Efficient Reinforcement Learning for Long-Horizon Tool-Use Agentic Tasks

Zelei Cheng* **Amritansh Mishra***
Sambit Sahu **William Campbell**
AI Foundations, Capital One
{zelei.cheng, amritansh.mishra, sambit.sahu, william.campbell}@capitalone.com

arXiv:2608.10357v1 [cs.LG] 11 Aug 2026

Abstract

Long-horizon tool-using agents must reason over user goals, domain policies, tool calls, simulator state, and delayed verifiable rewards. Reinforcement learning (RL) is a natural fit for this setting, but multi-turn on-policy rollouts create long contexts, while model-specific attention layers may require custom masks and learned sink normalization. We present SINKFLEX-RL, a modular training system for RL in dual-control tool-use environments. The system combines a Gymnasium-compatible environment wrapper, a VERL-style rollout dataflow, group-relative policy optimization without a separate value model, and a sink-aware FlexAttention path designed to preserve model-specific sink scaling under causal and sliding-window masks. In a preliminary τ^2-Bench retail run, validation reward (mean@1) rises from 0.25 early in training to 0.44 later in the observed training window, while training-score and trajectory-reward proxies also trend upward. In a fixed-configuration memory benchmark, the optimized attention path reduces peak VRAM from 28.06 GB to 22.52 GB at 4096 tokens, a 19.7% reduction, and runs the measured 8192-token configuration using 25.53 GB where the eager baseline runs out of memory. These results illustrate the value of integrating environment interfaces, RL dataflow, and attention-kernel design for memory-feasible long-horizon agent training.

1 Introduction

Language-model agents increasingly operate in interactive environments: they converse with users, inspect and update state through tools, follow domain policies, and receive sparse success signals only after long trajectories. Benchmarks such as τ-Bench and τ^2-Bench formalize this setting by evaluating agents in customer-service-style domains with tool APIs, policy constraints, and simulated users (Yao et al., 2024; Barres et al., 2025). These settings stress both reasoning quality and training-system capacity. A single trajectory may contain many dialogue turns, tool calls, and environment observations; on-policy RL multiplies this cost by requiring fresh rollouts, reward checking, and repeated policy updates.

This paper studies the systems side of memory-feasible agentic RL for a large open-weight mixture-of-experts (MoE) transformer. MoE architectures can increase model capacity at lower per-token feed-forward cost by routing tokens to a subset of experts (Fedus et al., 2022), but they do not remove the long-context attention bottleneck. As multi-turn trajectories grow, eager attention can exhaust high-bandwidth memory before backpropagation. Our current system evaluation covers sequence lengths through 8192 tokens; longer target workloads motivate the design but are not evaluated here. Existing fused attention kernels improve memory movement and throughput (Dao et al., 2022; Dao, 2024), but production models may require attention variants with learned sink parameters, heterogeneous masks, or backward behavior that is not exposed by a fixed kernel interface.

We therefore treat agentic RL as a systems-integration problem and make three contributions. First, we wrap dual-control environments behind a Gymnasium-style interface so that

*Equal contribution.

rollouts, tools, user simulators, and reward checkers can connect to a standard RL dataflow. Second, we use group-relative policy optimization (GRPO) to update the actor from groups of trajectories without training a separate value model, following the memory-motivated rationale introduced in DeepSeekMath (Shao et al., 2024) and a VERL-style post-training dataflow (Sheng et al., 2024). Third, we implement a sink-aware FlexAttention path that composes causal and sliding-window masks with differentiable sink scaling while avoiding eager attention-state materialization. We evaluate the integrated pipeline with a preliminary τ^2-Bench retail training run and a peak-memory study through 8192 tokens. The core contribution of this work is the integration of these components and the resulting memory feasibility.

2 Related Work

Agentic benchmarks. Evaluating language models as interactive agents requires environments in which models issue API calls, manage changing state, and process multi-turn feedback. AgentBench (Liu et al., 2024) motivated broad interactive evaluation, while WebArena (Zhou et al., 2024), GAIA (Mialon et al., 2024), and SWE-bench (Jimenez et al., 2024) target web navigation, general assistant capabilities, and software engineering, respectively. For stateful tool use, τ-Bench evaluates executable state changes rather than text-only judgments (Yao et al., 2024). We use τ^2-Bench (Barres et al., 2025), whose dual-control setting allows both the agent and a simulated user to influence task progress. This provides a representative testbed for gathering missing information, recovering from invalid actions, and maintaining consistency over multi-turn interactions.

