TVCACHE: A Stateful Tool-Value Cache for Post-Training LLM Agents

Abhishek Vijaya Kumar* · Bhaskar Kataria* · Byungsoo Oh · Emaad Manzoor · Rachee Singh
* Equal contribution

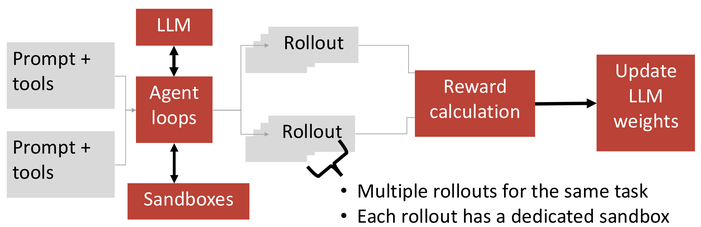
Cornell University

ICML International Conference On Machine Learning

TLDR: TVCACHE safely reuses repeated tool calls and sandboxes in RL post-training to reduce tool-call latency by 3.4×–6.9× without hurting training reward.

Tool calls dominate agentic RL rollouts with repeated tool calls across rollouts. Existing caches lead to incorrect behavior as they fail to capture the sandbox state.

RL post-training repeatedly executes expensive tool calls

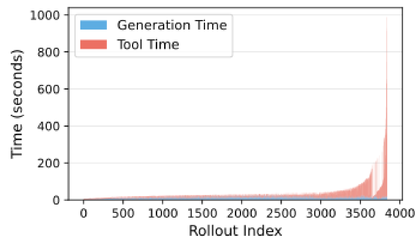


- Multiple rollouts for the same task
- Each rollout has a dedicated sandbox

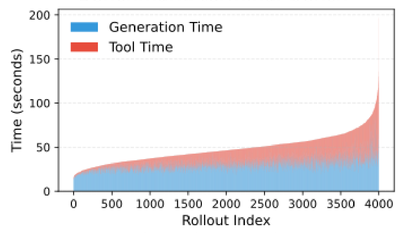
- RL post-training improves agents by generating **multiple rollouts for the same task**.
- Each rollout alternates between **reasoning tokens** and **tool calls**.
- Tool calls run in **isolated sandboxes** to support code execution, file edits, web search, and API calls.
- This makes rollout generation expensive: **tokens cost GPU time, tools add latency, and GPUs wait for tool responses**.

Takeaway: Tool-using RL agents repeatedly pay the cost of sandboxed tool execution.

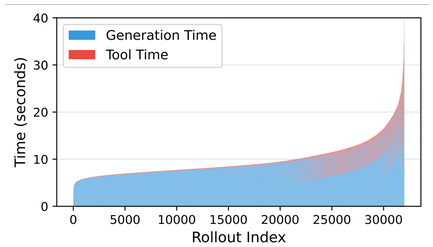
Tool execution is a major rollout bottleneck



Benchmark: terminal-bench



Benchmark: EgoSchema

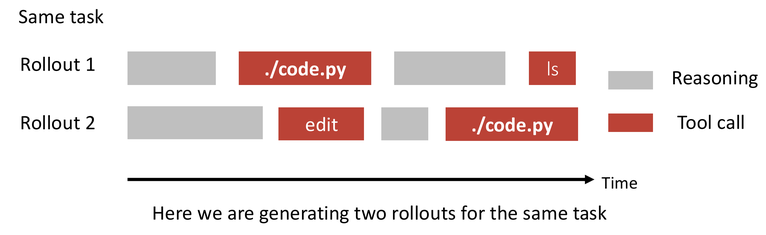


Benchmark: SkyRL-SQL

- Across terminal-bench, SkyRL-SQL, and EgoSchema, tool execution consumes **7–43% of rollout time**.
- This leaves GPUs waiting for tool responses during RL post-training.

