

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



# Reinforcement Learning Systems

**Carnegie Mellon University**  
Language Technologies Institute

Apurva Gandhi  
11-768: AI Agents

## 强化学习系统

Apurva Gandhi · 卡内基梅隆大学 · 语言技术研究所 · 11-768: AI 智能体

强化学习系统

## 第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

强化学习系统：概览

# RL Systems: Overview

2

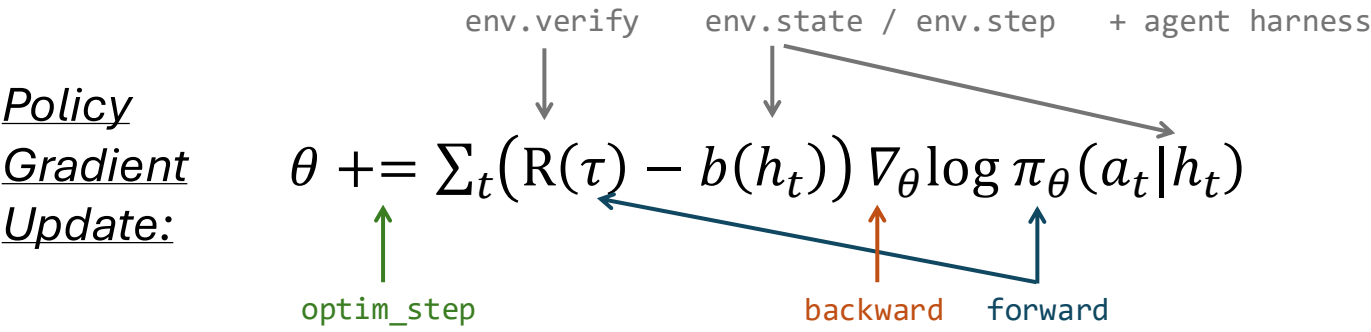
强化学习系统：概览

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

What operations must an LLM RL Framework support?



LLM 强化学习框架必须支持哪些操作？

策略梯度更新：

$$\theta += \sum_t (R(\tau) - b(h_t)) \nabla_{\theta} \log \pi_{\theta}(a_t | h_t)$$

env.verify 对应奖励  $R(\tau)$ ；env.state / env.step 与智能体执行框架共同产生轨迹  $\tau$  及状态历史  $h_t$ 。

optim\_step 执行参数更新；backward 计算梯度；forward 计算策略的动作概率。

译注 / 与原文翻译分开

公式中的变量和函数符号保留原样。

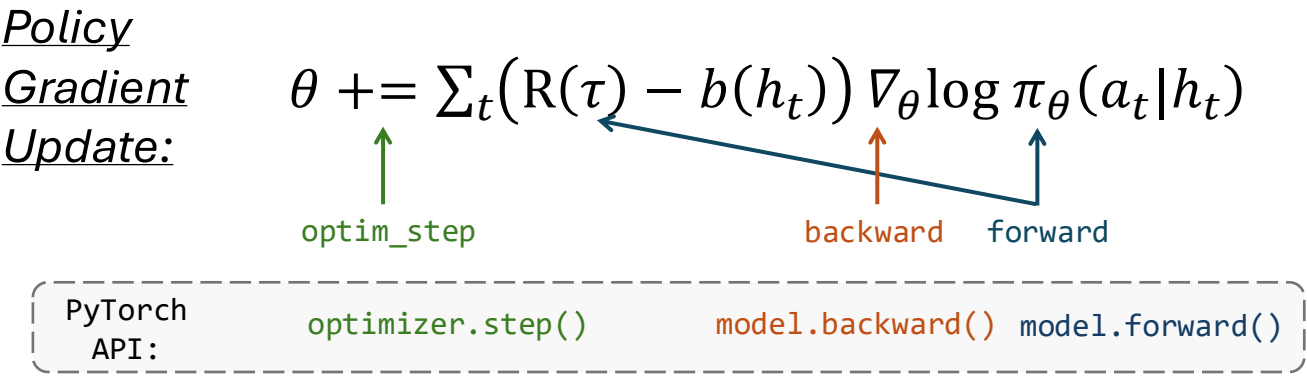
LLM 强化学习框架必须支持哪些操作？

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Why not just hand-roll custom implementation in vanilla PyTorch?



4

为什么不直接用原生 PyTorch 手写实现？

策略梯度更新：

$$\theta \ += \sum_t (R(\tau) - b(h_t)) \nabla_{\theta} \log \pi_{\theta}(a_t | h_t)$$

PyTorch API: `optimizer.step()` 对应 `optim_step`; `model.backward()` 对应 `backward`; `model.forward()` 对应 `forward`。