RL fine-tuning for multi-step tool use. In complex tool-use environments, annotating every intermediate action is costly, which motivates training with programmatically verifiable episode-level rewards (Zheng et al., 2025; Dong et al., 2026). PPO (Schulman et al., 2017) commonly uses a learned value function, adding memory and compute overhead for large language models. GRPO (Shao et al., 2024) removes the separate value model and normalizes rewards within groups of sampled outputs. Frameworks such as `verl` (Sheng et al., 2024) organize rollout generation, reward computation, and policy updates as a distributed post-training dataflow. Our work adopts these algorithmic and systems components and focuses on integrating them with long-context, model-compatible attention for multi-turn tool-use rollouts.

Attention mechanisms for long-context training. Transformers use attention as their core sequence-mixing operation (Vaswani et al., 2017), but dense attention has quadratic memory and compute in sequence length. FlashAttention and FlashAttention-2 reduce memory traffic by tiling exact attention and avoiding materialized attention matrices (Dao et al., 2022; Dao, 2024). StreamingLLM shows that retaining attention sinks can help stabilize windowed long-context inference (Xiao et al., 2024). PyTorch FlexAttention exposes a programmable interface for composing masks and score modifications (PyTorch Contributors, 2026). SINKFLEX-RL combines these ideas by treating efficient attention as both a memory problem and a model-compatibility problem when sink-aware normalization and heterogeneous masks must remain in the differentiable training path.

3 Agentic RL Setting

Dual-control environment. In single-control tool-use benchmarks, the agent is often the only actor that changes the environment through tools while the user supplies information passively. In dual-control environments, both sides can affect the world state: the agent chooses dialogue and API actions, while the simulated user can confirm choices, provide missing information, or take user-side actions. A task instance can be represented as

$$\mathcal{E} = (g, \pi_{\text{SOP}}, \mathcal{A}_{\text{tool}}, s_0, u), \tag{1}$$

where g is the user goal, π_{SOP} denotes domain-policy or standard-operating-procedure constraints, $\mathcal{A}_{\text{tool}}$ is the agent tool set, s_0 is the initial shared state, and u is the user simulator.

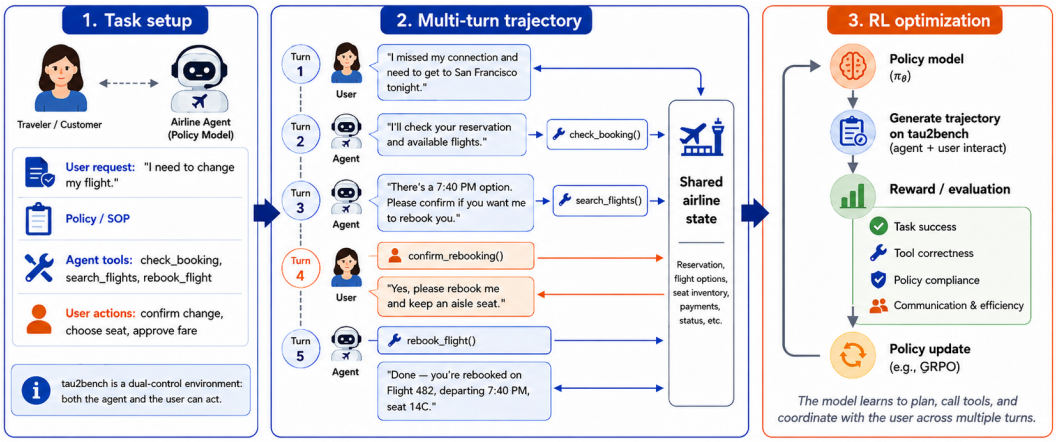

Figure 1: A dual-control agentic RL episode. A task specification initializes the environment, user–agent interaction produces a multi-turn trajectory, and a programmatic checker scores the resulting rollout group for a GRPO-style policy update.

A trajectory is

$$\tau = \{(o_t, a_t, r_t, d_t, i_t)\}_{t=1}^{T}, \tag{2}$$

where o_t is an observation, a_t is an agent action, r_t is a reward or diagnostic signal, d_t is a termination flag, and i_t contains metadata. The trainer consumes a trajectory-level reward R_i produced by the benchmark's programmatic checker. Other dashboard diagnostics are logged separately and are not outputs from a separately trained critic network.

