $x_{t,j}$.

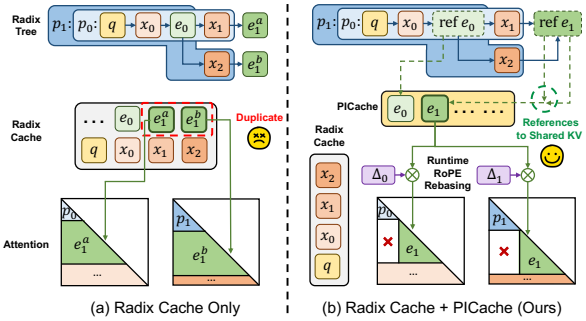

Figure 3: **Position-independent KV reuse.** PIC shares one item entry across radix-tree paths and rebases RoPE at runtime without KV copies.

For each block occurrence z, let $\mathcal{I}(z)$ denote its token positions within this available prefix. We use the additive mask M^{PIC}_{uv} during both rollout generation and policy optimization, where 0 permits attention from query position u to key position v and $-\infty$ blocks it:

$$M^{\mathrm{PIC}}_{uv} = \begin{cases} 0, & v \le u \wedge \left[\begin{array}{ll} \text{(C1)} & u \notin \bigcup_{k=0}^{t-1} \mathcal{I}(e_k) \\ \vee\ \text{(C2)} & \exists\, 0 \le k < t :\ u, v \in \mathcal{I}(e_k) \end{array} \right] \\ -\infty, & \text{otherwise.} \end{cases} \tag{4}$$

Condition C1 gives all non-item tokens ordinary causal access to preceding blocks. Condition C2 restricts every item token to its own item block. Each e_k is therefore independent of the preceding trajectory context, while every later x_t can still use all prior conversational and item blocks. For example, after two retrievals the accumulated context is $[q, x_0, e_0, x_1, e_1]$. When the policy generates x_2, it can attend to the entire context, while items e_0 and e_1 remain independently cacheable.

Design: Shared-reference Radix Tree with Copy-free RoPE Rebasing. Position-isolated attention removes the semantic dependence on preceding context, but cached keys still encode their original RoPE positions. To make these keys physically reusable, we integrate `PICache` with SGLang's RadixAttention, whose radix-tree nodes manage physical KV slots (Zheng et al., 2024). This requires multiple tree occurrences to share one physical entry and rebase it across token positions. Instead of storing a separate KV copy for every radix-tree occurrence of an item, we let all occurrences point to a single item-KV entry in `PICache`; non-item nodes retain SGLang's original storage. Thus, the two e_1 nodes in Fig. 3 share one physical entry. At within-item offset j, reusing an entry cached at position ℓ at position ℓ' yields the following rebased key and unchanged value:

$$K^{(r)}_{\ell'+j} = R(\ell'+j)R(\ell+j)^{-1}K_{\ell+j} = R(\ell'-\ell)K_{\ell+j}, \qquad V^{(r)}_{\ell'+j} = V_{\ell+j}, \tag{5}$$

where $R(\cdot)$ is RoPE's rotary transformation (Su et al., 2024). We rebase on the fly inside the attention kernel: each occurrence supplies its positional offset while the kernel reads the shared entry directly.

4.2 Demand-aware Rollout Scheduling

PIC makes item states reusable across requests, and the scheduler decides which requests execute together. As slots open, it admits ready requests from the waiting queue to the running queue for prefill and decode. Each completed request frees its slot for another waiting request. If its output

invokes the environment, the observation is appended to the trajectory's context to form the next request, which rejoins the waiting queue.

Challenge: Scheduling Wastes Item Prefill. Viewing each request in isolation hides item-level reuse across both concurrent and near-future requests. Fig. 4 illustrates two temporal failure modes for a recurring item: duplicate materialization within a demand burst and recomputation across bursts after eviction. Fig. 5 quantifies the first: concurrent duplicate prefill rises from 15% to 41% of executed item-prefill tokens as sampling increases from four to ten trajectories per query. Beyond concurrent duplication, overlapping item sets make a request's marginal prefill cost depend on its batch peers, but FCFS ignores this dependence and can form batches with costly item unions. Finally, hidden next-item demand causes premature eviction and re-prefill.

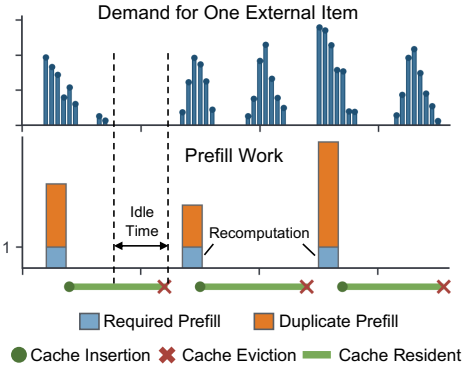

Figure 4: **FCFS wastes item-prefill work.** It duplicates concurrent item prefill and triggers recomputation after eviction.

Opportunity: Item Demand Reveals Reuse. First, PIC lets compatible requests share item KV, allowing one prefill to serve concurrent consumers. Second, waiting requests' known item sets and the current cache reveal shared work and missing items, allowing batch costs to be compared by item union. Third, historical transitions and newly revealed sibling outcomes support online estimates of the next items needed by running requests for admission and retention.

Design: Item-union-aware Batch Admission.