译注 / 与原文翻译分开

公式和 API 调用保留原样。

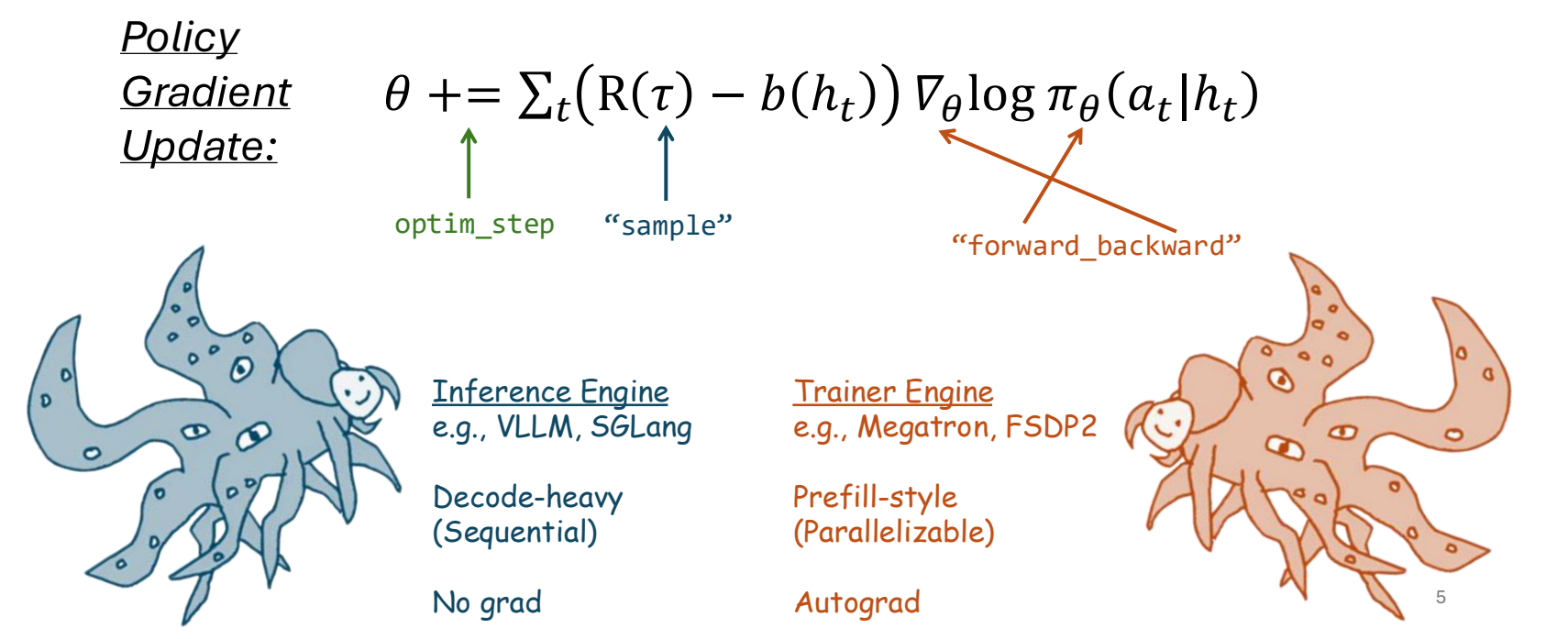
为什么不直接用原生 PyTorch 手写实现？

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

RL Frameworks are composed of two fundamentally different workloads!



强化学习框架由两种本质不同的工作负载组成！

策略梯度更新：

$$\theta \ += \sum_t (R(\tau) - b(h_t)) \nabla_{\theta} \log \pi_{\theta}(a_t | h_t)$$

采样轨迹由推理引擎执行，例如 vLLM、SGLang：逐词元解码为主（顺序执行），不计算梯度。

前向与反向计算由训练引擎执行，例如 Megatron、FSDP2：预填充式（可并行），使用自动微分。

optim\_step 更新参数；sample 采样；forward\_backward 执行前向与反向计算。

译注 / 与原文翻译分开

公式中的变量和函数符号保留原样。

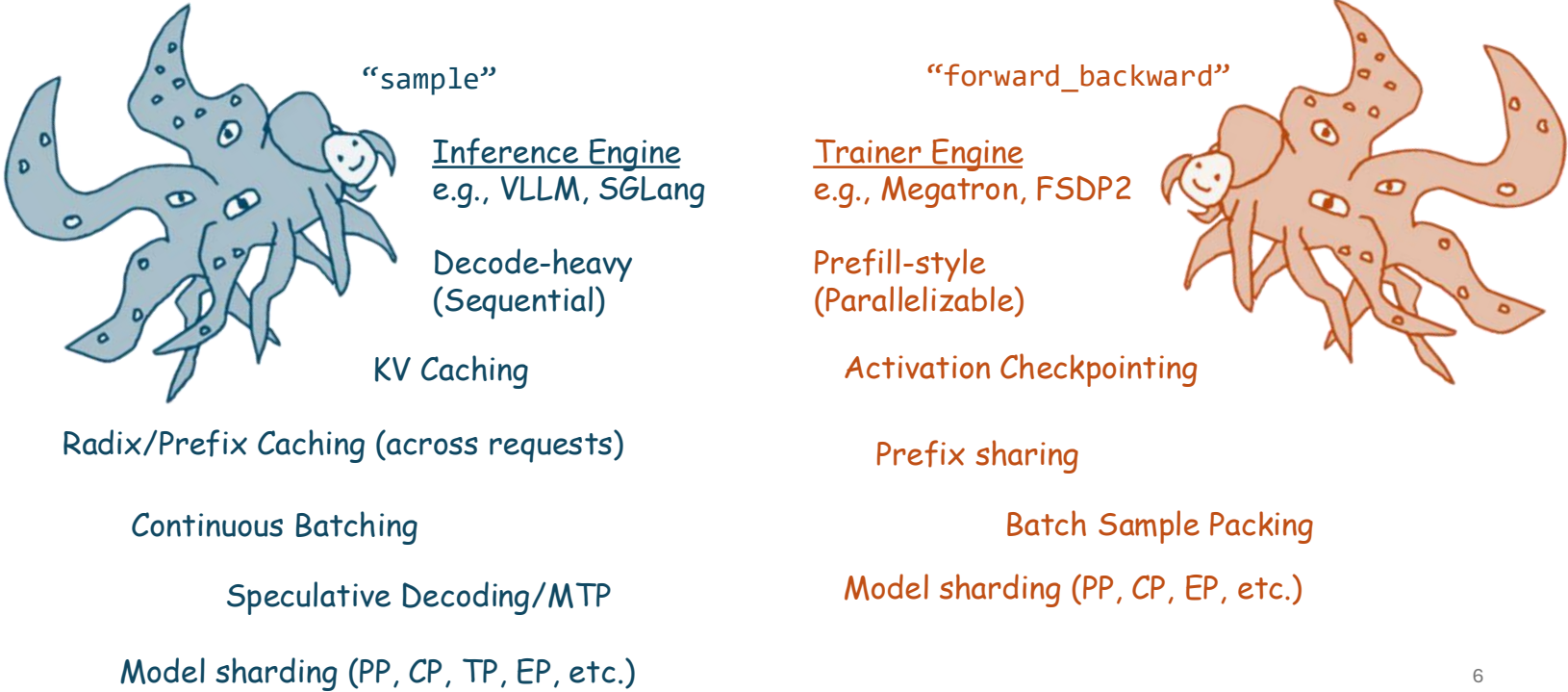
强化学习框架由两种本质不同的工作负载组成

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

RL Frameworks are composed of two fundamentally different workloads!