Why this setting stresses the training system. The environment creates three systems pressures. First, rewards are delayed and often verifiable only after the final state is checked, increasing the number of sampled tokens per useful gradient. Second, trajectories are multi-turn and tool-heavy, so the policy must retain user messages, tool outputs, and domain-policy constraints in context. Third, the system must coordinate environment execution, rollout generation, reward checking, and policy optimization while maintaining consistent episode state and policy versions.

Figure 1 summarizes the dual-control task structure used in this work: a task specification initializes a shared world state, the user and agent take alternating actions, and the resulting trajectory is scored by a programmatic checker before the rollout group is used for a GRPO-style update.

4 Training Pipeline

Figure 2 summarizes the training dataflow. We describe the components abstractly to avoid dependence on deployment-specific infrastructure.

Environment wrapper. Each benchmark domain is exposed through a common reset/step interface. On reset, the wrapper samples a task, initializes the shared state, and constructs the first policy observation. On step, it routes model outputs to one of three handlers: natural-language response, tool call, or termination. The wrapper then advances the user simulator and state backend and returns the next observation together with reward metadata. This separation keeps benchmark-specific user simulation, tool behavior, and checking logic outside the trainer and is intended to support additional executable environments through the same interface.

Rollout worker. The rollout worker owns the agent loop. At each turn, it formats the observation, samples from the current policy π_θ, parses the response into an action, and

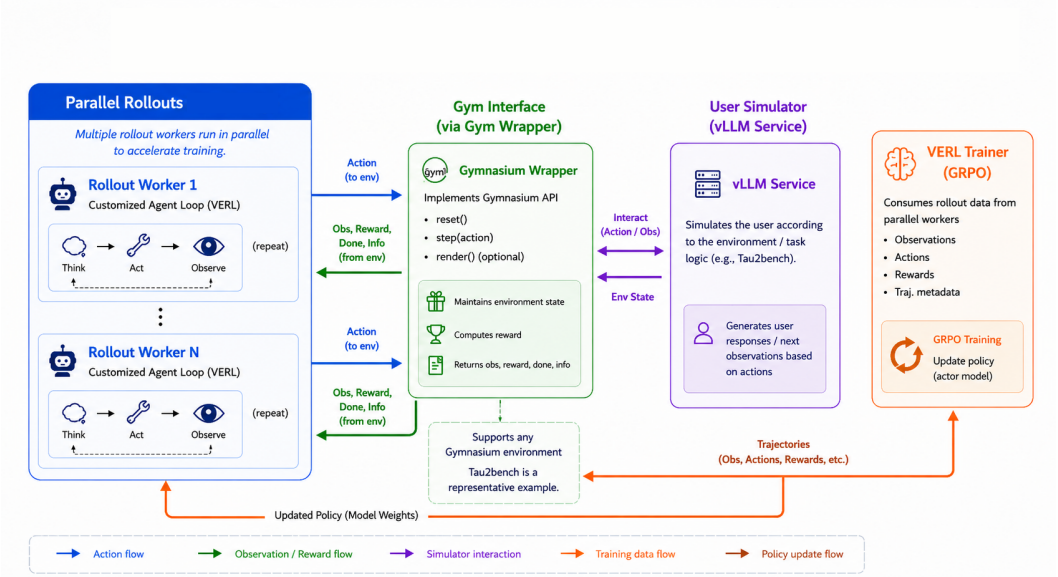

Figure 2: Training-system dataflow. The actor samples multi-turn trajectories through rollout workers and a Gymnasium-compatible wrapper; trajectory rewards and metadata flow to a GRPO trainer, which updates and redistributes policy weights.

appends the transition to the trajectory buffer. The worker repeats until the environment terminates or a maximum-turn budget is reached. The trainer receives token-level log probabilities, action masks, trajectory rewards, and episode metadata without directly depending on benchmark-specific parsing or tool implementations.