To build each batch, TraSURE first shortlists feasible requests with low marginal item work, then uses running requests' next-item posteriors to select materializations likely to be reused soon (detailed in Algorithm 2 in Appendix A).

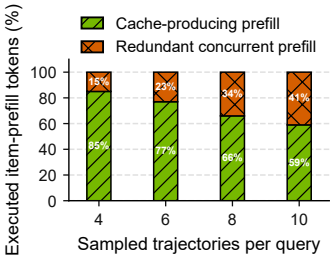

Figure 5: **Concurrent prefill waste grows with fan-out.**

1 *Form Shared-item Batches.* Let W be the waiting queue and $\mathcal{E}_u$ the items required by request $u \in W$. A batch $B \subseteq W$ requires the item union $\mathcal{E}(B) = \bigcup_{u \in B} \mathcal{E}_u$. For example, requests needing $\{e_1, e_2\}$ and $\{e_2, e_3\}$ require three distinct items, with e_2 shared. To realize this batch-level reuse, we design a single-writer, multi-reader materialization protocol in `PICache` (Algorithm 3 in Appendix A). For each missing item, one leader prefills a shared set of physical KV slots, while compatible followers map their item positions to the same slots and wait until the entry becomes `ready` before computing dependent suffixes. Leaders for different items execute in parallel, turning concurrent misses into one shared materialization.

2 *Shortlist Requests by Marginal Work.* Let C be the cached items, $\mathcal{E}(r)$ the items required by request r, and L_e the token length of item e. Starting from $B = \varnothing$, we rank feasible waiting requests by their additional item-prefill work:

$$\Delta(r \mid B, C) = \sum_{e \in \mathcal{E}(r) \setminus (C \cup \mathcal{E}(B))} L_e. \tag{6}$$

We retain a shortlist of feasible requests with low marginal work, exploiting cached items and within-batch sharing. Step **3** selects among these candidates; we then update $\mathcal{E}(B)$ and repeat until the batch is full.

3 *Account for Next-step Demand.* Current demand guides candidate formation, while the items materialized now also affect later cache reuse. We predict running requests' next items so admission can account for future demand before the corresponding requests enter the waiting queue.

Each running request belongs to a *transition group* $g = (q, e^-)$, identified by its prompt q and most recently retrieved item e^-, or `START` before the first retrieval. Let $H_{g,o}$ and $O_{g,o}$ count historical and current-collection transitions from g to outcome o, where o is an item or `STOP`. Their normalized

sum gives the online posterior:

$$\widehat{p}(o \mid g) = \frac{H_{g,o} + O_{g,o}}{\sum_{o'} (H_{g,o'} + O_{g,o'})}. \tag{7}$$

Revealed sibling outcomes update $O_{g,o}$ and refine the posterior online. Within the shortlist, we select the request with greatest expected next-step reuse of newly materialized items by running requests, weighted by item length. A candidate with higher current work can therefore be selected when it offers greater subsequent reuse.

Admission and Eviction. `PICache` rechecks capacity for the selected item union, protects required entries, and allocates missing KV slots. Current and posterior-estimated demand guide eviction among unprotected entries; reservations remain protected through their last consumer. Fig. 6 illustrates the decision. With e_0 and e_2 cached, the scheduler selects $\{R_0, R_1\}$, maximizing immediate reuse, materializing e_1 and e_3 once, and aligning the cache with next-round demand. The posterior guides only scheduling and cache management; the policy still generates actions, and the environment returns external items. Consequently, prediction errors may reduce reuse through suboptimal admission or eviction decisions, but cannot alter the generated trajectory or its policy distribution.

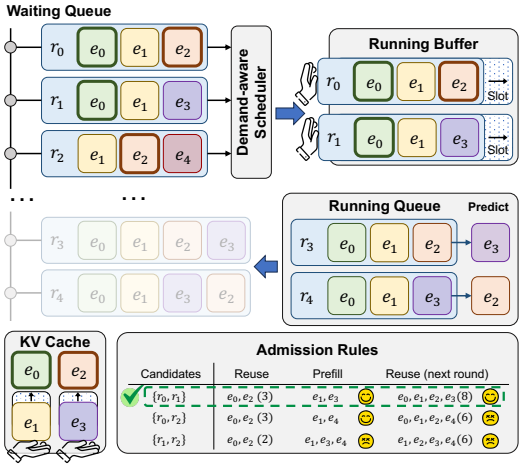

Figure 6: **Demand-aware admission.**

4.3 Posterior-guided Speculative Decoding

PIC and demand-aware scheduling accelerate prefill, but policy spans still decode autoregressively. TraSURE uses the transition-group posterior from Section 4.2 to allocate a fixed draft-tree budget to likely continuations.

Challenge: Pooled Drafting Wastes Nodes.

Model-free N-gram drafting ranks continuations globally. Because useful continuations depend on the prompt and latest retrieved item, a fixed node budget may be spent on paths frequent in other transition groups but unlikely for the current request.

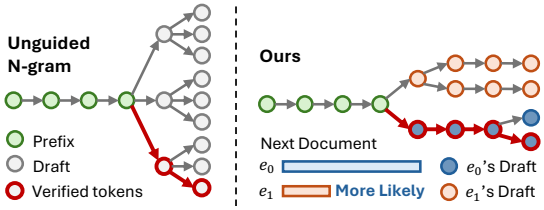

Figure 7: **Posterior-guided drafting.** A fixed draft budget prioritizes likely next-item continuations.

