AReaL 2.0: From Efficient RL Post-Training to Comprehensive Agentic Data Platform

AReaL Community Co-Founder

Assistant Professor @ CSE, HKUST

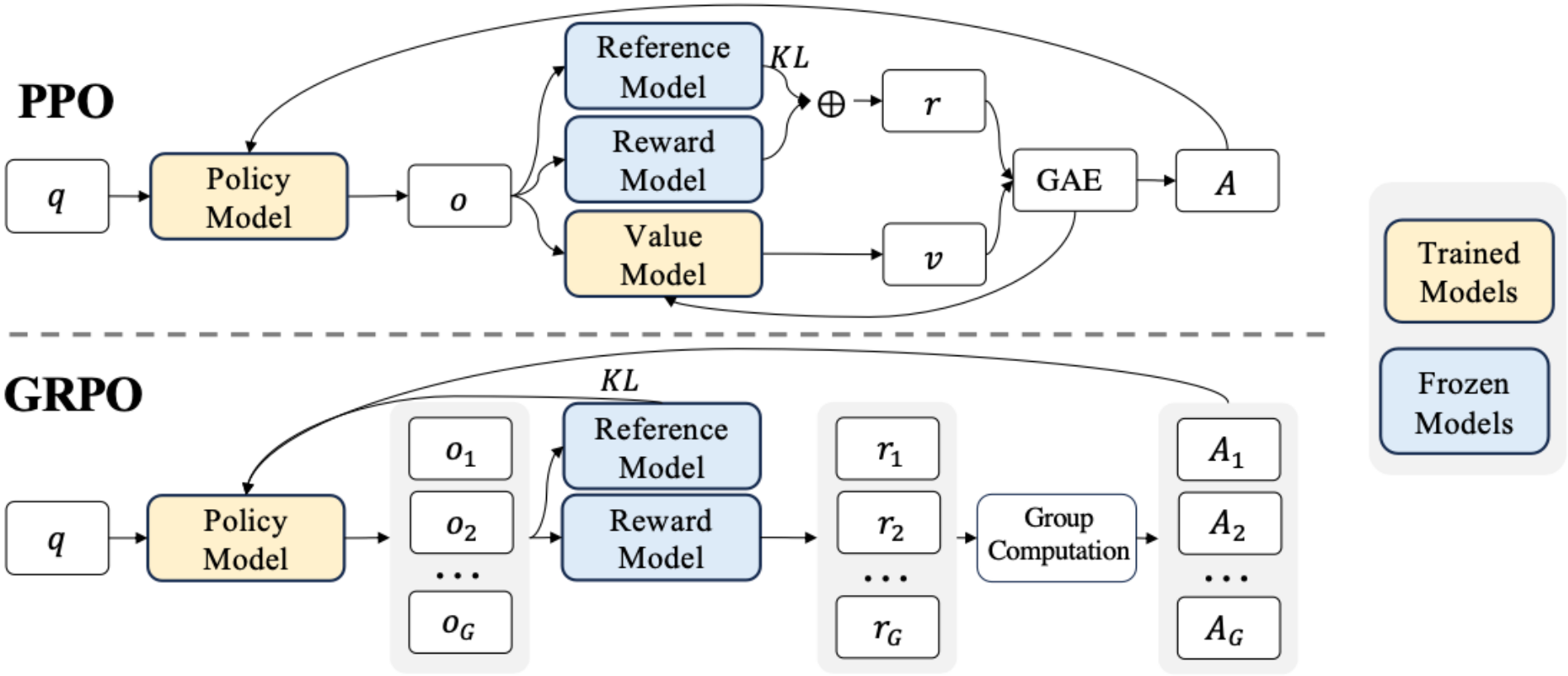
Binhang Yuan

20.04.26

AReaL 1.0:

Asynchronous RL Is Efficient for LLM Post-Training.

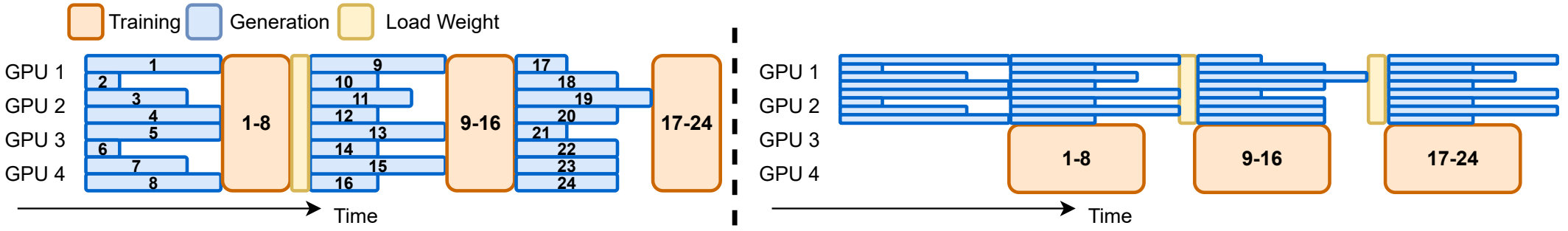
RL for LLM Post Training



3

RL Training: Sync vs. Async

- **Rollout generation**: The policy LLM produces responses from a batch of prompts using generative inference.
- **Reward computation**: Using prompts and generated responses, the critic computes their values, the reference policy computes their reference log probabilities, and the reward model computes their rewards, potentially with functions executed in a sandbox.
- **Training**: The policy LLM are updated via gradient based optimization, using the batch of data produced by previous stages and the reward function.