Policy update. Given a prompt or task context x and a group of G sampled rollouts $y_1, \ldots, y_G$ from the old policy, GRPO computes a group-normalized trajectory advantage

$$\hat{A}_i = \frac{R_i - \mu(R_{1:G})}{\sigma(R_{1:G}) + \epsilon_A}, \tag{3}$$

where R_i is the programmatically computed trajectory reward and ϵ_A is a small numerical constant. Under outcome-level supervision, the same normalized advantage is assigned to every optimized token in rollout y_i. For token $y_{i,t}$, the importance-sampling ratio is

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t} \mid x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid x, y_{i,<t})}. \tag{4}$$

Define the clipped ratio

$$\bar{\rho}_{i,t}(\theta) = \text{clip}(\rho_{i,t}(\theta), 1 - \epsilon_c, 1 + \epsilon_c)\,. \tag{5}$$

The loss minimized by the trainer is

$$\mathcal{J}_{i,t}(\theta) = \min\left(\rho_{i,t}(\theta)\hat{A}_i, \bar{\rho}_{i,t}(\theta)\hat{A}_i\right), \tag{6}$$

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{\sum_i T_i} \sum_{i=1}^{G} \sum_{t=1}^{T_i} \mathcal{J}_{i,t}(\theta) + \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{7}$$

where T_i is the optimized token length of rollout i, ϵ_c is the clipping radius, and the KL term regularizes the policy toward the reference model. This is the minimization form of the usual PPO/GRPO clipped maximization objective (Schulman et al., 2017). The baseline design does not train a separate critic or value network.

5 Sink-Aware FlexAttention

Memory wall. A trajectory of length n induces an $n \times n$ attention score structure for each relevant batch/head component when implemented eagerly. The number of score positions therefore grows quadratically with context length; for example, $n = 20{,}000$ corresponds to 4×10^8 positions before accounting for activations, optimizer state, MoE routing, or rollout batching. Fused attention methods address this by computing attention in tiles and reducing high-bandwidth-memory traffic (Dao et al., 2022). A model-specific attention layer, however, may also require custom score modifications, mixed causal and sliding-window masks, or learned sink logic.

Attention sinks. Attention sinks are tokens or learned mechanisms that absorb attention mass and can stabilize long-context behavior. Prior work shows that retaining initial sink tokens can recover quality under windowed attention in streaming inference (Xiao et al., 2024). Our setting differs from inference-only KV-cache management: the sink behavior is part of the trainable attention computation and therefore must remain connected to the forward and backward paths.

Zero-value-sink equivalence. To avoid materializing an explicit sink token in the key and value caches, we use the algebraic form of a zero-valued sink. Let s_η be a learned sink logit and let its value vector be $v_{\text{sink}} = \mathbf{0}$. For a query q, the attention output with the explicit sink is

$$O_{\text{sink}} = \frac{\sum_i \exp(q \cdot k_i) v_i + \exp(s_\eta)\mathbf{0}}{\sum_i \exp(q \cdot k_i) + \exp(s_\eta)} \tag{8}$$

$$= \left(\frac{\sum_i \exp(q \cdot k_i)}{\sum_i \exp(q \cdot k_i) + \exp(s_\eta)} \right) O_{\text{std}}. \tag{9}$$

Writing $\ell = \log \sum_i \exp(q \cdot k_i)$ gives

$$\alpha_{\text{sink}} = \frac{\exp(\ell)}{\exp(\ell) + \exp(s_\eta)} = \sigma(\ell - s_\eta), \qquad O_{\text{sink}} = \alpha_{\text{sink}} O_{\text{std}}. \tag{10}$$

Under the zero-value assumption, explicit sink materialization is therefore algebraically equivalent to scaling the standard attention output by a factor computed from the log-sum-exp statistic.

```python
import torch
import torch.nn.functional as F

# 1. Explicit zero-value sink materialization
scores_with_sink = torch.cat([sink_score, scores], dim=-1)
v_with_sink = torch.cat([torch.zeros_like(v[:, :, :1, :]), v], dim=-2)
out_explicit = F.softmax(scores_with_sink, dim=-1) @ v_with_sink

# 2. Equivalent output scaling
out_std = F.softmax(scores, dim=-1) @ v
lse = torch.logsumexp(scores, dim=-1, keepdim=True)
out_scaled = out_std * torch.sigmoid(lse - sink_score)

assert torch.allclose(out_explicit, out_scaled, atol=1e-6)
```

Figure 3: Illustration of the zero-value-sink identity. The production implementation uses fused attention statistics rather than materializing the dense score tensor shown in this small reference example.