Opportunity:
Sibling Trajectories Guide Drafting.

Sibling trajectories in group $g = (q, e^-)$ sample the same request state. Their posterior $\widehat{p}(o \mid g)$ prioritizes continuations that led to likely next outcomes (Fig. 7).

Design: Posterior-weighted Draft-tree Shaping. For transition group g and recent suffix h, TraSURE builds a fixed-size draft tree biased toward outcome-relevant continuations.

1 Build Outcome-conditioned Continuation Pools. Let $\mathcal{H}$ contain records (g, o, x) from historical trajectories and completed siblings, where span x transitions from g to outcome o:

$$\mathcal{C}_{g,o} = \{\{x \mid (g, o, x) \in \mathcal{H}\}\}, \tag{8}$$

and add each completed sibling immediately, making its continuation available to later siblings.

2 Posterior-weighted Frontier Expansion. We suffix-match h against $\{\mathcal{C}_{g,o}\}_o$ to obtain candidates. For frontier node v, let $\pi(v)$ be its root path and $\mathcal{O}(v)$ the outcomes supporting it. We score v by

$$S_{\text{draft}}(v \mid g, h) = \sum_{o \in \mathcal{O}(v)} \underbrace{\widehat{p}(o \mid g)}_{\text{transition-group posterior}} \underbrace{P_{\text{token}}(\pi(v) \mid h; \mathcal{C}_{g,o})}_{\text{N-gram path score}}, \tag{9}$$

where P_{token} is the N-gram score from Eq. (2). The builder merges shared prefixes and expands the highest-scoring node until the budget is filled. The unchanged target verifier accepts only target-consistent tokens, so shaping the draft tree preserves the target-policy rollout distribution.

5 Evaluation

5.1 Experimental Setup

Models and Workloads. We evaluate Qwen3-4B/8B/14B (Yang et al., 2025) and Llama-3.1-8B (Grattafiori et al., 2024) on text-only MultiHop-RAG (Tang & Yang, 2024), and Qwen3-Omni-30B-A3B-Instruct (Xu et al., 2025) and MiniCPM-o 4.5 (Cui et al., 2026) on multimodal AVHaystacks (Chowdhury et al., 2025), CiteVQA (Ma et al., 2026), and MHDocVQA (Tanaka et al., 2025). Each query's trajectories access a shared corpus of cacheable items.

System and Measurement Setup. We implement TraSURE with slime (Zhu et al., 2025) and SGLang (Zheng et al., 2024), comparing with vanilla slime for speed and full-causal policies for quality. Controlled replay fixes recorded sequences while cache and scheduling use only causal history; on-policy runs measure live speed and quality. Appendix B details the setup.

5.2 Results

PIC Preserves Policy Quality. Because PIC changes attention semantics, we train PIC and full-causal policies independently from identical checkpoints. Fig. 8(a) compares them after matched training steps: both improve through roughly 80 steps and then stabilize. Using PIC for rollouts but full-causal attention for policy optimization creates a train-inference mismatch that causes quality to collapse, as does omitting RoPE rebasing. Across six models and four workloads, every final PIC score is within 1 pp of its independently trained full-causal counterpart (Fig. 8(b)).

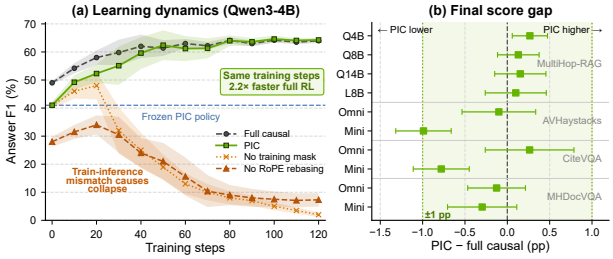

Figure 8: **PIC preserves policy quality.** (a) PIC matches full causal on MultiHop-RAG; mismatched attention collapses. At matched progress, TraSURE runs live full RL 2.2× faster. (b) PIC stays within 1 pp across workloads.

End-to-end Rollout Acceleration. Under replay, TraSURE accelerates rollout by 2.74×-5.82× across all model-workload pairs. Reported live rollout speedups range from 2.87× to 6.01×. With policy updates, replay and live full-RL speedups are 1.41×-1.98× and 1.38×-2.16×. Appendix Table 4 complements the relative results in Table 1 with averaged rollout time per prompt group under constrained KV-cache capacity.

Table 1: **End-to-end rollout and full-RL acceleration over vanilla slime.** All use $G = 16$ trajectories per prompt. Replay is fixed; Live is on-policy; Full RL includes policy-update cost.

Workload	Model	E2E rollout speedup ↑		Full RL speedup ↑	
		Replay	Live	Replay	Live
MultiHop-RAG	Qwen3-4B	5.08×	5.32×	1.98×	2.16×
	Qwen3-8B	4.82×	4.70×	1.79×	1.71×
	Qwen3-14B	5.82×	6.01×	1.91×	1.98×
	Llama-3.1-8B	3.49×	3.41×	1.87×	1.75×
AVHaystacks	Qwen3-Omni	4.41×	4.64×	1.47×	1.63×
	MiniCPM-o 4.5	3.09×	2.93×	1.50×	1.44×
CiteVQA	Qwen3-Omni	3.66×	3.81×	1.41×	1.38×
	MiniCPM-o 4.5	3.16×	3.24×	1.79×	1.85×
MHDocVQA	Qwen3-Omni	4.08×	4.26×	1.63×	1.68×
	MiniCPM-o 4.5	2.74×	2.87×	1.48×	1.61×

Component Ablation. Table 2 progressively enables the three components for all six models. Step is relative to the preceding row and Total to vanilla slime. PIC supplies the largest gain; scheduling and speculative decoding add complementary gains across all models and workloads.