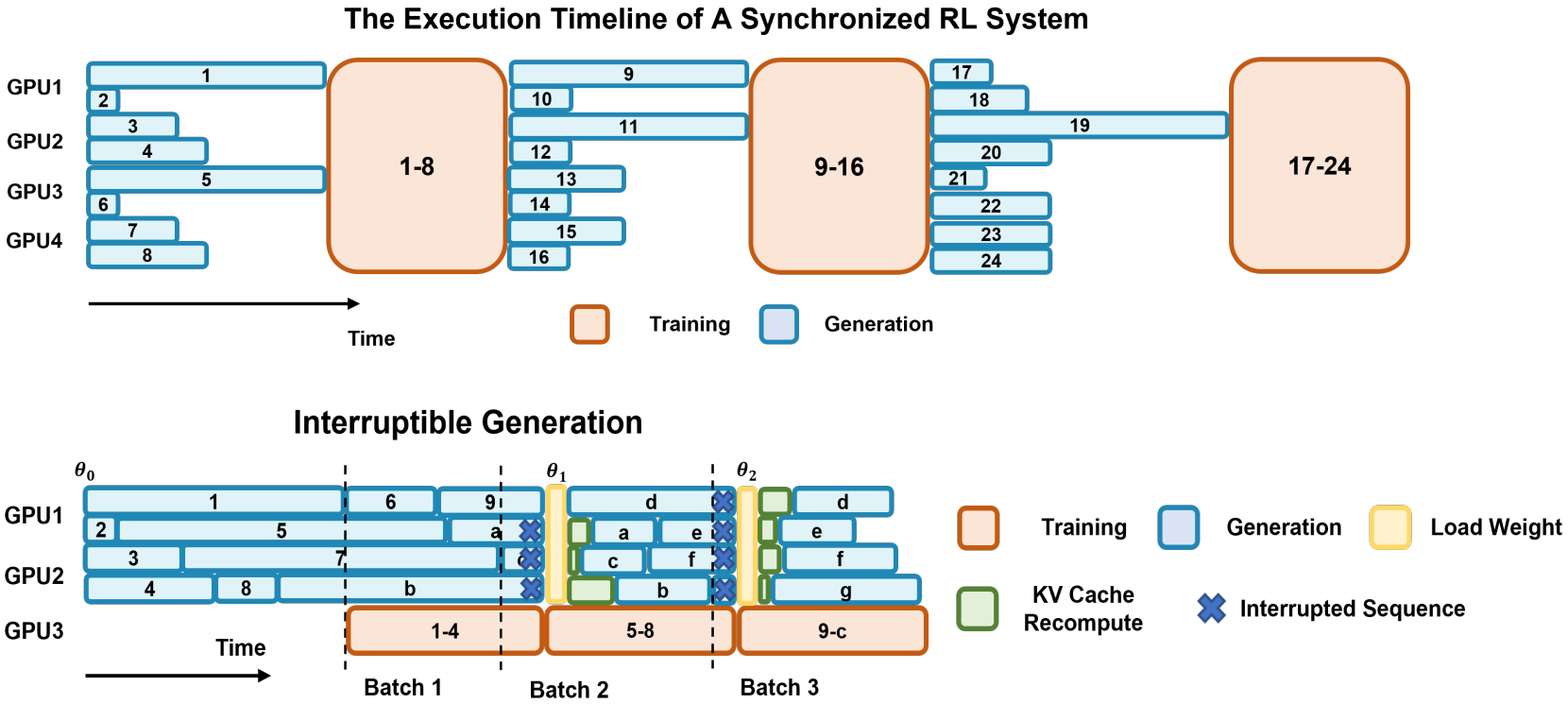


Sync RL Training

1-Step Async RL Training

AReaL:

Asynchronous Reinforcement Learning for Efficient and Scalable Language Reasoning



AReaL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning

Wei Fu[12], Jiaxuan Gao[1], Xujie Shen[2], Chen Zhu[2], Zhiyu Mei[12], Chuyi He[2], Shusheng Xu[12], Guo Wei[2], Jun Mei[2], Jiashu Wang[3], Tongkai Yang[2], Binhang Yuan[3], Yi Wu[1]

[1] IIIS, Tsinghua University, [2] Ant Research, [3] HKUST
fuwth17@gmail.com, jxwuyi@gmail.com

Abstract

Reinforcement learning (RL) has become a trending paradigm for training large language models (LLMs), particularly for reasoning tasks. Effective RL for LLMs requires massive parallelization and poses an urgent need for efficient training systems. Most existing large-scale RL systems for LLMs are synchronous by alternating generation and training in a batch setting, where the rollouts in each training batch are generated by the same (or latest) model. This stabilizes RL training but suffers from severe system-level inefficiency. Generation must wait until the longest output in the batch is completed before model update, resulting in GPU underutilization. We present AReaL, a *fully asynchronous* RL system that completely decouples generation from training. Rollout workers in AReaL continuously generate new outputs without waiting, while training workers update the model whenever a batch of data is collected. AReaL also incorporates a collection of system-level optimizations, leading to substantially higher GPU utilization. To stabilize RL training, AReaL balances the workload of rollout and training workers to control data staleness, and adopts a staleness-enhanced PPO variant to better handle outdated training samples. Extensive experiments on math and code reasoning benchmarks show that AReaL achieves **up to 2.77× training speedup** compared to synchronous systems with the same number of GPUs and matched or even improved final performance. The code of AReaL is available at `https://github.com/inclusionAI/AReaL/`.

1 Introduction

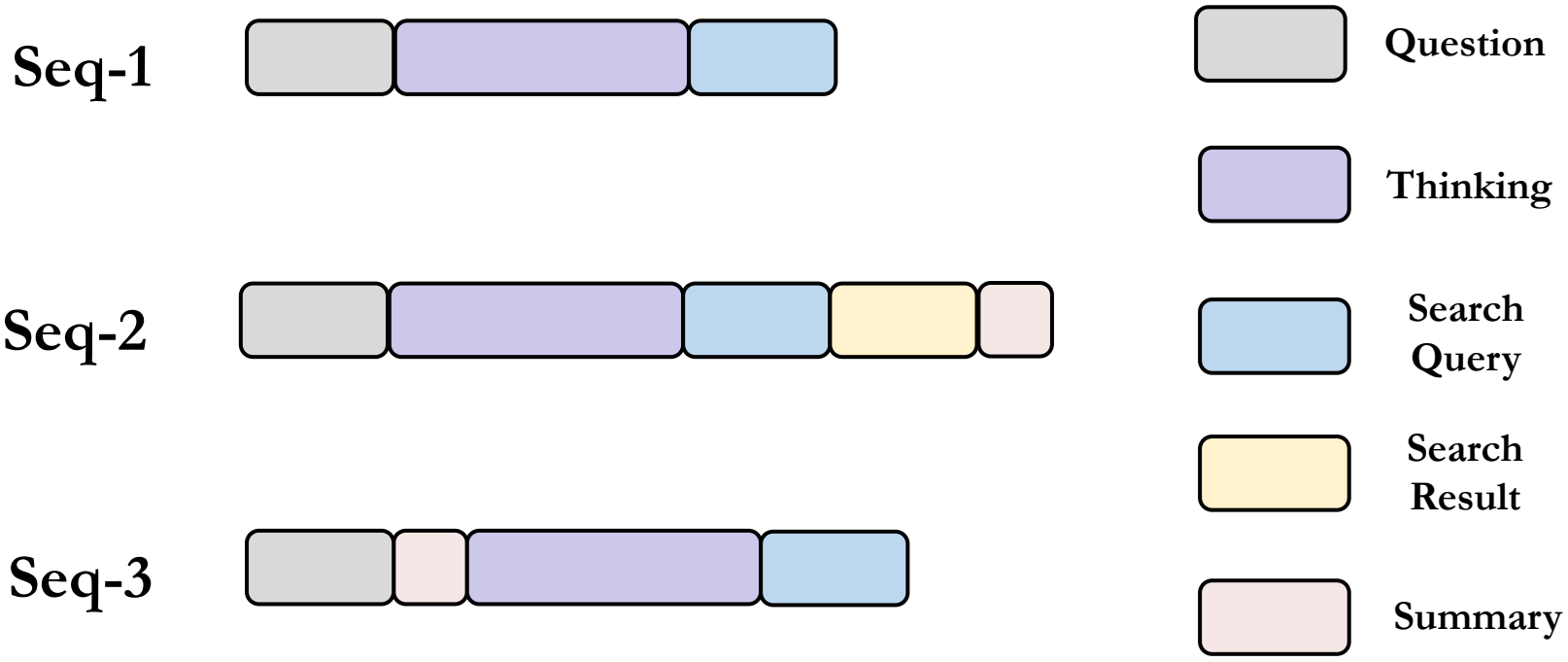
Reinforcement learning (RL) has emerged as a new scaling paradigm for enhancing the capabilities of large language models (LLMs) by enabling thinking abilities [52]. Given a prompt, RL allows an LLM to generate thinking tokens before outputting a final answer, enabling *test-time scaling* [29, 47]. These thinking LLMs are named Large Reasoning Models (LRMs) and have been shown to have particularly strong capabilities on challenging reasoning problems, such as math [9, 5, 20], coding [3, 14, 15], logic puzzles [22, 34], and agentic tasks [23, 57].