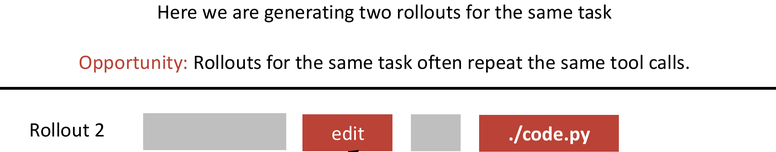
Takeaway: Reducing repeated tool execution can directly speed up rollout generation.

Same tool calls arepeat cross rollouts – can we reuse them?



Here we are generating two rollouts for the same task

Opportunity: Rollouts for the same task often repeat the same tool calls.

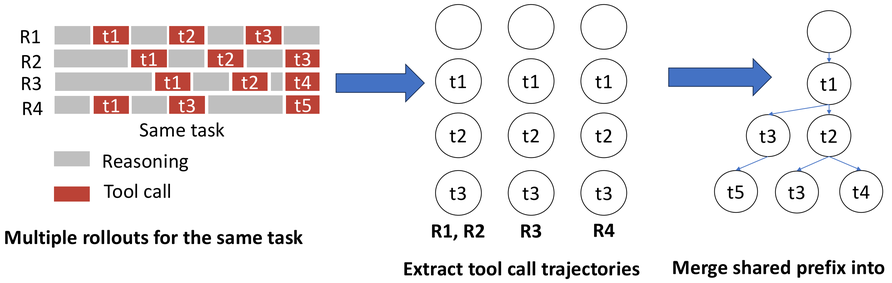


Challenge: Tool calls can modify sandbox state. In Rollout 2, edit mutates the files before ./code.py executes.

Key Insight: **Safe reuse of the tool call outputs requires a stateful cache**.
Stateless naive cache results in about 27% of incorrect cache hits

Our Solution: TVCache: A Stateful Tool-Value cache

Solution: Cache tool calls by the trajectory



Multiple rollouts for the same task

Extract tool call trajectories

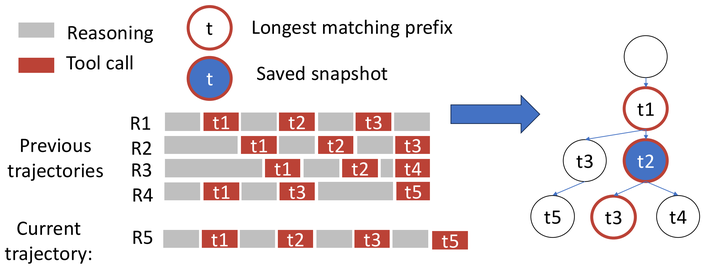
Merge shared prefix into a TCG

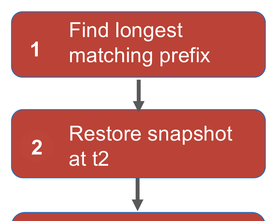
Solution: Cache tool calls by trajectory

- TVCACHE builds a **Tool Call Graph (TCG)** for each task.
- Each node represents a **tool call** and stores the corresponding **tool response**.
- Shared prefixes across rollouts are stored once.
- TVCache reuses a cached response only when the **tool-call history matches**.

Takeaway: Cache key = current tool call + preceding trajectory.

Cache misses restore the closest reusable sandbox

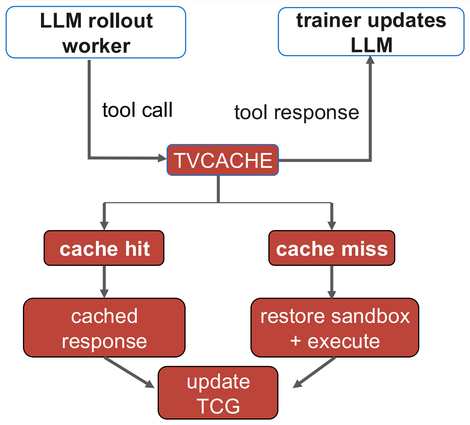




- On a cache miss, TVCache finds the **longest matching prefix** in the TCG.
- If that node has a saved snapshot, TVCache **restores the sandbox** from that point.
- TVCache then executes only the **unmatched suffix** of the trajectory.
- Snapshots are stored **selectively (**when restoring is cheaper than re executing)

Takeaway: Cache misses reuse the closest available state instead of restarting from scratch.