6

推理引擎（例如 vLLM、SGLang）：逐词元解码为主（顺序执行）。优化方法包括 KV 缓存、Radix/前缀缓存（跨请求）、连续批处理、推测解码/MTP，以及模型分片（PP、CP、TP、EP 等）。

训练引擎（例如 Megatron、FSDP2）：预填充式（可并行）。优化方法包括激活检查点、前缀共享、批次样本打包，以及模型分片（PP、CP、EP 等）。

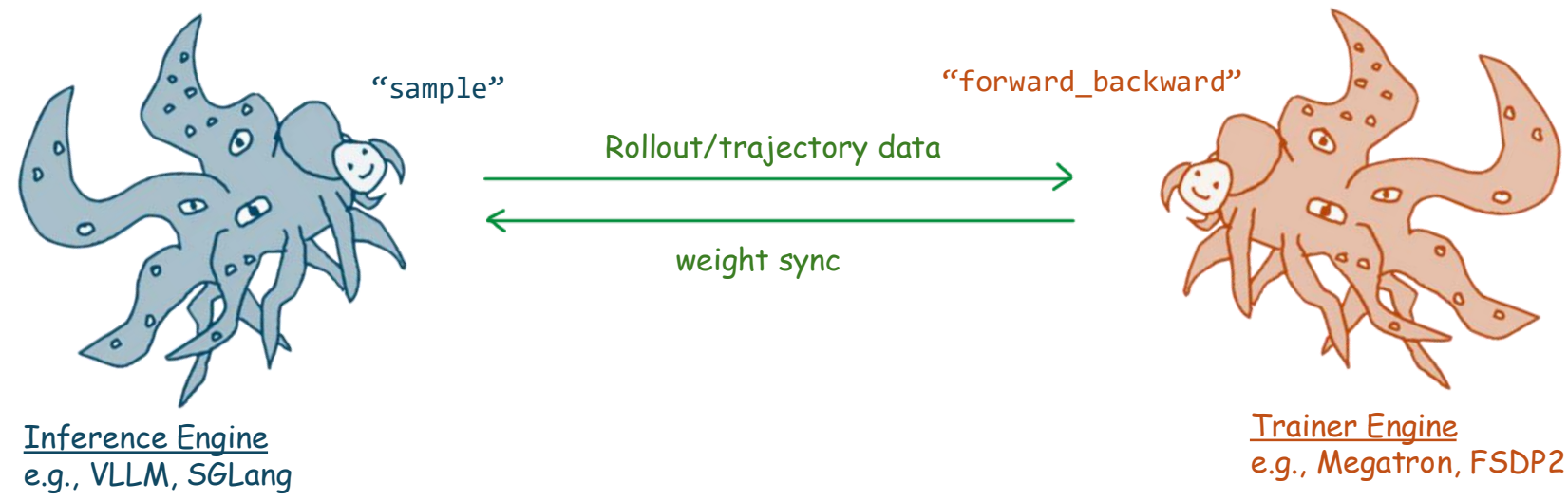
强化学习框架由两种本质不同的工作负载组成

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Interaction Between Train and Inference Engines



7

训练引擎与推理引擎之间的交互。

推理引擎（例如 vLLM、SGLang）向训练引擎发送采样轨迹数据。

训练引擎（例如 Megatron、FSDP2）向推理引擎同步权重。

推理端执行 sample；训练端执行 forward\_backward。

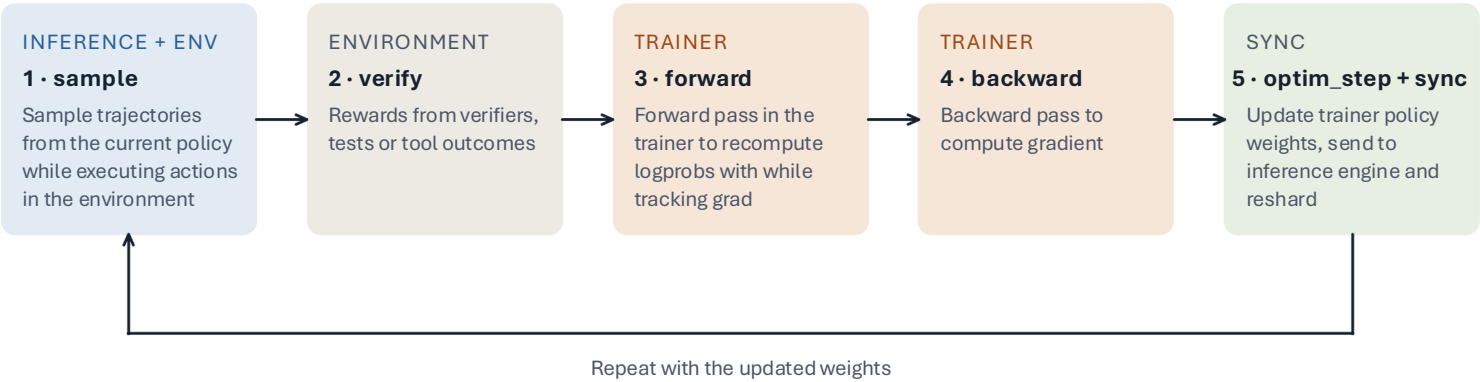
训练引擎与推理引擎之间的交互

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Breaking down a training step



8

- 1 · 推理 + 环境：从当前策略采样轨迹，并在环境中执行动作。
- 2 · 环境验证：从验证器、测试或工具结果获得奖励。
- 3 · 训练端前向计算：训练端执行前向计算，重新计算 logprob，同时跟踪梯度。
- 4 · 训练端反向计算：执行反向传播以计算梯度。
- 5 · 优化步骤 + 同步：更新训练策略权重，将权重发送到推理引擎并重新分片。

使用更新后的权重重复以上过程。

译注 / 与原文翻译分开

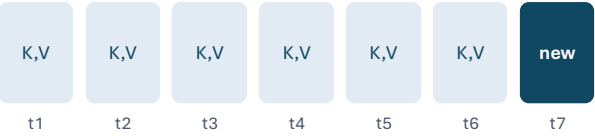
logprob 按原术语保留。

拆解一次训练步骤

# Inference Optimizations: KV Caching



## KV CACHE · ONE ENTRY PER PAST TOKEN



Step t7 computes q, k, v for **the new token only**, attends over every cached k, v, then appends its own.

## HOW BIG IS IT?

- Llama-3-8B in bf16:
- **128 KB per token**
  - 1 32k-token trace  $\approx$  **4 GB**;
  - 64 of them  $\approx$  **256 GB**