Implementation pattern. The implementation uses FlexAttention as a programmable attention substrate (PyTorch Contributors, 2026). A mask function composes causal constraints with an optional sliding-window constraint and an always-visible prefix. The mask is compiled into a block-sparse structure so that fully masked blocks can be skipped. When

requested, the attention call returns both the output and an auxiliary log-sum-exp statistic. The sink path then applies the model-specific scaling function to the output:

$$M_{b,h,q,k} = \mathbb{1}[k \le q] \wedge \mathbb{1}[q - k \le w \ \vee \ k < p], \tag{11}$$

$$(z, \ell) = \text{FlexAttention}(Q, K, V; \text{BlockMask}(M)), \tag{12}$$

$$\alpha_{\text{sink}} = f_\eta(\ell), \tag{13}$$

$$z' = z \odot \alpha_{\text{sink}}, \tag{14}$$

where w is the local-window size, p is the number of always-visible prefix positions, and f_η is the learned sink-scaling function. For the zero-value-sink construction above, $f_\eta(\ell) = \sigma(\ell - s_\eta)$. The prefix budget p and the learned sink logit s_η are distinct: p controls token visibility in the mask, whereas s_η reallocates softmax mass through output scaling.

Gradient flow and autograd integration. The data-dependent sink scale couples the gradients of the attention output z and the log-sum-exp statistic ℓ. Let d denote the head dimension. By the chain rule,

$$\nabla_z \mathcal{L} = \nabla_{z'} \mathcal{L} \odot \alpha_{\text{sink}}, \tag{15}$$

$$\nabla_{\alpha_{\text{sink}}} \mathcal{L} = \sum_{j=1}^{d} \left(\nabla_{z'} \mathcal{L} \odot z\right)_j, \tag{16}$$

$$\nabla_\ell \mathcal{L} = \nabla_{\alpha_{\text{sink}}} \mathcal{L} \odot f'_\eta(\ell), \tag{17}$$

and the sink-parameter gradient is

$$\nabla_\eta \mathcal{L} = \nabla_{\alpha_{\text{sink}}} \mathcal{L} \odot \frac{\partial f_\eta(\ell)}{\partial \eta}. \tag{18}$$

A fixed fused-kernel interface may not expose the auxiliary gradient path through ℓ. Our implementation composes FlexAttention with the sink-scaling operation under AOTAutograd and `torch.compile`, allowing the compiler to generate forward and backward code for Q, K, V, and the sink parameters without materializing an $O(n^2)$ Jacobian.

Why a programmable kernel interface is needed. Optimized attention kernels are essential, but a fixed interface can omit model-specific masking, score modification, or auxiliary-gradient behavior. In on-policy RL, a small attention mismatch can be repeatedly amplified across long trajectories and policy updates. The FlexAttention path therefore prioritizes explicit control over the model's attention semantics while retaining a fused, block-sparse execution path.

Memory-oriented implementation optimizations. Profiling identified two avoidable sources of memory allocation: eager materialization in the sink-rescaling computation and replication of mask metadata across batch and head dimensions. We address them as follows.

Compilation and fusion. We apply `torch.compile` to the composed FlexAttention and sink-scaling path. PyTorch Inductor can fuse eligible pointwise operations with surrounding generated code, reducing the number and lifetime of materialized intermediate tensors.

Mask broadcasting. When the attention pattern is shared across batch elements and heads, block-mask construction omits explicit batch and head dimensions and relies on kernel-side broadcasting. This avoids storing repeated copies of the same sparse mask metadata. The aggregate effect of the optimized path is evaluated through peak-VRAM measurements in Section 6; we do not report a separate per-optimization ablation.

6 Experiments

We report two complementary measurements: preliminary policy-learning trends in the τ^2-Bench retail domain and peak-memory scaling for the attention implementation. The first

documents behavior over an observed training window; it is not an algorithmic comparison. The second measures peak VRAM only and does not establish throughput or wall-clock improvements.

6.1 Experimental Design

Experiment 1: Preliminary retail training. We evaluate the integrated GRPO pipeline on the retail domain of τ^2-Bench (Barres et al., 2025). The agent interacts with a dynamic user simulator, executes multi-step API calls, and follows domain-specific standard operating procedures. The purpose of this experiment is to document whether validation reward and associated training diagnostics improve over the observed run, not to isolate the causal effect of GRPO relative to another optimizer.