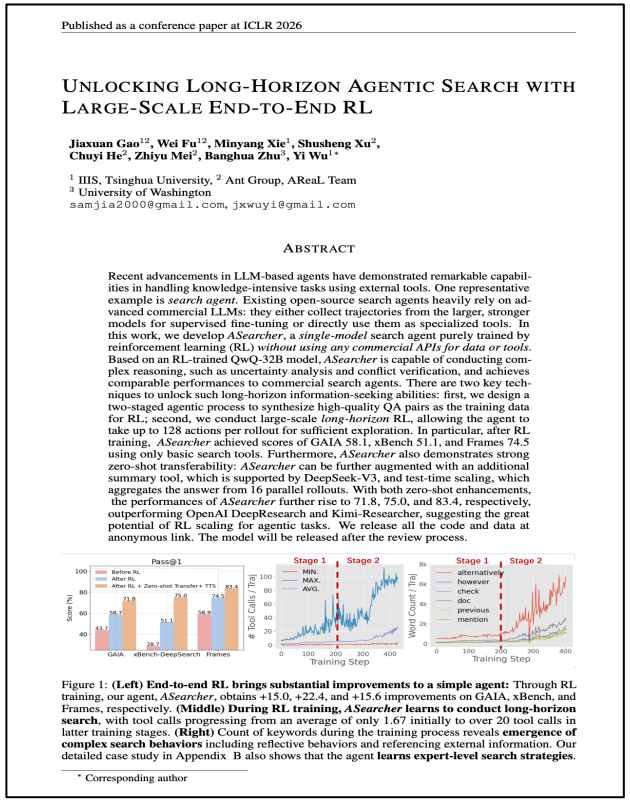
Effective RL training often requires massive parallelization to derive a large batch of rollouts for sufficient exploration, which is the key to obtaining optimal model performance. For example, popular RL algorithms, such as PPO [42] and GRPO [43], often require an effective training batch of thousands of outputs [60, 61, 53]. Moreover, an LRM can generate tens of thousands of thinking tokens for each input prompt [6], further posing an urgent need for an efficient training system to run RL training on a large scale.

[NeurIPS 2025]

ASearcher:

Beyond Ten Turns: Unlocking Long-Horizon Agentic Search with Large-Scale Asynchronous RL





Published as a conference paper at ICLR 2026

UNLOCKING LONG-HORIZON AGENTIC SEARCH WITH LARGE-SCALE END-TO-END RL

Jiaxuan Gao[1,2], Wei Fu[1,2], Minyang Xie[1], Shusheng Xu[2], Chuyi He[2], Zhiyu Mei[2], Banghua Zhu[3], Yi Wu[1,*]

[1] IIIS, Tsinghua University, [2] Ant Group, AReaL Team
[3] University of Washington
samjia2000@gmail.com, jxwuyi@gmail.com

ABSTRACT

Recent advancements in LLM-based agents have demonstrated remarkable capabilities in handling knowledge-intensive tasks using external tools. One representative example is *search agent*. Existing open-source search agents heavily rely on advanced commercial LLMs: they either collect trajectories from the larger, stronger models for supervised fine-tuning or directly use them as specialized tools. In this work, we develop *ASearcher*, a *single-model* search agent purely trained by reinforcement learning (RL) *without using any commercial APIs for data or tools*. Based on an RL-trained QwQ-32B model, *ASearcher* is capable of conducting complex reasoning, such as uncertainty analysis and conflict verification, and achieves comparable performances to commercial search agents. There are two key techniques to unlock such long-horizon information-seeking abilities: first, we design a two-staged agentic process to synthesize high-quality QA pairs as the training data for RL; second, we conduct large-scale *long-horizon* RL, allowing the agent to take up to 128 actions per rollout for sufficient exploration. In particular, after RL training, *ASearcher* achieved scores of GAIA 58.1, xBench 51.1, and Frames 74.5 using only basic search tools. Furthermore, *ASearcher* also demonstrates strong zero-shot transferability: *ASearcher* can be further augmented with an additional summary tool, which is supported by DeepSeek-V3, and test-time scaling, which aggregates the answer from 16 parallel rollouts. With both zero-shot enhancements, the performances of *ASearcher* further rise to 71.8, 75.0, and 83.4, respectively, outperforming OpenAI DeepResearch and Kimi-Researcher, suggesting the great potential of RL scaling for agentic tasks. We release all the code and data at anonymous link. The model will be released after the review process.