**KV Caching trades a lot of recomputation for a lot of memory.**

推理优化：KV 缓存。

KV 缓存：每个过去的词元对应一条缓存记录。

第  $t_7$  步只为新词元计算 q、k、v；对所有已缓存的 k、v 执行注意力计算，然后追加新词元自己的 k、v。

缓存大小：Llama-3-8B 使用 bf16 时，每个词元 128 KB；一条 32k 词元的轨迹约 4 GB；64条约 256 GB。

KV 缓存用大量内存换取大量重复计算的减少。

译注 / 与原文翻译分开

图中的 32k 按原文保留。

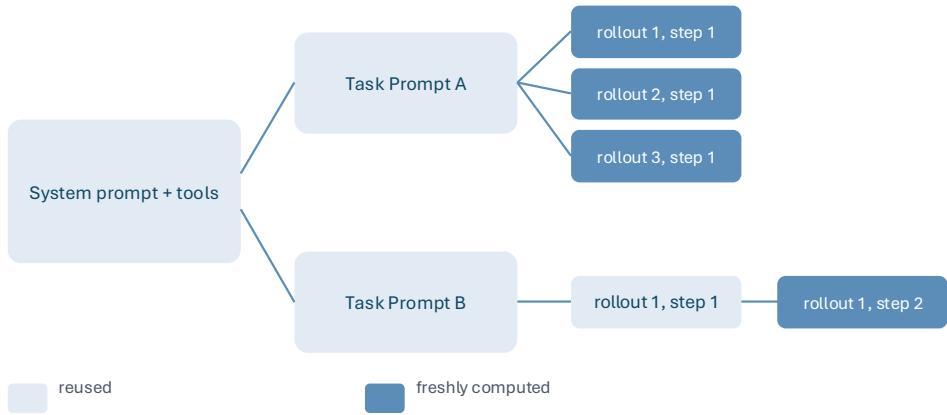
推理优化：KV 缓存

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Inference Optimizations: Prefix Caching



## Agentic RL especially benefits from prefix caching.

- In Group Rollouts (e.g., GRPO), many rollouts share the same task prompts.
- KV from past steps/history of a rollout can be reused in future steps.

[SGLang RadixAttention](#), [vLLM Automatic Prefix Caching](#)

推理优化：前缀缓存。

系统提示词 + 工具可在多个任务之间复用；任务提示词 A、B 各自进入采样轨迹。图中标注了 rollout 1、2、3 的第1步，以及 rollout 1 的第2步；复用部分标为 reused，新计算部分标为 freshly computed。

智能体强化学习尤其能从前缀缓存中受益。

在组内采样轨迹（例如 GRPO）中，许多轨迹共享相同任务提示词。

同一条采样轨迹先前步骤或历史记录中的 KV 可在后续步骤中复用。

SGLang RadixAttention · vLLM Automatic Prefix Caching

推理优化：前缀缓存

第 12 讲 · 强化学习系统

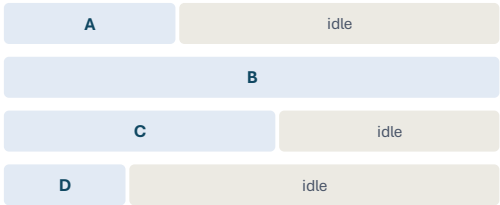
EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Inference Optimizations: Continuous Batching

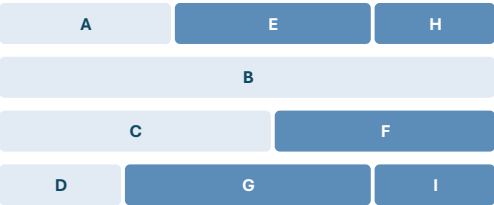


STATIC BATCHING



Every slot waits for the longest request (B).

CONTINUOUS BATCHING



Freed slots are refilled at the next step.

Schedule at the granularity of one decode step: finished sequences leave the batch at once, and queued requests take their slots.

Usually accompanied with chunked prefill to avoid long prompts from causing decode latency spikes. Alternatively, can disaggregate prefill and decode.

Yu et al., 2022 (Orca, iteration-level scheduling)

推理优化：连续批处理。

静态批处理：每个位置都要等待最长的请求（B）完成。连续批处理：空出的处理位置会在下一步补入新请求。

调度粒度为一次逐词元解码步骤：序列结束后立即离开批次，排队中的请求接替其位置。示意图用 A–D 表示原有请求，用 E–I 表示后续补入的请求。

通常会配合分块预填充，避免长提示词导致解码延迟突增。另一种做法是将预填充与解码分离。

Yu 等, 2022 (Orca, 迭代级调度)

推理优化：连续批处理

第 12 讲 · 强化学习系统

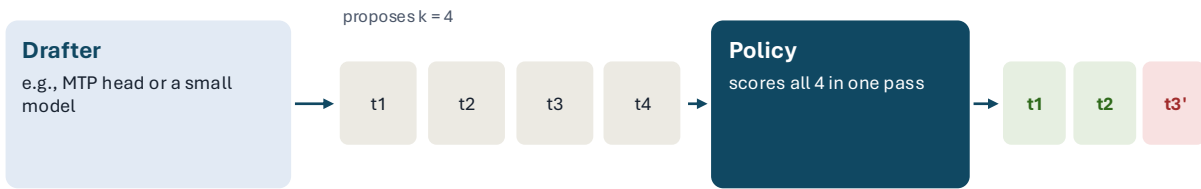
EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Inference Optimizations: Speculative Decoding



A cheap drafter guesses k tokens; the policy checks all k in **one parallel pass**. Rejection sampling keeps the policy's exact distribution.



**RL-SPECIFIC CATCH**  
The policy changes every training step, so a frozen drafter's acceptance rate decays. Train it online with an auxiliary objective (e.g., SFT, distillation).

推理优化：推测解码。

低成本的起草器猜测 k 个词元；策略通过一次并行计算检查全部 k 个词元。拒绝采样保留策略原有的精确分布。

起草器（例如 MTP head 或小型模型）提出 k=4 个词元；策略一次性为这4个词元评分。图中示意起草序列  $t_1$ 、 $t_2$ 、 $t_3$ 、 $t_4$ ，策略接受到  $t_1$ 、 $t_2$ ，并将后续候选改为  $t_3'$ 。

强化学习特有的难点：策略每个训练步骤都会变化，因此冻结的起草器接受率会下降。可使用辅助目标（例如 SFT、蒸馏）在线训练起草器。

Leviathan 等, 2023

推理优化：推测解码

第 12 讲 · 强化学习系统

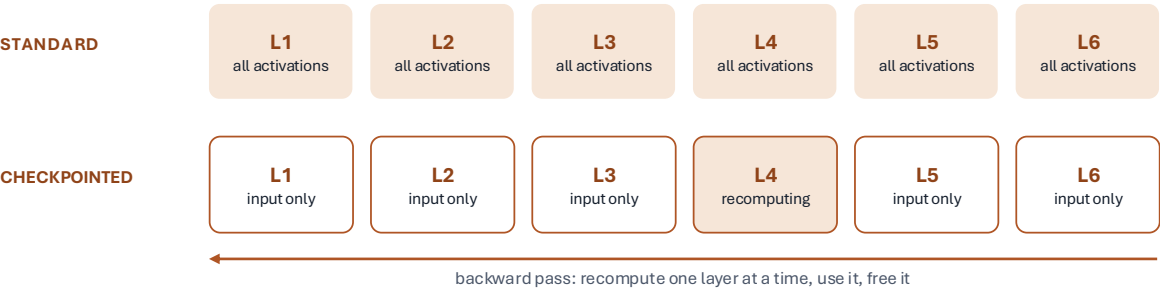
EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Trainer Optimizations: Activation Checkpointing



Keep only each layer's **input** in the forward pass; recompute the layer's internals when the backward pass reaches it.