We track three metrics:

- **Validation Reward (mean@1):** the benchmark's programmatically verified task-success score for a single sampled validation trajectory per task.
- **Training Score Proxy:** a dashboard diagnostic monitored during training. The dashboard historically labels this trace as a "critic-style score," but the baseline does not train a separate critic or value network.
- **Trajectory Reward Proxy:** a rolling summary of raw episode rewards used to monitor the direction and variability of the observed training run.

Experiment 2: Peak-memory scaling. We compare the model-native eager attention reference path with the optimized sink-aware FlexAttention path under the same fixed model, batch, and training configuration. The optimized path is configured to reproduce the same causal/sliding-window policy and sink scaling as the reference path. We measure peak high-bandwidth-memory usage (**Peak VRAM in GB**) at sequence lengths of 1024, 2048, 4096, and 8192 tokens. Because the current evaluation reports neither throughput nor a forward/backward numerical-equivalence test, the conclusions are limited to observed peak memory and execution feasibility in this configuration.

6.2 Experimental Results

6.2.1 Preliminary Retail Training Results

Table 1 summarizes values estimated from the available dashboard screenshots. The retail validation reward rises from 0.25 early in training to 0.44 later in the observed training window. These values represent two portions of the same run rather than an untrained baseline and a final converged model.

Table 1: **Preliminary retail training trends.** Values are visually estimated from dashboard screenshots and should be interpreted as approximate. Higher values are better.

Metric	Early training	Later training
Retail validation reward (mean@1)	0.25	0.44
Training score proxy (mean)	0.18	0.40
Trajectory reward proxy (mean)	0.18	0.39

Individual evaluations are variable, as expected for multi-turn, state-dependent tasks. Over the displayed window, the smoothed validation trace trends upward, while the training-score and trajectory-reward proxies rise from 0.18 to 0.40 and 0.39, respectively. These observations are consistent with the integrated pipeline receiving a useful learning signal under the implemented GRPO update. Without multiple seeds, an optimizer baseline, or exported scalar logs, they do not isolate the effect of GRPO, establish variance reduction, or support a statistical significance claim.

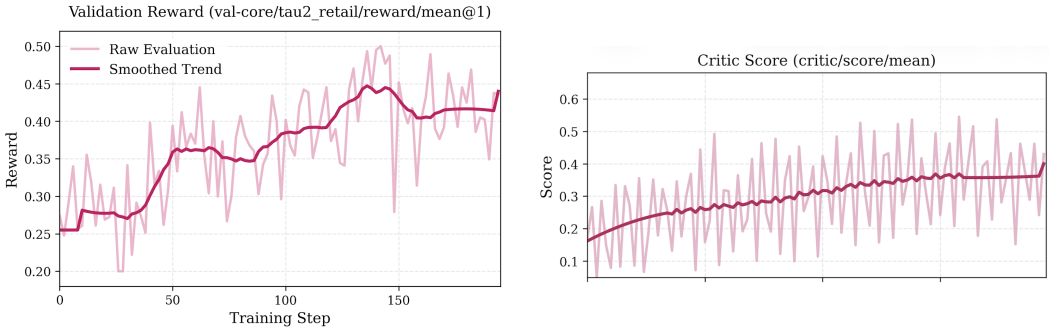

Figure 4: **Approximate learning curves.** The left panel shows retail validation reward; the right panel shows the training-score diagnostic whose dashboard label is "critic-style score." Dots denote noisy per-evaluation readings estimated from screenshots, and thick lines show the displayed smoothed trends.

6.2.2 Peak-Memory Results

Table 2 reports peak VRAM for the two attention paths. The optimized path uses less memory at every sequence length for which both paths complete, while the eager path runs out of memory at 8192 tokens.

Table 2: **Peak-memory comparison.** Peak VRAM during the measured training configuration. The optimized sink-aware FlexAttention path completes the 8192-token configuration for which the eager reference path runs out of memory.