Table 2: **Progressive rollout ablation across six models.** All results use $G = 16$ trajectories per prompt. Rows add components cumulatively to form the complete TraSURE design. Qwen3-Omni and MiniCPM-o report geometric means over three multimodal workloads. CD/FD denote current/future demand.

Configuration	Qwen3-4B		Qwen3-8B		Qwen3-14B		Llama-3.1-8B		Qwen3-Omni		MiniCPM-o	
	Step	Total	Step	Total	Step	Total	Step	Total	Step	Total	Step	Total
Vanilla slime	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×	–	1.00×
+ PIC	2.31×	2.31×	2.24×	2.24×	2.97×	2.97×	1.74×	1.74×	2.71×	2.71×	2.05×	2.05×
+ CD scheduling	1.37×	3.16×	1.45×	3.25×	1.24×	3.69×	1.47×	2.56×	1.16×	3.14×	1.29×	2.64×
+ FD lookahead	1.19×	3.76×	1.14×	3.71×	1.12×	4.13×	1.11×	2.84×	1.09×	3.42×	1.01×	2.67×
+ Posterior-guided SD	1.35×	5.08×	1.30×	4.82×	1.41×	5.82×	1.23×	3.49×	1.18×	4.04×	1.12×	2.99×

Table 3: **Matched-budget speculative decoding.**

Model	Accepted / verification		Speedup vs. no SD		Gain
	N-gram	Posterior	N-gram	Posterior	
Qwen3-4B	2.80	**3.90**	1.14×	**1.35**×	**1.18**×
Llama-3.1-8B	1.44	**1.92**	1.10×	**1.23**×	**1.12**×

Posterior-guided Speculative Decoding. At the same draft-tree budget, posterior guidance raises accepted tokens per verification by 39% for Qwen3-4B and 33% for Llama-3.1-8B. The improved acceptance yields 1.18× and 1.12× gains over N-gram speculative decoding, respectively.

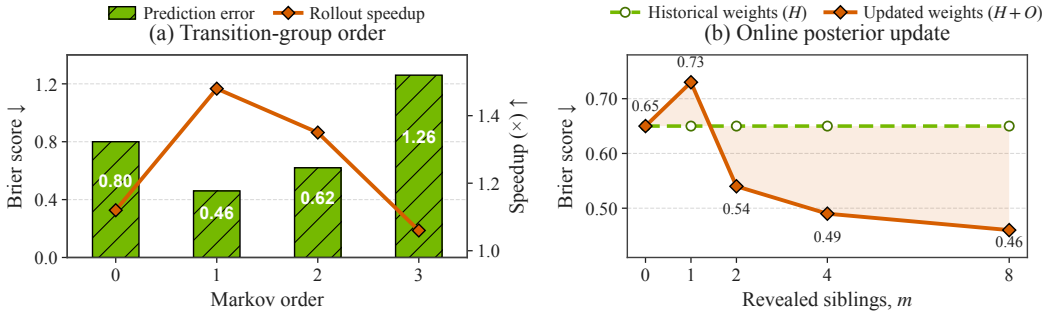

Figure 9: **Posterior ablations.** (a) Markov order. (b) Online sibling updates.

Posterior Ablations. Fig. 9(a) varies the number $k \in \{0, 1, 2, 3\}$ of recent items in the transition-group key. Order zero uses q; order one is our $g = (q, e^-)$ posterior for FD and SD. It gives 1.48× speedup over PIC + CD and the lowest Brier score (0.46), versus speedups of 1.12×, 1.35×, and 1.06× at orders zero, two, and three. Higher orders split the history into finer groups with fewer observations. Holding first-order grouping fixed, Fig. 9(b) isolates online sibling evidence. The two match at $m = 0$; after a transient increase at $m = 1$, updating lowers Brier score from 0.65 to 0.54, 0.49, and 0.46 at $m = 2$, 4, and 8, supporting online outcome-weight updates.

6 Conclusion

We present TraSURE, which treats cross-trajectory recurrence as a first-class systems opportunity for group-based on-policy RL. It combines position-independent item-KV caching, demand-aware scheduling, and posterior-guided speculative decoding. Controlled replay reaches up to 5.82× rollout speedup, while the live-rollout results reach up to 6.0×. Independently trained PIC policies remain within 1 percentage point of their full-causal counterparts across six models and four information-seeking workloads. These results demonstrate lower rollout cost with comparable policy quality on the evaluated workloads. This makes trajectory reuse practical for on-policy training.

AI USE STATEMENT

We used generative AI tools solely to check grammatical errors in the manuscript. We did not use these tools to generate code, figures, or other substantive content. We reviewed all AI-assisted edits and take full responsibility for the final content of this work.

ETHICS STATEMENT

This work studies the execution efficiency of information-seeking RL over shared corpora using existing benchmark workloads. Lower rollout costs can make agent development more accessible, but can also facilitate the scaling of harmful applications. The proposed acceleration techniques do not establish the safety, factual accuracy, or fairness of the resulting agents. Deployments should preserve access controls for external information, restrict cache sharing to authorized users and data, and account for privacy, bias, and applicable dataset and model usage restrictions.