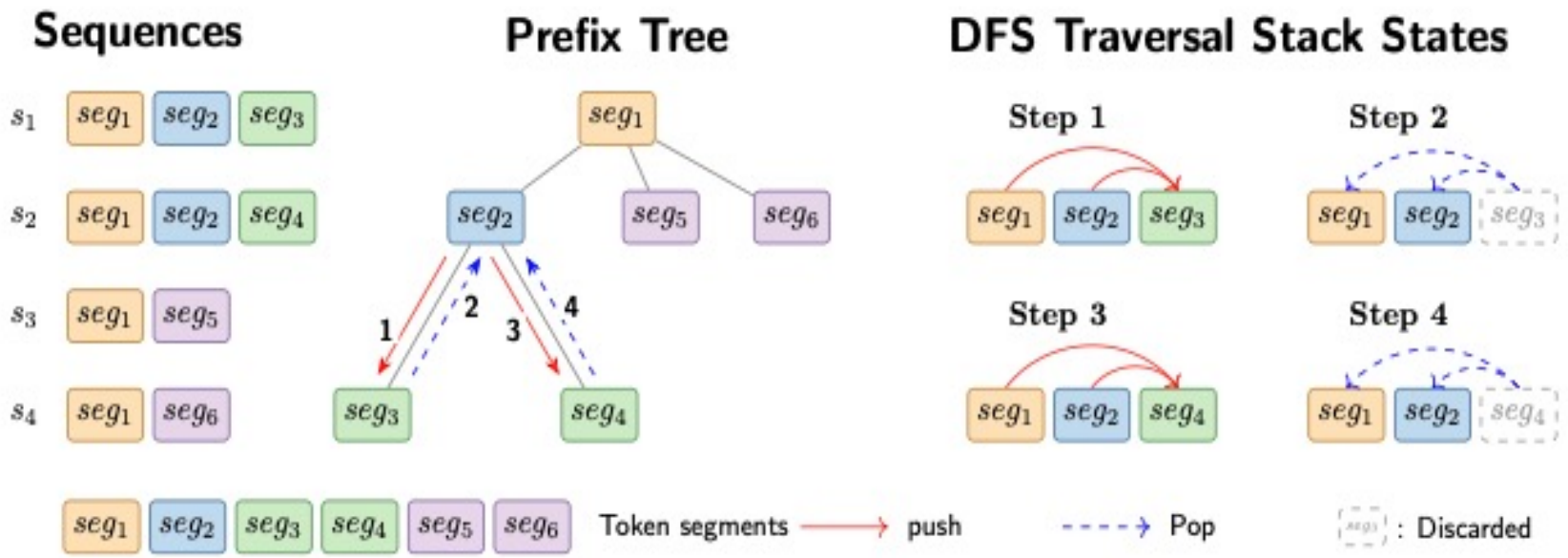
Figure 1: **(Left) End-to-end RL brings substantial improvements to a simple agent:** Through RL training, our agent, *ASearcher*, obtains +15.0, +22.4, and +15.6 improvements on GAIA, xBench, and Frames, respectively. **(Middle) During RL training, *ASearcher* learns to conduct long-horizon search**, with tool calls progressing from an average of only 1.67 initially to over 20 tool calls in latter training stages. **(Right)** Count of keywords during the training process reveals **emergence of complex search behaviors** including reflective behaviors and referencing external information. Our detailed case study in Appendix B also shows that the agent **learns expert-level search strategies**.

* Corresponding author

[ICLR 2026]

AReaL-DTA:

Dynamic Tree Attention for Efficient Reinforcement Learning of Large Language Models



AReaL-DTA: Dynamic Tree Attention for Efficient Reinforcement Learning of Large Language Models

Jiarui Zhang[1,2*], Yuchen Yang[2*], Ran Yan[1,3*], Zhiyu Mei[3], Liyuan Zhang[2], Daifeng Li[1], Wei Fu[2,3], Jiaxuan Gao[2,3], Shusheng Xu[3], Yi Wu[2], Binhang Yuan[1]

[1]HKUST, [2]Tsinghua University, [3]AReaL Team, Ant Group

*Equal contribution

Abstract

Reinforcement learning (RL) based post-training for large language models (LLMs) is computationally expensive, as it generates many rollout sequences that could frequently share long token prefixes. Existing RL frameworks usually process these sequences independently, repeatedly recomputing identical prefixes during forward and backward passes during policy model training, leading to substantial inefficiencies in computation and memory usage. Although prefix sharing naturally induces a tree structure over rollouts, prior tree-attention–based solutions rely on fully materialized attention masks and scale poorly in RL settings. In this paper, we introduce AReaL-DTA to efficiently exploit prefix sharing in RL training. AReaL-DTA employs a depth-first-search (DFS)–based execution strategy that dynamically traverses the rollout prefix tree during both forward and backward computation, materializing only a single root-to-leaf path at a time. To further improve scalability, AReaL-DTA incorporates a load-balanced distributed batching mechanism that dynamically constructs and processes prefix trees across multiple GPUs. Across the popular RL post-training workload, AReaL-DTA achieves up to 8.31× in τ^2-bench higher training throughput.

1 Introduction

Post-training large language models (LLMs) with reinforcement learning (RL) is often computationally intensive [1, 2]. RL post-training workflows often require generating many similar rollout sequences for each prompt or environment state to explore different outcomes or gather sufficient reward signals [3, 4]. Crucially, these sequences frequently share long prefixes — for example, multiple candidate responses may begin with the same user prompt or initial dialogue turns [5]. In current practice, each sequence is processed independently, leading to redundant computations of the same prefix across rollouts during the training of the policy model. This repetition incurs high computational cost and memory overhead, as the training of the policy model needlessly re-computes identical token prefixes in each forward and backward pass, which results in a major bottleneck that slows training throughput and inflates GPU memory usage. In this paper, we want to explore *how to effectively leverage this prefix sharing paradigm to improve RL training efficiency*.

Prefix sharing is ubiquitous in modern RL training workflows for LLMs. For instance, RL for advanced reasoning LLMs and multi-turn agent training often sample multiple continuations per context, all starting with the same prompt or chain-of-thought reasoning steps [3, 4, 6, 7]. These branching trajectories naturally form a prefix tree structure: they share a common stem (prefix) before diverging into different outcomes. Yet, vanilla RL training pipelines fail to exploit this structure — they treat each branch as a separate sequence. As a result, the overlapping prefix computations are performed multiple times, wasting compute budget and memory

[Arxiv. 2602.00482]

AReaL-Hex:

Accommodating Asynchronous RL Training over Heterogeneous GPUs

- Computation by itself is heterogeneous:
  - LLM inference is still I/O bounded;
  - LLM Training is computation bounded.
- The communication volume is significantly reduced compared with standard pretraining:
  - Inference workers only needs to fetch weights after the training workers conducted SGD updates.