**MEMORY**

From every layer's activations to one input per layer, plus a single layer's activations at a time.

**COST**

Full recompute adds ~one forward pass: **≈ +33% compute**. Selective recompute (only attention) costs far less.

Chen et al., 2016 (gradient checkpointing); Korthikanti et al., 2022 (selective activation recomputation).

训练优化：激活检查点。

前向计算时只保留每一层的输入；反向计算到达该层时，再重新计算这一层的内部激活。

标准方式：L1–L6 的所有激活都保留。使用检查点：L1、L2、L3、L5、L6 仅保留输入；反向传播到 L4 时重新计算。反向阶段一次重算一层，使用后释放。

内存：从保存所有层的激活，变为每层保存一个输入，并在任一时刻只保存单层激活。

计算成本：全部重算会增加约一次前向计算，计算量约增加33%；选择性重算（仅注意力层）成本低得多。

Chen 等, 2016 (梯度检查点) ； Korthikanti 等, 2022 (选择性激活重算)

## 训练优化：激活检查点

第 12 讲 · 强化学习系统

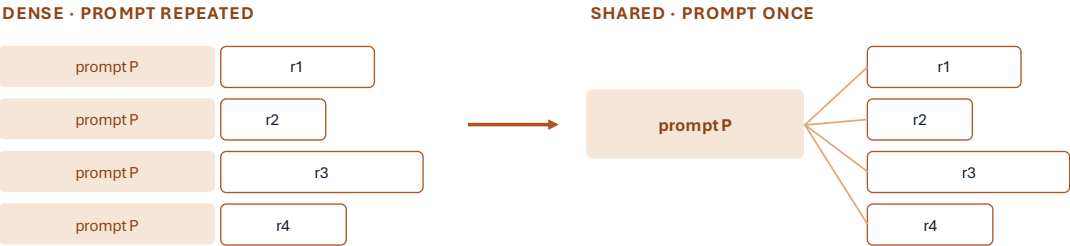
EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



# Trainer Optimizations: Prefix Sharing

RL batches repeat prefixes: the prompt across a GRPO group, the history across turns. **Compute each once, forward and backward.** (Counter-part to prefix caching in inference)



Tree attention mask (no attention across branches) · position IDs as in each standalone sequence · gradients from every branch accumulate into P

**SHARED PROMPTS · SNOWFLAKE ZoRRO**

In long-prompt GRPO batches, **80–95%** of tokens are duplicate prompt tokens. Deduplicating them gives up to **6×** faster actor updates.

**PREFIX TREES · AReaL-DTA**

Multi-turn  $\tau^2$ -bench rollouts compress **9.43×** as a prefix tree. A DFS walks one root-to-leaf path at a time: up to **8.31×** vs dense training.

[Snowflake AI Research, 2026 \(ZoRRO, Arctic RL\)](#); [Zhang et al., 2026 \(AReaL-DTA: Dynamic Tree Attention, ICML\)](#).

训练优化：前缀共享。

强化学习批次会重复使用前缀：GRPO 组内共享提示词，多轮对话间共享历史。前向和反向计算时，每个前缀只计算一次。（这是推理前缀缓存的对应方法。）

密集批次——提示词重复：每个样本分别包含提示词 P 和轨迹  $r_1$ 、 $r_2$ 、 $r_3$ 、 $r_4$ 。共享批次——提示词仅出现一次：提示词 P 与  $r_1$ – $r_4$  共享。

使用树形注意力掩码（不同分支之间不互相注意）；位置 ID 与各自独立序列中的位置一致；每个分支的梯度累积到 P。

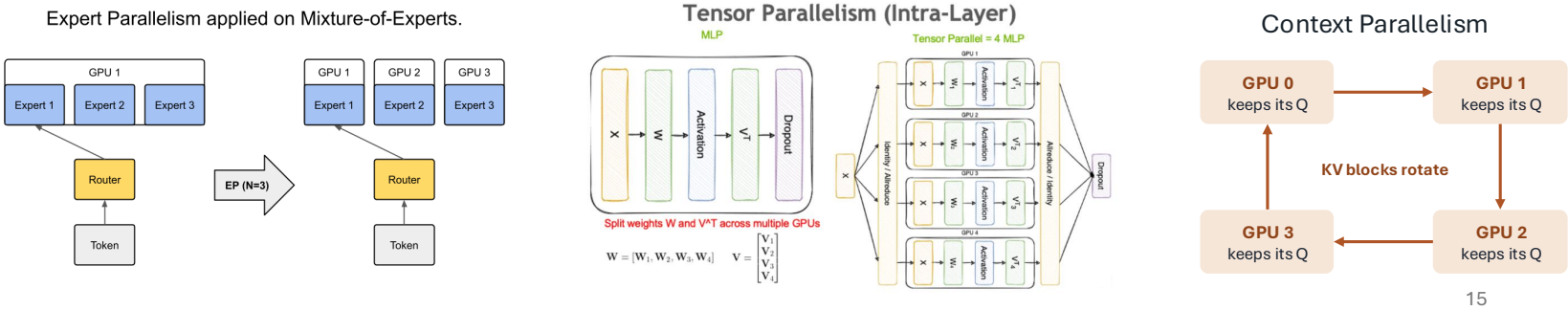
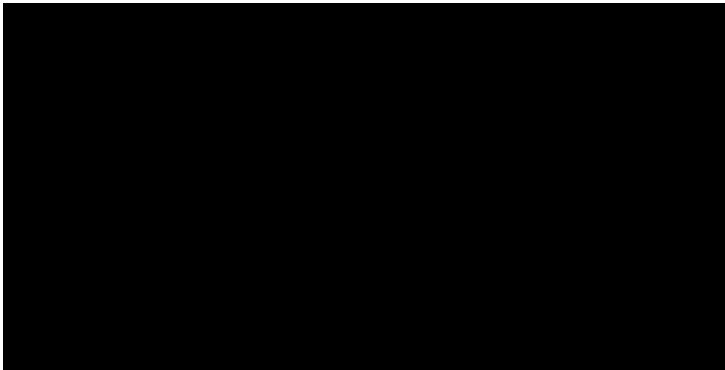
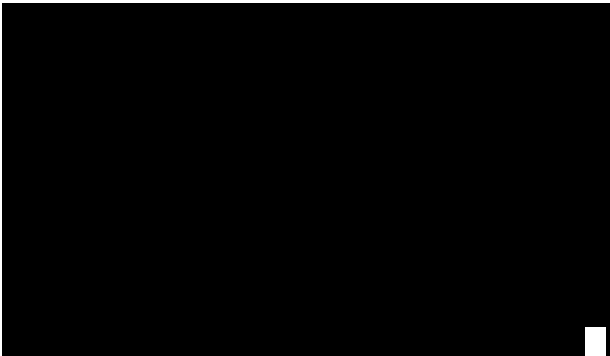
共享提示词 · Snowflake ZoRRO：长提示词 GRPO 批次中，80–95% 的词元是重复提示词词元；去重后，actor 更新速度最多可提高6倍。