REPRODUCIBILITY STATEMENT

We describe our method in Sections 4.1-4.3 and provide the experimental setup in Section 5.1, including the models, workloads, implementation details, hardware, and baselines. Additional evaluation details and ablation studies are presented in Section 5.2.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avi Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, pp. 9112–9141, 2024. URL https://openreview.net/forum?id=hSyW5go0v8.

Lorenz Brehme, Benedikt Dornauer, Thomas Ströhle, Maximilian Ehrhart, and Ruth Breu. Retrieval-augmented generation in industry: An interview study on use cases, requirements, challenges, and evaluation. In *Proceedings of the 17th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR*, pp. 110–122. SciTePress, 2025. doi: 10.5220/0013739500004000.

Jonathan D. Chang, Andrew Drozdov, Shubham Toshniwal, Owen Oertell, Alexander Trott, Jacob Portes, Abhay Gupta, Pallavi Koppol, Ashutosh Baheti, Sean Kulinski, Ivan Zhou, Irene Dea, Krista Opsahl-Ong, Simon Favreau-Lessard, Sean Owen, Jose Javier Gonzalez Ortiz, Arnav Singhvi, Xabi Andrade, Cindy Wang, Kartik Sreenivasan, Sam Havens, Jialu Liu, Peyton DeNiro, Wen Sun, Michael Bendersky, and Jonathan Frankle. KARL: Knowledge agents via reinforcement learning. *arXiv preprint arXiv:2603.05218*, 2026. doi: 10.48550/arXiv.2603.05218.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023a.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023b.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. ReSearch: Learning to reason with search for LLMs via reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025.

Nayoung Choi, Grace Byun, Andrew Chung, Ellie S. Paek, Shinsun Lee, and Jinho D. Choi. Reference-aligned retrieval-augmented question answering over heterogeneous proprietary documents. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 5626–5633. Association for Computing Machinery, 2025. doi: 10.1145/3746252.3761566.

Sanjoy Chowdhury, Mohamed Elmoghany, Yohan Abeysinghe, Junjie Fei, Sayan Nag, Salman H. Khan, Mohamed Elhoseiny, and Dinesh Manocha. MAGNET: A multi-agent framework for finding audio-visual needles by reasoning over multi-video haystacks. In *Advances in Neural Information Processing Systems*, volume 38, pp. 49255–49291, 2025. doi: 10.52202/085713-1648.

Junbo Cui, Bokai Xu, Chongyi Wang, Tianyu Yu, Weiyue Sun, Yingjing Xu, Tianran Wang, et al. MiniCPM-o 4.5: Towards real-time full-duplex omni-modal interaction, 2026. URL https://arxiv.org/abs/2604.27393.

Aaron Grattafiori et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl. *arXiv preprint arXiv:2508.18588*, 2025.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1582–1595, 2024. doi: 10.18653/v1/2024.naacl-long.88.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent context caching for serving large language models. *arXiv preprint arXiv:2410.15332*, 2024.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. SAM decoding: Speculative decoding via suffix automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12187–12204, 2025. doi: 10.18653/v1/2025.acl-long.595.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 7969–7992, 2023. doi: 10.18653/v1/2023.emnlp-main.495.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Minseo Kim, Minjae Lee, Seunghyuk Oh, Kevin Galim, Donghoon Kim, Coleman Hooper, Harman Singh, Amir Gholami, Hyung Il Koo, and Wonjun Kang. EfficientRollout: System-aware self-speculative decoding for RL rollouts. *arXiv preprint arXiv:2606.18967*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Chuanhao Li, Zhen Li, Chenchen Jing, Shuo Liu, Wenqi Shao, Yuwei Wu, Ping Luo, Yu Qiao, and Kaipeng Zhang. SearchLVLMs: A plug-and-play framework for augmenting large vision-language models by searching up-to-date internet knowledge. In *Advances in Neural Information Processing Systems*, volume 37, pp. 64582–64603, 2024. doi: 10.52202/079017-2060.

Bingshuai Liu, Ante Wang, Zijun Min, Liang Yao, Haibo Zhang, Yang Liu, Xu Han, Peng Li, Anxiang Zeng, and Jinsong Su. SPEC-RL: Accelerating on-policy reinforcement learning with speculative rollouts. *arXiv preprint arXiv:2509.23232*, 2025.

Shilong Liu, Hao Cheng, Haotian Liu, Hao Zhang, Feng Li, Tianhe Ren, Xueyan Zou, Jianwei Yang, Hang Su, Jun Zhu, Lei Zhang, Jianfeng Gao, and Chunyuan Li. LLaVA-Plus: Learning to use tools for creating multimodal agents. In *Computer Vision – ECCV 2024*, volume 15105 of *Lecture Notes in Computer Science*, pp. 126–142. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-72970-6_8.

Dongsheng Ma, Jiayu Li, Zhengren Wang, Yijie Wang, Jiahao Kong, Weijun Zeng, Jutao Xiao, Jie Yang, Wentao Zhang, Bin Wang, and Conghui He. CiteVQA: Benchmarking evidence attribution for trustworthy document intelligence, 2026. URL https://arxiv.org/abs/2605.12882.