AReaL-Hex: Accommodating Asynchronous RL Training over Heterogeneous GPUs

Ran Yan[1*], Youhe Jiang[1*], Tianyuan Wu[1], Jiaxuan Gao[2], Zhiyu Mei[3], Wei Fu[2], Haohui Mai[1], Wei Wang[1], Yi Wu[2], Binhang Yuan[1]

[1]HKUST, [2]Tsinghua University, [3]Ant Group

*Equal contribution

Abstract

Maximizing the training throughput and cost-efficiency of reinforcement learning (RL) for large language models (LLMs) is essential to democratize this advanced technique. One promising but challenging approach is to deploy such a computational workflow over heterogeneous GPU clusters. Unlike conventional large-scale LLM pretraining, RL training generally decomposes into three coupled stages, i.e., rollout generation, reward computation, and policy/value updates, which exhibit markedly different compute intensities, memory footprints, and communication patterns. Recent research shows that fully asynchronous RL training can disaggregate these stages across disjoint hardware pools without sacrificing training stability, creating a great opportunity for real-world heterogeneous deployment. Towards this end, we present AReaL-Hex, a heterogeneity-aware asynchronous RL training system that effectively schedules how to execute rollout generation and policy model training over heterogeneous GPU clusters while enforcing data staleness bounds. Concretely, AReaL-Hex uses a two-phase scheduler: (i) a constrained search with mixed-integer linear programming to select per-stage parallelization strategies and workload assignments given a resource budget, and (ii) a graph-partitioning step that allocates heterogeneous GPUs and interconnects to maximize end-to-end throughput. Built atop a fully asynchronous RL architecture, AReaL-Hex maps HBM-I/O-bound generation and compute-bound optimization to more cost-efficient resources and balances their producer–consumer interactions to avoid both idleness and stale rollout trajectories. On the mathematical reasoning task with various model scales (1.5B, 7B, and 14B), compared to homogeneous deployments of state-of-the-art asynchronous RL systems: (i) When maintaining the same total budgets, AReaL-Hex delivers up to 1.50× higher training throughput; (ii) When achieving the same training throughput, AReaL-Hex results in up to 1.46× reduction in training cost.

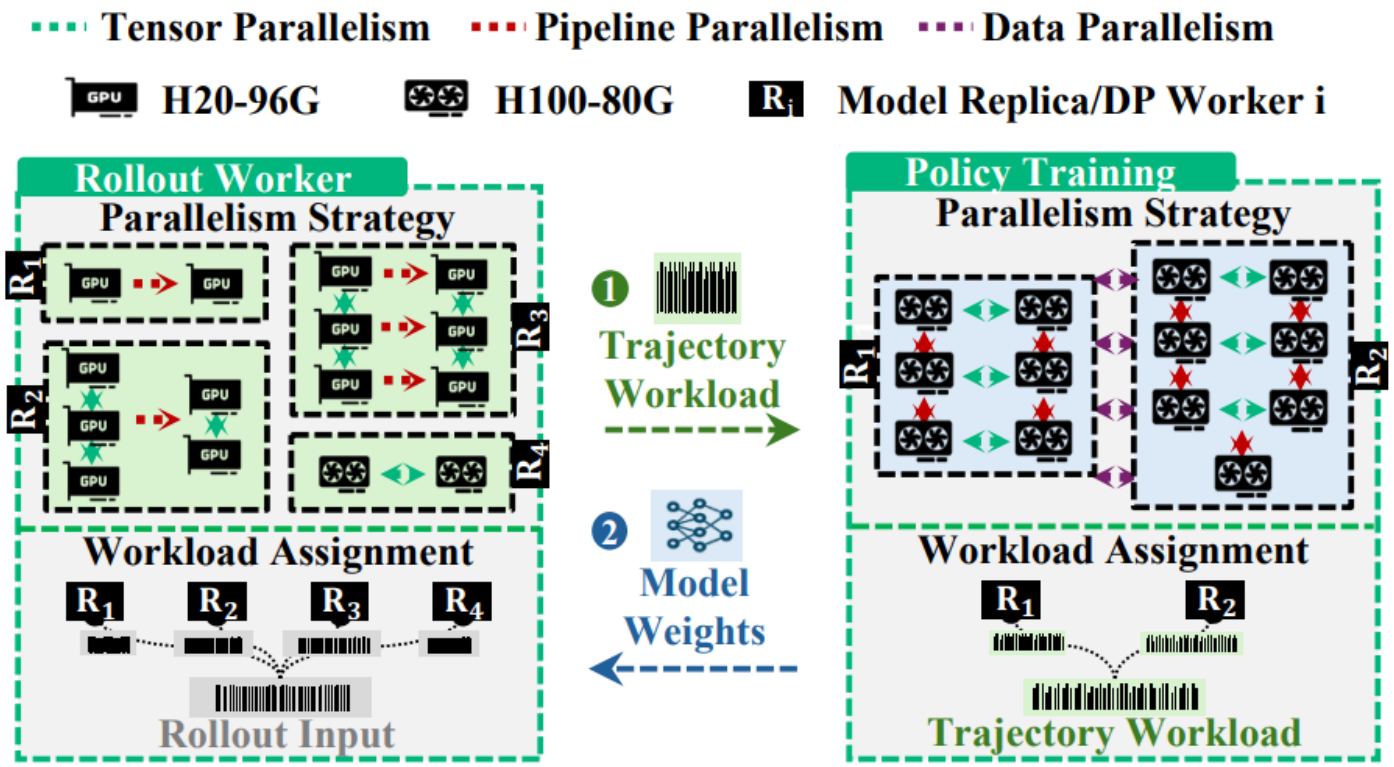
1 Introduction

Reinforcement learning (RL) has emerged as a popular approach for enhancing large language models (LLMs) across various tasks [1], including math [2], code generation [3], agentic tool use [4], etc, which becomes essential to advance the development of AI. Unlike conventional large-scale pretraining by standard stochastic gradient-based optimization, RL training comprises multiple coupled stages, where each stage includes heterogeneous compute intensities, memory footprints, and communication patterns [5]; furthermore, recent study suggests that asynchronous RL paradigm can effectively disaggregate these different stages over different set of hardwares without compromising the stability in RL training [6, 7]. Such an intrinsic computational heterogeneity under the disaggregated infrastructure naturally provides a great opportunity to deploy such a