前缀树 · AReaL-DTA：多轮  $\tau^2$ -bench 采样轨迹压缩为前缀树后达到9.43倍压缩。深度优先搜索（DFS）每次遍历一条从根到叶的路径；相较密集训练最多快8.31倍。

Snowflake AI Research, 2026 (ZoRRO、Arctic RL) ； Zhang 等, 2026 ( AReaL-DTA: 动态树注意力, ICML)

## 训练优化：前缀共享

# Optimizations: Parallelism



优化：并行化

优化：并行化。

专家并行应用于混合专家模型（Mixture-of-Experts）：Token 经过 Router 后分发给不同 GPU 上的 Expert；EP（N=3）示例包含 Expert 1、2、3。

张量并行（层内）：将 MLP 的权重 W 和  $V^T$  分到多个 GPU；示例为 4 路张量并行。图中列出权重、激活、dropout 等层内计算。

上下文并行：GPU 0、GPU 1、GPU 2、GPU 3 各自保留自己的 Q；KV 块在 GPU 之间轮转。

Parallelisms · NVIDIA NeMo Framework User Guide

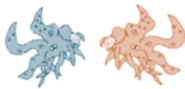
译注 / 与原文翻译分开

并行化示意图中的部分算子标签较小；已转写主要可辨标签。

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



# Train-Inference Interactions: Weight Sync

## Full Weight Sync:

- Full broadcast to every inference rank.
- 1T parameter model → ~2TB sync

## • Delta Weight Sync:

- ~99% of parameters unchanged between RL weight updates bf16 change.
- Sync only the delta.

Note: Parameters need to be re-sharded for inference engine parallelism layout.

Mihai & Belilovsky (PULSE), 2026

16

训练与推理交互：权重同步。

全量权重同步：向每个推理 rank 完整广播；一个拥有1T参数的模型需要同步约 2TB数据。

增量权重同步：在 RL 权重更新之间，约99%的参数保持不变（原图将其描述为 bf16 change）；只同步变化部分。

注意：参数必须重新分片，以适配推理引擎的并行布局。

Mihai 与 Belilovsky (PULSE) ， 2026

译注 / 与原文翻译分开

原图中 “bf16 change” 这一修饰语的语义不够明确，按原文保留并注明。

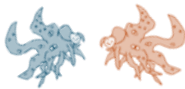
## 训练与推理交互：权重同步

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Train-Inference Interactions: Rollout Routing Replay (R3)



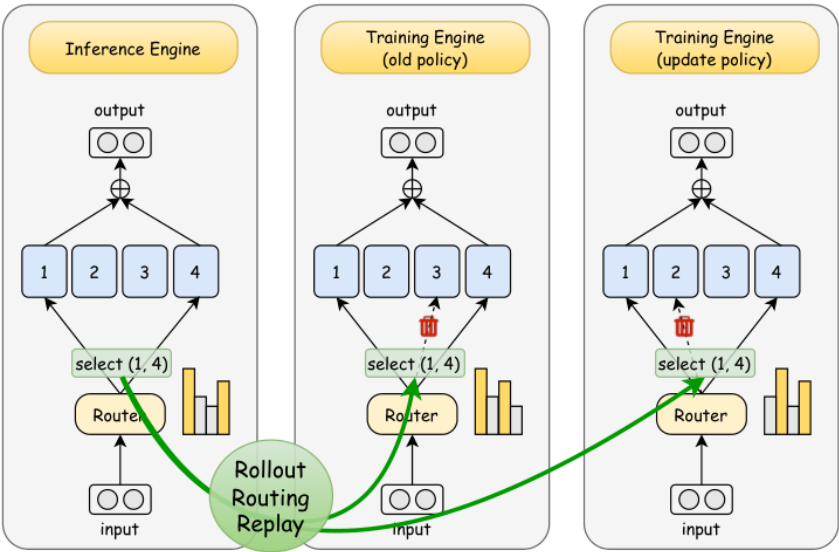
### Problem

Tiny numerical differences can select different MoE experts.

### R3 Solution

Record rollout expert choices and replay the selections in training.

Inference engine must provide routing decisions to training engine.



Ma et al., 2025, R3, Fig. 1

训练与推理交互：采样轨迹路由重放（R3）。

问题：微小数值差异可能导致选择不同的 MoE 专家。

R3 解决方案：记录采样轨迹中的专家选择，并在训练时重放这些选择。

推理引擎必须向训练引擎提供路由决策。

Ma 等, 2025, R3, 图1

## 训练与推理交互：采样轨迹路由重放（R3）

智能体强化学习：关键难点

# Agentic RL: The Sharp Bits

18

智能体强化学习：关键难点

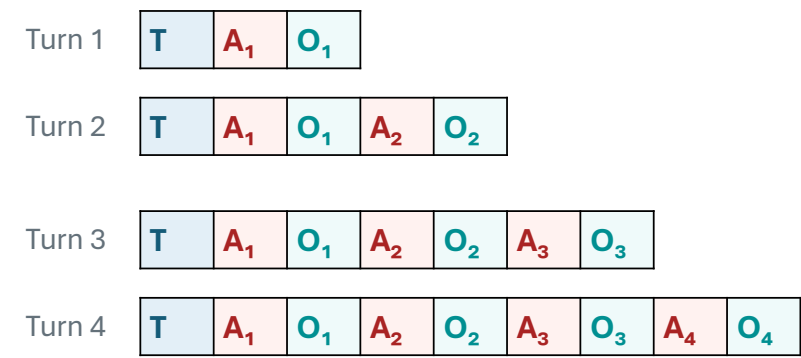
第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Sequence Extension