Zelei Shao, Vikranth Srivatsa, Sanjana Srivastava, Qingyang Wu, Alpay Ariyak, Xiaoxia Wu, Ameen Patel, Jue Wang, Percy Liang, Tri Dao, Ce Zhang, Yiying Zhang, Ben Athiwaratkun, Chenfeng Xu, and Junxiong Wang. Beat the long tail: Distribution-aware speculative decoding for RL training. In *Proceedings of Machine Learning and Systems*, volume 8, pp. 1626–1638, 2026.

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.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-Searcher: Incentivizing the search capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063.

Ryota Tanaka, Taichi Iki, Taku Hasegawa, Kyosuke Nishida, Kuniko Saito, and Jun Suzuki. VDocRAG: Retrieval-augmented generation over visually-rich documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 24827–24837, 2025.

Yixuan Tang and Yi Yang. MultiHop-RAG: Benchmarking retrieval-augmented generation for multi-hop queries. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=t4eB3zYWBK.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 10014–10037, 2023. doi: 10.18653/v1/2023.acl-long.557.

Vime Contributors. Vime: An LLM post-training framework with vLLM for RL scaling. https://github.com/vllm-project/vime, 2026. GitHub repository; accessed 2026-08-26.

vLLM Project. vLLM features. https://docs.vllm.ai/en/latest/features/, 2026. Accessed: 2026-08-25.

Jin Xu, Zhifang Guo, Hangrui Hu, Yunfei Chu, Xiong Wang, Jinzheng He, Yuxuan Wang, et al. Qwen3-Omni technical report, 2025. URL https://arxiv.org/abs/2509.17765.

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

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the twentieth European conference on computer systems*, pp. 94–109, 2025. doi: 10.1145/3689031.3696098.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru 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, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244. Curran Associates, Inc., 2025. doi: 10.52202/085713-3775.

Zhiyuan Zhai and Xin Wang. Selective Rollout: Mid-trajectory termination for multi-sample agent RL. *arXiv preprint arXiv:2605.05802*, 2026.

Zhi Zhang, Zhen Han, Costas Mavromatis, Qi Zhu, Yunyi Zhang, Sheng Guan, Dingmin Wang, Xiong Zhou, Shuai Wang, Soji Adeshina, Vassilis Ioannidis, and Huzefa Rangwala. Train less, learn more: Adaptive efficient rollout optimization for group-based reinforcement learning. *arXiv preprint arXiv:2602.14338*, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H. Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583, 2024. doi: 10.52202/079017-2000.

Yuzhen Zhou, Jiajun Li, Yusheng Su, Gowtham Ramesh, Zilin Zhu, Xiang Long, Chenyang Zhao, Jin Pan, Xiaodong Yu, Ze Wang, Kangrui Du, Jialian Wu, Ximeng Sun, Jiang Liu, Qiaolin Yu, Hao Chen, Zicheng Liu, and Emad Barsoum. APRIL: Active partial rollouts in reinforcement learning to tame long-tail generation. *arXiv preprint arXiv:2509.18521*, 2025.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An LLM post-training framework for RL scaling. https://github.com/THUDM/slime, 2025. GitHub repository; accessed 2026-08-26.

A END-TO-END TRASURE ROLLOUT ALGORITHM

Algorithm 1 gives the online rollout loop. After PIC prefill of batch B, POSTERIORDRAFTTREE (line 9) suffix-matches the pools in Eq. (8) and expands the highest-scoring frontier nodes under Eq. (9) within budget D. The backend batches VERIFYANDDECODE (line 10) across running requests, producing spans x_u ending in retrieval actions or final answers. AWAITNEXTCOMPLETION returns one completed request and its span at a time. Only revealed outcomes update the posterior, which guides scheduling and drafting; current-policy verification preserves the rollout distribution.

Algorithm 1 End-to-end TraSURE group rollout

Require: prompts Q; trajectories per prompt G; policy π_{θ_v}; historical transition counts H; continuation pools $\{\mathcal{C}_{g,o}\}$ of generated spans; PIC capacity K; batch limit b; draft-tree budget D
Ensure: completed trajectory groups $\{\mathcal{T}(q) : q \in Q\}$
State: W: waiting queue; R: running queue; C: items with cached KV states; $O_{g,o}$: current-collection transition counts from group g to outcome o (an item or `STOP`).
Request: $u = (q, i, r, g)$ records prompt q, trajectory index i, context prefix r, and group $g = (q, e^-)$, where e^- is the latest retrieved item or `START` before any retrieval.
Output: x_u is the policy-generated token span returned when request u completes.