[Arxiv. 2511.00796]

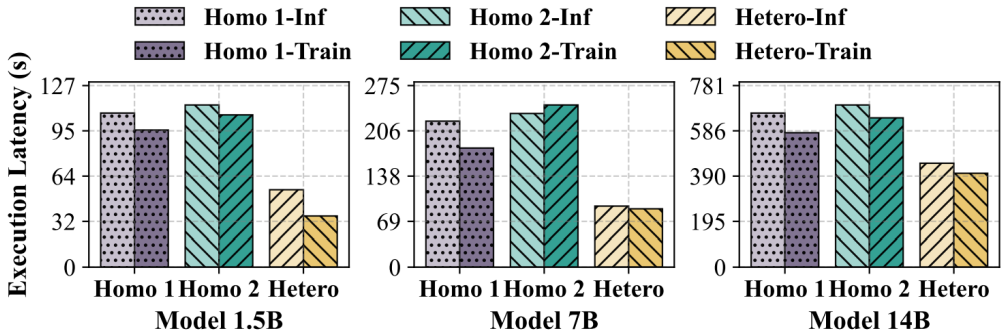
AReaL-Hex

Accommodating Asynchronous RL Training over Heterogeneous GPUs



Model	System	Throughput	Cost
1.5B model	AReaL (H800)	1.84×10^4 t/s	$126.72/h
	AReaL-Hex	1.79×10^4 t/s	$86.64/h
7B model	AReaL (H800)	5.98×10^3 t/s	$126.72/h
	AReaL-Hex	6.00×10^3 t/s	$86.64/h
14B model	AReaL (H800)	2.16×10^3 t/s	$126.72/h
	AReaL-Hex	2.30×10^3 t/s	$86.64/h

Cost with the same end-to-end throughput

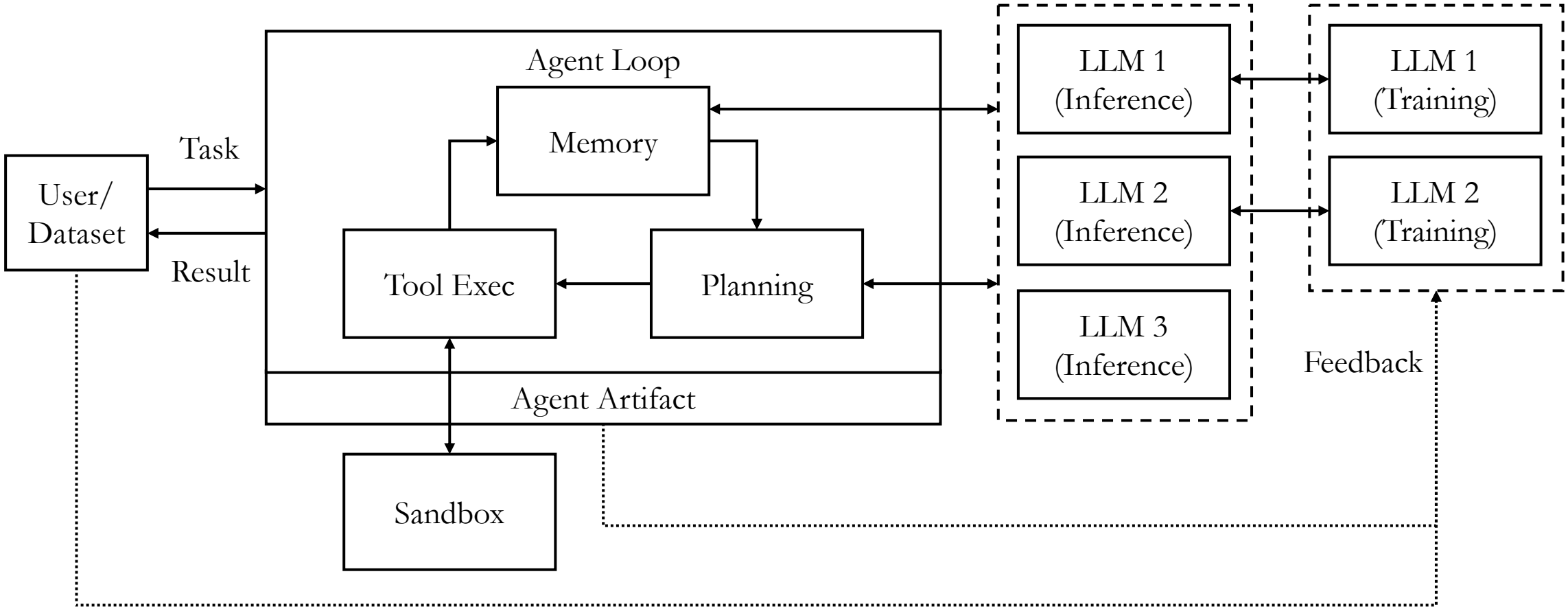


Throughput breakdown under the same cost

AReaL 2.0:

Agent-centric Platform to Support Flexible Online RL.