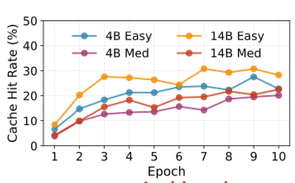
TVCache is transparent to rollout generation



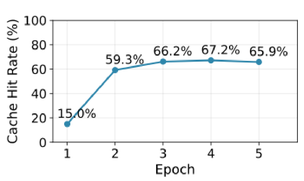
Takeaway: TVCache speeds up tool use without changing the RL training loop.

Results

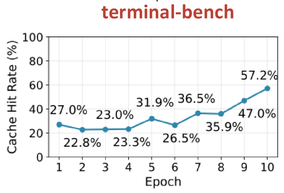
Repeated trajectories create cache hits



terminal-bench



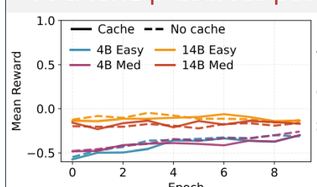
EgoSchema



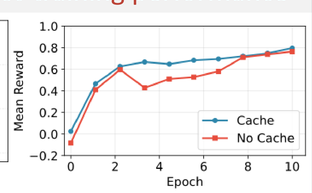
SkyRL-SQL

Across workloads, TVCACHE turns repeated tool-call trajectories into cache hits, reaching up to 67% cache hit rate

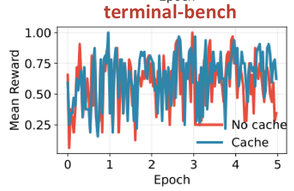
TVCACHE preserves post-training performance



terminal-bench



EgoSchema



SkyRL-SQL

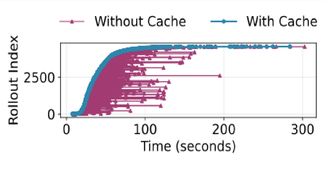
TVCACHE closely matches the **no-cache baseline** across benchmarks, showing that trajectory-based reuse improves efficiency without sacrificing training quality.

Tool call time speedup

Model	Tasks	Median tool call speedup
Qwen-4B	Easy	6.18x
Qwen-4B	Medium	6.92x
Qwen-14B	Easy	3.44x
Qwen-14B	Medium	5.55x

Median tool call time reduction for terminal-bench

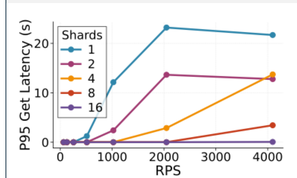
TVCACHE reduces median tool-call time by **3.44×–6.92×** across model sizes and task difficulties, showing that repeated tool executions can be substantially reused.



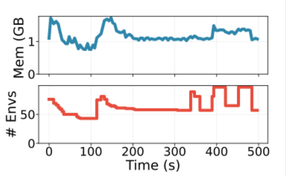
Rollout generation time for EgoSchema

With TVCACHE, rollouts complete earlier and with lower tail latency, indicating that cached tool responses reduce waiting time during rollout generation.

Scalability and memory overhead



TVCACHE scales with sharding
Because each task's TCG is independent, TVCACHE shards cache servers by task ID. With 16 shards, TVCACHE sustains **4096 RPS** while keeping P95 GET latency low at **6.1 ms**.



Memory overhead with TVCache
Memory stays around **1 GB** and peaks near **2 GB**, even while proactively launching **32 containers** for the next step. The gradual increase comes from cached sandbox snapshots in the TCG.