1: $W \leftarrow \{(q, i, [q], (q, \texttt{START})) : q \in Q, 0 \le i < G\}$
2: $R \leftarrow \varnothing$, $C \leftarrow \varnothing$, $O_{g,o} \leftarrow 0$ for all observed (g, o)
3: **while** $W \neq \varnothing$ or $R \neq \varnothing$ **do**
4: **if** $W \neq \varnothing$ and rollout slots are available **then**
5: $B \leftarrow$ DEMANDAWAREADMIT$(W, R, C, H + O, K, b)$; remove B from W ▷ Algorithm 2
6: PICPREFILL(B, C, K, θ_v) ▷ Algorithm 3
7: **for all** $u = (q, i, r, g) \in B$ **do**
8: $h \leftarrow$ recent token suffix of r
9: $T_D \leftarrow$ POSTERIORDRAFTTREE$(g, h, H + O, \{\mathcal{C}_{g,o}\}, D)$
10: launch $x_u \leftarrow$ VERIFYANDDECODE(π_{θ_v}, r, T_D); add u to R
11: **end for**
12: **end if**
13: $(u = (q, i, r, g), x) \leftarrow$ AWAITNEXTCOMPLETION(R) ▷ $x = x_u$ for completed u
14: $o \leftarrow$ EXECUTE(x) ▷ o is a returned item or `STOP`
15: $O_{g,o} \leftarrow O_{g,o} + 1$; insert x into $\mathcal{C}_{g,o}$
16: remove u from R and release its references to shared PIC entries
17: **if** $o =$ `STOP` **then**
18: append x to $\tau_i(q)$ and mark the trajectory complete
19: **else**
20: append (x, o) to $\tau_i(q)$ and enqueue $(q, i, [r, x, o], (q, o))$ into W
21: **end if**
22: **end while**
23: **return** $\{\mathcal{T}(q) : q \in Q\}$

Algorithm 2 first shortlists feasible requests by marginal work (Eq. (6)), then selects by expected future reuse; candidates may have different current costs. Algorithm 3 combines one writer per missing item and shared KV slots with the mask and RoPE rebasing in Eqs. (4) and (5). Protection begins before allocation and ends only after the last active consumer finishes.

Algorithm 2 Demand-aware batch admission

procedure DEMANDAWAREADMIT(W, R, C, N, K, b)
 $B \leftarrow \varnothing$
 while $|B| < b$ and a feasible request remains in $W \setminus B$ **do**
 for all $u \in W \setminus B$ feasible under capacity K **do**
 $\Delta(u \mid B, C) \leftarrow \sum_{e \in \mathcal{E}(u) \setminus (C \cup \mathcal{E}(B))} L_e$
 $F(u) \leftarrow \sum_{\rho \in R} \sum_{e \in \mathcal{E}(u) \setminus (C \cup \mathcal{E}(B))} \widehat{p}_N(e \mid g_\rho) L_e$
 end for
 form a shortlist $\mathcal{A} \subseteq W \setminus B$ of feasible requests with low $\Delta(u \mid B, C)$
 $u^* \leftarrow \arg\max_{u \in \mathcal{A}} F(u)$; $B \leftarrow B \cup \{u^*\}$ ▷ ties: lower Δ
 end while
 return B
end procedure

Algorithm 3 Position-independent item prefill

procedure PICPREFILL(B, C, K, θ_v)
  $U \leftarrow \bigcup_{u \in B} \mathcal{E}(u)$
  protect $U \cap C$; evict only unprotected entries using current and posterior demand
  for all $e \in U \setminus C$ **in parallel do**
    reserve one shared physical KV entry for e and elect one request as writer
    writer prefills e under the position-isolated mask; followers wait for `ready`
    publish the entry in C; map every occurrence of e to its shared KV slots
  end for
  prefill request suffixes, rebasing each shared item's RoPE keys at attention time
end procedure

B Experimental Details

B.1 Models and Workloads

We evaluate six models on four information-seeking workloads over shared corpora: Qwen3-4B/8B/14B (Yang et al., 2025) and Llama-3.1-8B (Grattafiori et al., 2024) on text-only MultiHop-RAG (Tang & Yang, 2024); Qwen3-Omni-30B-A3B-Instruct (Xu et al., 2025) and MiniCPM-o 4.5 (Cui et al., 2026) on multimodal AVHaystacks (Chowdhury et al., 2025), CiteVQA (Ma et al., 2026), and MHDocVQA (Tanaka et al., 2025). Each query's trajectories share access to a corpus of cacheable external items, including text passages and multimodal objects.