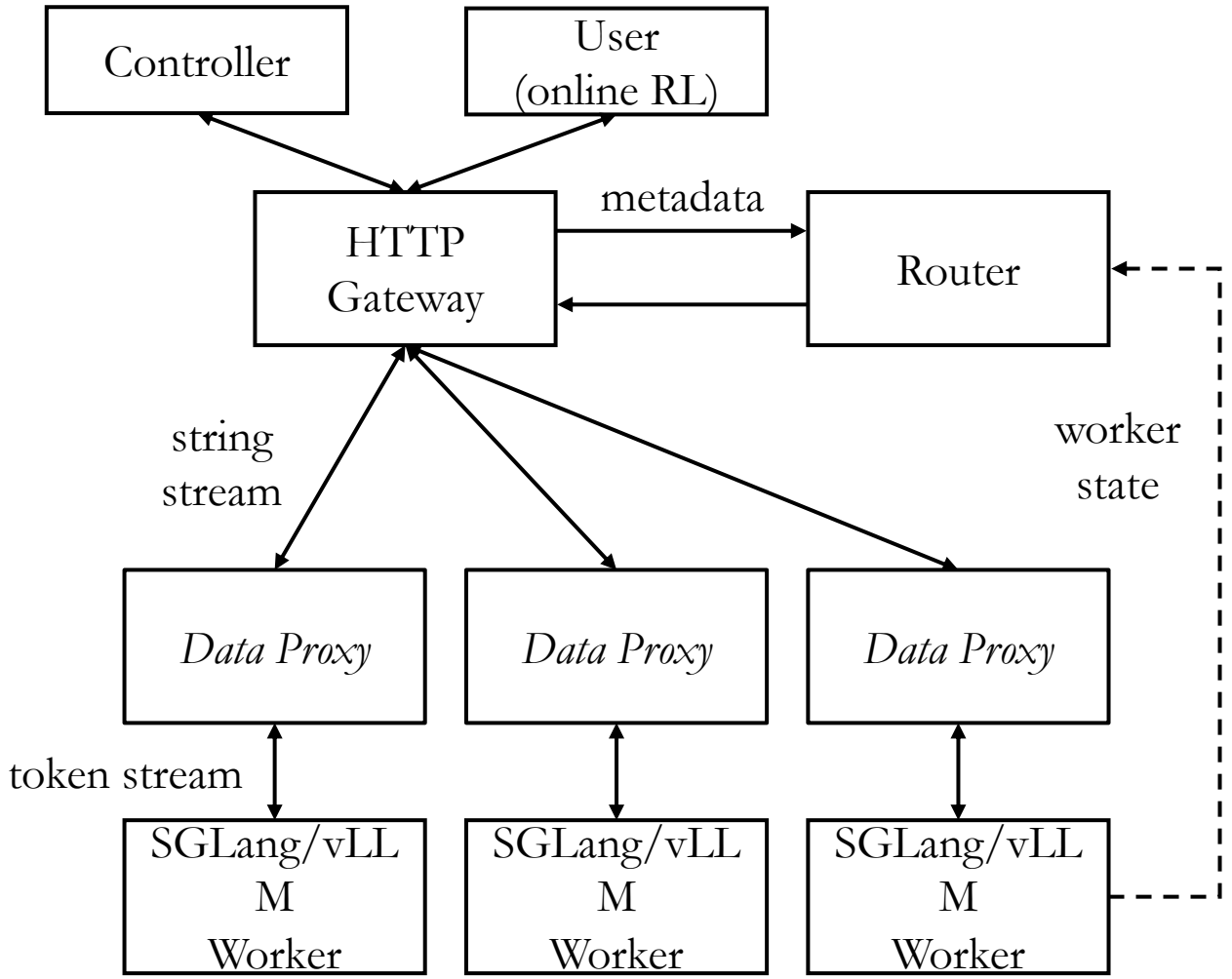
Agent-Centric Online RL Infra



AReaL 2.0 Inference:

Agentic Inference with Comprehensive Data Proxy

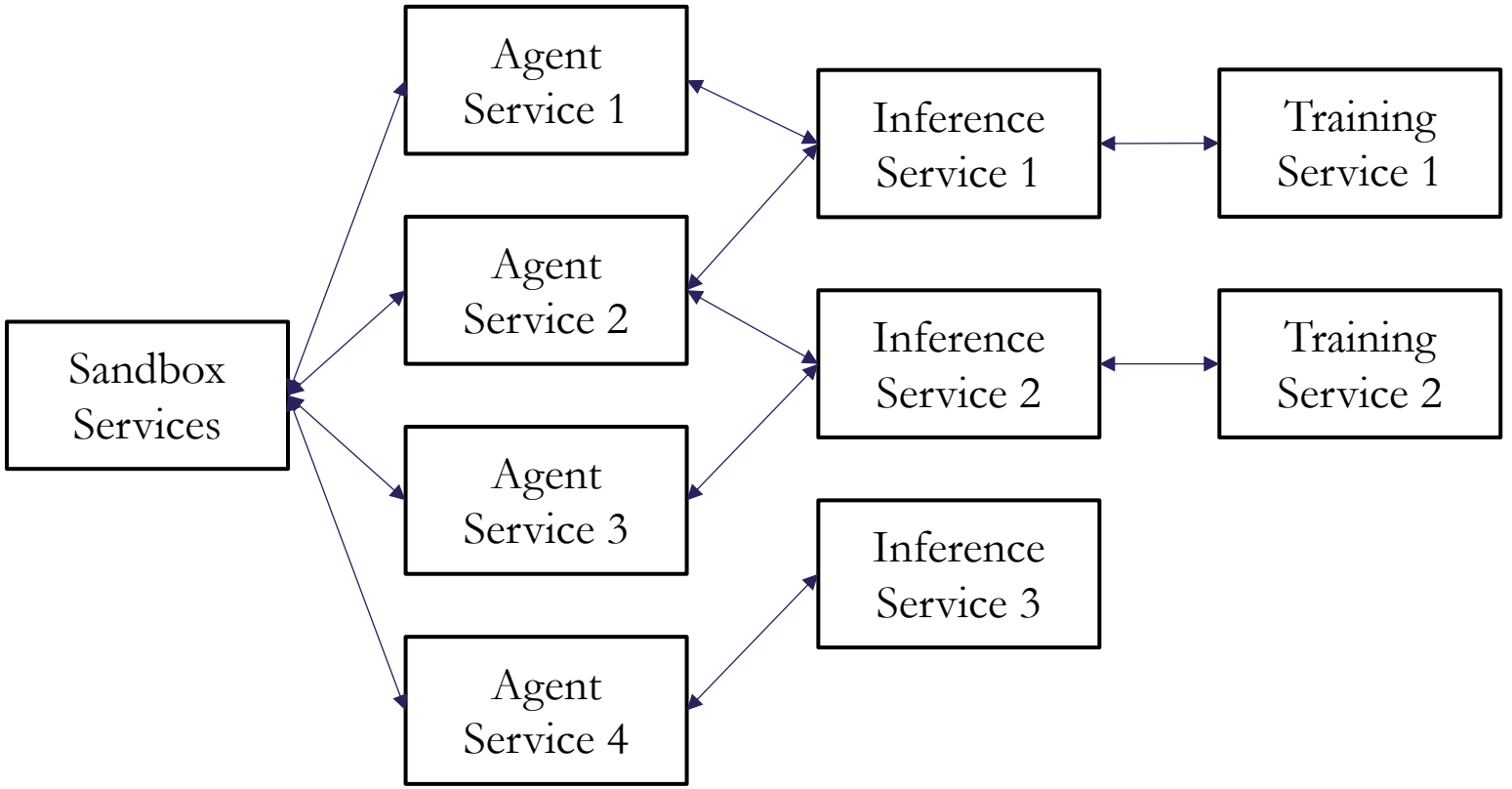
- Routing:
  - Route requests to different workers.
- Data proxy
  - Agent session lifecycle.
  - Tokenization consistency.
  - Rollout interruption.
  - Backend compatibility (vllm/sglang).
  - Data storing and transmission.