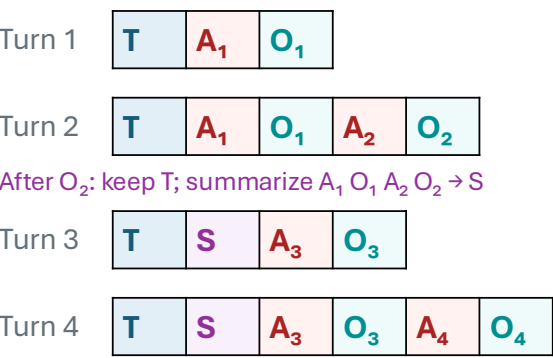
**Harness A**  
Append to the existing history



T = task    A<sub>i</sub> = action    O<sub>i</sub> = observation    S = compacted interaction history

**Which harness is better for training efficiency?**

**Harness B**  
Compact every two observations



执行框架 A：将新内容追加到现有历史。执行框架 B：每两次观测压缩一次。

T = 任务；A<sub>i</sub> = 动作；O<sub>i</sub> = 观测；S = 压缩后的交互历史。

执行框架 A：第1轮 T、A<sub>1</sub>、O<sub>1</sub>；第2轮 T、A<sub>1</sub>、O<sub>1</sub>、A<sub>2</sub>、O<sub>2</sub>；之后继续追加至 A<sub>4</sub>、O<sub>4</sub>。

执行框架 B：第2次观测 O<sub>2</sub> 后保留 T，将 A<sub>1</sub>、O<sub>1</sub>、A<sub>2</sub>、O<sub>2</sub> 总结为 S；第3轮使用 T、S、A<sub>3</sub>、O<sub>3</sub>；第4轮继续追加 A<sub>4</sub>、O<sub>4</sub>。

哪个执行框架的训练效率更高？

404: Sequence Extension · Tinker 文档

序列扩展

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Sequence Extension: Training Batch

One trajectory advantage  $\hat{A}$ , broadcast to every action token

Harness A: one concatenated datapoint

|            |      |                |                |                |                |                |                |                |                |
|------------|------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| Token span | Task | A <sub>1</sub> | O <sub>1</sub> | A <sub>2</sub> | O <sub>2</sub> | A <sub>3</sub> | O <sub>3</sub> | A <sub>4</sub> | O <sub>4</sub> |
| Loss mask  | 0    | 1              | 0              | 1              | 0              | 1              | 0              | 1              | 0              |
| Advantage  | 0    | $\hat{A}$      | 0              | $\hat{A}$      | 0              | $\hat{A}$      | 0              | $\hat{A}$      | 0              |

Harness B: two datapoints, both containing Task

|            |      |                |                |                |                |      |   |                |                |                |                |
|------------|------|----------------|----------------|----------------|----------------|------|---|----------------|----------------|----------------|----------------|
| Token span | Task | A <sub>1</sub> | O <sub>1</sub> | A <sub>2</sub> | O <sub>2</sub> | Task | S | A <sub>3</sub> | O <sub>3</sub> | A <sub>4</sub> | O <sub>4</sub> |
| Loss mask  | 0    | 1              | 0              | 1              | 0              | 0    | 0 | 1              | 0              | 1              | 0              |
| Advantage  | 0    | $\hat{A}$      | 0              | $\hat{A}$      | 0              | 0    | 0 | $\hat{A}$      | 0              | $\hat{A}$      | 0              |

Repeated task tokens have zero direct loss, but still require context computation.

Preserve exact prefixes and concatenate for as long as practical.

24

一条轨迹的优势  $\hat{A}$  广播到每个动作词元。

执行框架 A：一个拼接后的数据点。词元跨度：Task、A<sub>1</sub>、O<sub>1</sub>、A<sub>2</sub>、O<sub>2</sub>、A<sub>3</sub>、O<sub>3</sub>、A<sub>4</sub>、O<sub>4</sub>。Loss mask：0、1、0、1、0、1、0、1、0。Advantage：0、 $\hat{A}$ 、0、 $\hat{A}$ 、0、 $\hat{A}$ 、0、 $\hat{A}$ 、0。

执行框架 B：两个数据点，二者都包含 Task。第一个点为 Task、A<sub>1</sub>、O<sub>1</sub>、A<sub>2</sub>、O<sub>2</sub>；第二个点为 Task、S、A<sub>3</sub>、O<sub>3</sub>、A<sub>4</sub>、O<sub>4</sub>。

Loss mask：第一个点 0、1、0、1、0；第二个点 0、0、1、0、1、0。Advantage：第一个点 0、 $\hat{A}$ 、0、 $\hat{A}$ 、0；第二个点 0、0、 $\hat{A}$ 、0、 $\hat{A}$ 、0。

重复的任务词元没有直接损失，但仍需计算其上下文。

尽可能保留完全相同的前缀，并在可行范围内尽量拼接。

译注 / 与原文翻译分开

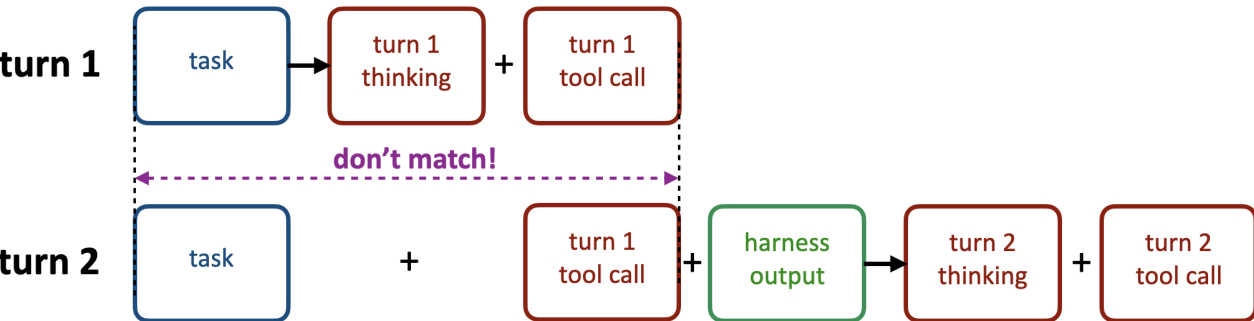
表格中的符号和数值按原图保留。

序列扩展：训练批次

# Chat Template Quirks

Popular chat templates may not satisfy the sequence extension property.

E.g., Qwen 3 / Qwen 3.5’s default chat template removes reasoning from past history.