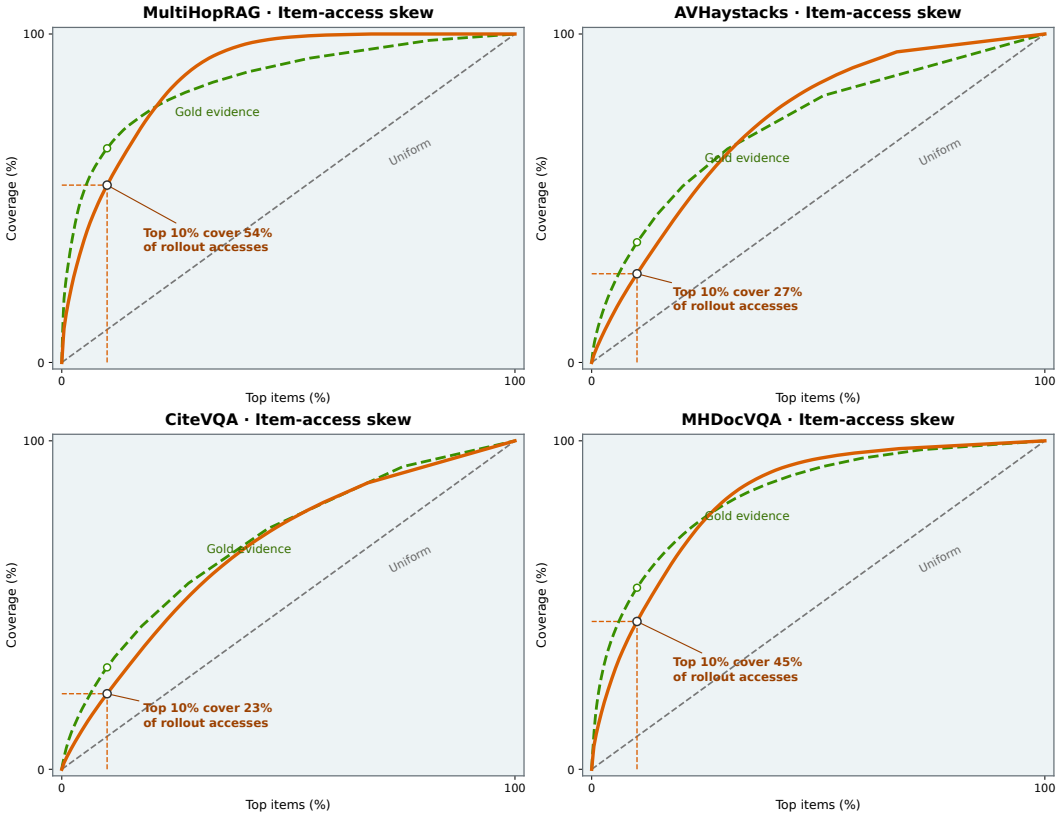

Figure 10: **Item-access skew across workloads.** Cumulative coverage of rollout item accesses and gold evidence as items are ranked by access frequency.

B.2 System Setup and Baselines

We implement TraSURE in slime v0.3.0 (Zhu et al., 2025), using SGLang (Zheng et al., 2024) for rollouts and the matching PIC mask during policy optimization. Vanilla slime is a strong per-

formance baseline: its high-performance SGLang rollout backend already provides radix caching. Our system modifies the same framework and backend; paired runs use the same host and settings and include all overheads. Independently trained full-causal policies are the quality baseline. Qwen3-4B/8B and Llama-3.1-8B run on 8× RTX 6000 Ada GPUs; Qwen3-14B, Qwen3-Omni, and MiniCPM-o 4.5 run on 8× RTX PRO 6000 Blackwell GPUs.

B.3 Measurement Protocol

To isolate system and component speedups from trajectory variation, we use causal trace replay. We collect trajectories with vanilla slime and replay the same token, action, and observation sequence in every system. Model forwards, cache management, and scheduling execute normally, while each trajectory advances using its recorded token or observation. Posterior estimates and speculative drafts use only causally revealed history, never future trace events. Thus, all systems process identical trajectories, but their cache states and execution orders evolve according to their own mechanisms. To evaluate policy quality, we instead conduct standard on-policy training: each system starts from the same policy checkpoint and prompts, then independently samples and executes its own trajectories without trace forcing. We also measure rollout speed during these live runs. Table 1 reports end-to-end speedup under controlled replay and live execution; differences are expected because independent sampling changes the workload, while the quality results in Section 5.2 come from the on-policy training runs rather than from the controlled replay traces used for timing.

B.4 Absolute Rollout Time

Across both $G = 8$ and $G = 16$, TraSURE consistently accelerates rollout across different trajectory counts per prompt. RadixCache spends most of its rollout time in prefill, highlighting repeated prefix computation as its dominant bottleneck.

Table 4: **Averaged replay rollout time per prompt group.** Controlled replay fixes trajectory content across systems; G denotes the number of trajectories sampled per prompt.

Workload	Model	G	Averaged rollout time per prompt group (s)		Speedup	RadixCache time share		
			RadixCache	TraSURE		Prefill	Decode	Others
MultiHop-RAG	Qwen3-4B	8	20.87	4.53	4.61×	86.4%	11.2%	2.4%
		16	41.02	8.08	5.08×	85.3%	12.9%	1.8%
	Qwen3-8B	8	30.98	7.06	4.39×	77.4%	19.8%	2.8%
		16	61.43	12.74	4.82×	82.0%	15.5%	2.5%
	Qwen3-14B	8	57.58	11.70	4.92×	79.2%	19.2%	1.6%
		16	107.47	18.46	5.82×	85.4%	13.1%	1.5%
	Llama-3.1-8B	8	35.83	9.84	3.64×	80.1%	17.0%	2.9%
		16	56.68	16.23	3.49×	76.7%	19.8%	3.5%
AVHaystacks	Qwen3-Omni	8	3.49	0.84	4.17×	67.0%	25.3%	7.7%
		16	6.77	1.53	4.41×	72.9%	20.6%	6.5%
	MiniCPM-o 4.5	8	3.80	1.29	2.94×	65.8%	22.5%	11.7%
		16	7.34	2.37	3.09×	63.5%	28.5%	8.0%
CiteVQA	Qwen3-Omni	8	21.23	6.12	3.47×	63.1%	24.6%	12.3%
		16	45.34	12.39	3.66×	66.9%	23.1%	10.0%
	MiniCPM-o 4.5	8	9.45	3.21	2.94×	74.2%	18.0%	7.8%
		16	18.31	5.79	3.16×	79.2%	10.8%	10.0%
MHDocVQA	Qwen3-Omni	8	33.64	8.69	3.87×	77.1%	15.1%	7.8%
		16	68.59	16.81	4.08×	80.7%	12.2%	7.1%
	MiniCPM-o 4.5	8	5.61	2.17	2.59×	73.1%	21.5%	5.4%
		16	10.62	3.87	2.74×	80.9%	12.4%	6.7%