Sequence Length	Eager Baseline (GB)	Flex + Sink (GB)	Savings
1024	21.07	20.11	0.96 GB (4.5%)
2048	23.27	21.02	2.25 GB (9.7%)
4096	28.06	22.52	5.54 GB (19.7%)
8192	OOM	25.53	–

At 4096 tokens, the optimized path reduces peak VRAM from 28.06 GB to 22.52 GB, a reduction of 5.54 GB or 19.7%. At 8192 tokens, it completes the measured configuration with a peak allocation of 25.53 GB, whereas the eager reference path encounters an out-of-memory error. Thus, in this fixed configuration, the optimized path removes the 8192-token memory failure observed for the eager baseline. Because the experiment measures only peak VRAM, it does not establish improvements in throughput, latency, total training time, or accelerator utilization.

7 Discussion and Limitations

First, our system is particularly well suited to agentic tasks that operate in controlled environments with programmatically verifiable outcomes. This setting is naturally instantiated by τ^2-Bench, where the policy interacts with a user simulator, invokes domain tools, and receives feedback based on the resulting environment state. By grounding optimization in auditable actions and objective task outcomes, the framework enables scalable and reproducible policy learning. The same approach has strong potential to extend to other domains that provide structured tool interfaces, executable environments, or clearly defined success criteria. For more open-ended applications, programmatic verification can be complemented with human feedback or learned reward models, providing a natural path toward broader task coverage.

Second, the preliminary retail-domain experiment provides encouraging evidence for the effectiveness of the proposed policy-learning pipeline. Over the observed training window,

the displayed reward-associated traces exhibit a clear upward trend from the early to later stages of training, indicating that the policy is able to benefit from the optimization signal. These results serve as a promising proof of concept for applying GRPO to tool-using agentic systems. They also motivate broader evaluations across domains, random seeds, model scales, and training horizons to further characterize the robustness and generality of the observed improvements.

Third, programmatic rewards provide an efficient and scalable source of supervision by directly connecting policy behavior to verifiable task outcomes. This foundation can support increasingly rich reward functions that jointly assess final-state correctness, policy compliance, tool-use quality, and dialogue naturalness. GRPO is especially well matched to settings in which rollout groups contain diverse behavioral outcomes, allowing relative advantages to identify and reinforce stronger trajectories. Techniques such as diversity-aware sampling, curriculum design, adaptive grouping, and reward shaping offer promising directions for maintaining informative reward variation and improving optimization efficiency.

Forth, the memory evaluation demonstrates the feasibility of executing the proposed training approach at sequence lengths of up to 8192 tokens under the tested configuration. In particular, the sink-aware execution path provides a practical foundation for scaling agentic policy learning to longer interaction trajectories while retaining the intended eager reference semantics. The current peak-VRAM measurements establish an important systems proof of concept, and the same evaluation framework can be extended to characterize throughput, latency, training time, and accelerator utilization. Forward- and backward-equivalence tests, together with targeted validation of sink handling, masking, position indexing, and log-sum-exp statistics, can provide additional confirmation of semantic consistency. Overall, these results establish a strong foundation for developing efficient, reliable, and scalable training systems for long-horizon tool-using agents.

8 Conclusion

We presented SINKFLEX-RL, a modular RL training system that integrates a Gymnasium-compatible dual-control environment interface, VERL-style rollout dataflow, GRPO updates without a separate value model, and a sink-aware FlexAttention path. In the current evidence, retail validation and training diagnostics trend upward over one preliminary run, while the optimized attention path reduces peak memory and completes the measured 8192-token configuration where the eager reference path runs out of memory. These results support the narrower claim that environment, RL-dataflow, and attention-kernel integration can improve the memory feasibility of long-horizon agent training. They do not yet establish algorithmic superiority, broad generalization, exact implementation equivalence, or end-to-end computational speedup.

Ethics Statement

The system trains agents that can call tools and update environment state. Such agents should be evaluated for policy compliance, user deception, unsafe tool use, privacy leakage, and simulator overfitting before deployment. The present work focuses on system feasibility and preliminary training behavior and does not claim deployment readiness.

References

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pp. 35549–35562, 2024.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Qihua Dong, Gozde Sahin, Pei Wang, Zhaowei Cai, Robik Shrestha, Hao Yang, and Davide Modolo. Visual reasoning through tool-supervised reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8993–9002, 2026.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

PyTorch Contributors. Pytorch flexattention documentation. `https://docs.pytorch.org/docs/stable/nn.attention.flex_attention.html`, 2026. Accessed 2026-06-21.

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

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 414–431, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.