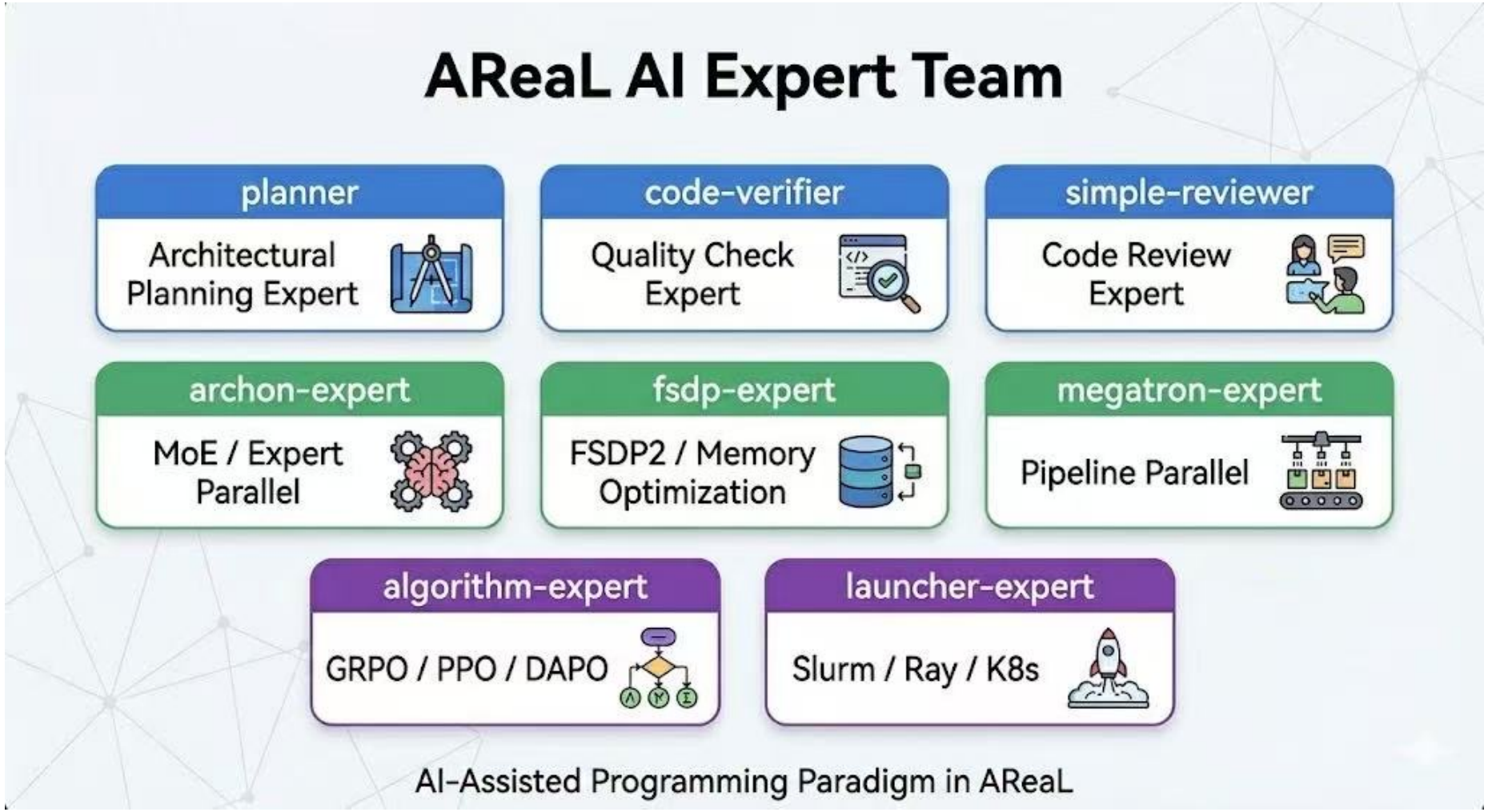
AReaL 2.0 Training:

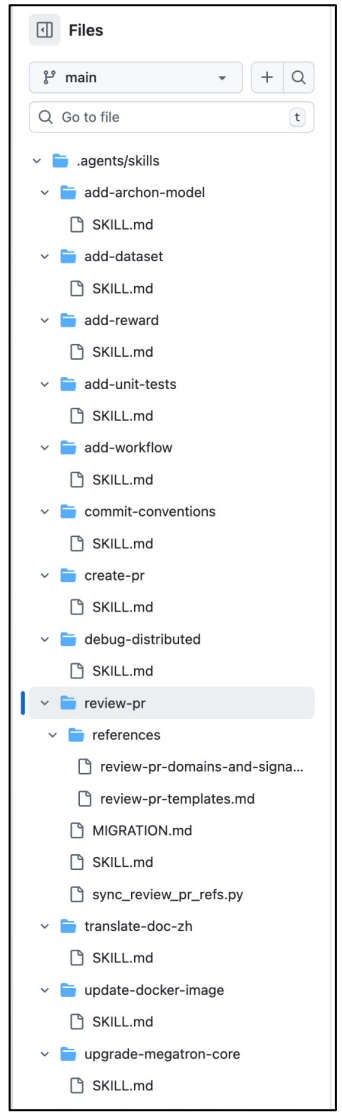
RL Training as a Light-weight Flexible Dynamic Micro-Service

- RL training:
  - A lightweight online micro-service.
- Online RL paradigms:
  - Training worker as plugin on demand.
  - Multi-agent online RL training.
  - In-context RL for agent service.
  - On policy distillation.



AI Assisted Development





Summary:

- In AReaL 1.0, we show that asynchronous RL is efficient:
  - ✓ **ASearcher:** asynchronous multi-turn RL beyond ten turns.
  - ✓ **AReaL-DTA:** effective system optimization with tree attention.
  - ✓ **AReaL-Hex:** effective scheduling of asynchronous RL under heterogeneity.
- Our roadmap for AReaL 2.0, an Agent-Centric Platform to Support Flexible RL Paradigms:
  - ❑ Agentic inference with comprehensive data proxy.
  - ❑ Agentic RL training as a light-weight flexible dynamic micro-service.
  - ❑ AI assisted development for the open-source community.
- Lots of ongoing research based on AReaL open-source community could be more impactful if they could be presented at PyTorch Day.

Thank you!