No Token Left Behind: Demystifying Token-In-Token-Out in Miles

常用聊天模板可能不满足序列扩展性质。

例如，Qwen 3 / Qwen 3.5 的默认聊天模板会从过去的历史中删除推理内容。

图示：第1轮包含 task、turn 1 thinking 和 turn 1 tool call；第2轮模板重新组织历史，只留下 task、turn 1 tool call、执行框架输出（harness output）、turn 2 thinking、turn 2 tool call，前后序列“无法匹配”。

No Token Left Behind: Demystifying Token-In-Token-Out in Miles

## 聊天模板的细节问题

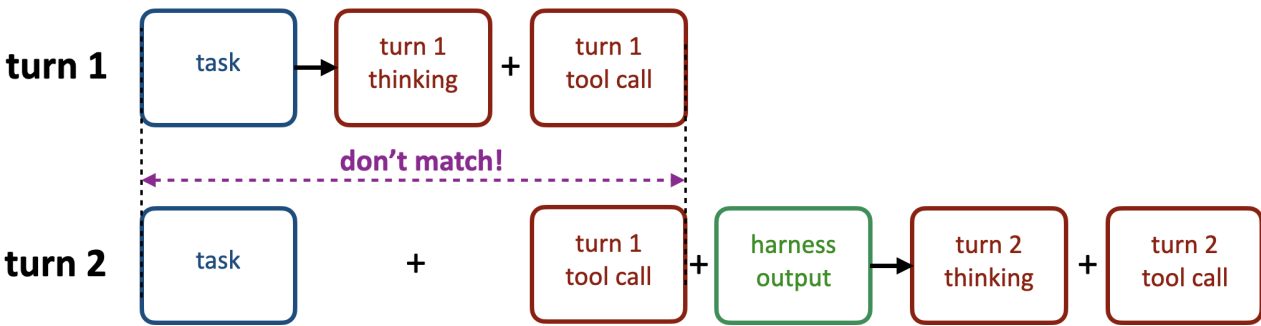
第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

# Chat Template Quirks

E.g., Qwen 3 / Qwen 3.5’s default chat template removes reasoning from past history.



Agent RL training frameworks often ship with their own custom chat template or “renderers” that preserve sequence extension (e.g., Tinker, PrimeRL).

例如，Qwen 3 / Qwen 3.5 的默认聊天模板会从过去的历史中删除推理内容。

图示：第1轮包含 task、turn 1 thinking 和 turn 1 tool call；第2轮保留 task 与 turn 1 tool call，再接入 执行框架输出（harness output）、turn 2 thinking 和 turn 2 tool call。重新生成的历史与最初序列 “无法匹配” 。

智能体 RL 训练框架通常会提供自己的聊天模板或 renderer，以保留序列扩展性质（例如 Tinker、PrimeRL）。

## 聊天模板的细节问题

# Token-In, Token-out (TITO)

In what format should we store and transport rollout/trajectory data from the inference engines to the trainer engines?

- 1. String
- 2. Tokens

23

应以什么格式存储并传输从推理引擎到训练引擎的采样轨迹数据？

- 1. 字符串。
- 2. 词元。

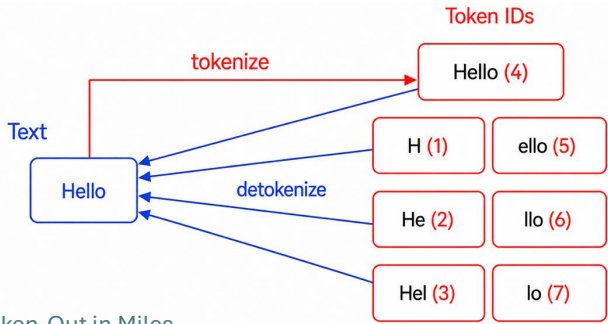
词元输入、词元输出 (TITO)

# Token-In, Token-out (TITO)

In what format should we store and transport rollout/trajectory data from the inference engines to the trainer engines?

1. ~~String~~

2. Tokens
- *Small, inconsistent transformations in string space can lead to inconsistent tokenizations between train and inference (e.g., chat template adding some whitespace)*
  - *Tokenization is one-to-one, but de-tokenization is many-to-one.*



No Token Left Behind: Demystifying Token-In-Token-Out in Miles

应以什么格式存储并传输从推理引擎到训练引擎的采样轨迹数据？

1. 字符串（原图用红色删除线划去）。2. 词元（原图用绿色标示）。

字符串空间中微小且不一致的转换，会导致训练与推理阶段的分词结果不一致（例如聊天模板额外添加空白字符）。

分词是一对一映射，但反分词是多对一映射。

图中 Text 表示文本，Token IDs 表示词元编号；tokenize 表示分词，detokenize 表示反分词。文本 Hello 分词为 Hello (4) ；Hello (4) 、H (1) + ello (5) 、He (2) + llo (6) 、Hel (3) + lo (7) 这四种词元序列，反分词后都得到同一个文本 Hello。

No Token Left Behind: Demystifying Token-In-Token-Out in Miles

## 词元输入、词元输出 (TITO)

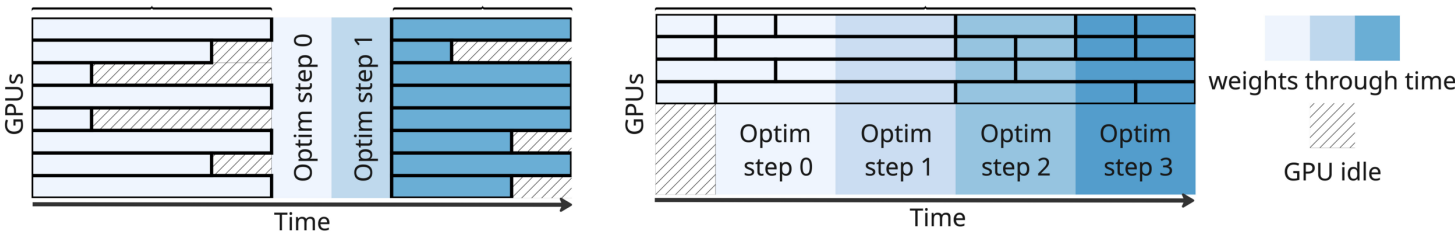
第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Async RL

Overlap inference and weight updates. Reduces GPU idle due to very long rollouts. Requires staleness/off-policy corrections.



Sync RL

- Co-locate Train & Inference Engines (same GPUs shared by both)

Async RL

- Split GPUs between Train & Inference
- Agentic RL is often rollout-bound (long trajectories), so can dedicate more GPUs to inference (e.g., 3:1 ratio)
- Requires *continuous batching* and *in-flight weight-update support*.

[Pipeline RL](#), [AReal](#)

异步强化学习。

让推理与权重更新重叠进行，从而减少超长采样轨迹造成的 GPU 空闲。需要进行陈旧性 / 离策略校正。

同步 RL：训练引擎与推理引擎共置，二者共享同一组 GPU。

异步 RL：训练与推理分用不同 GPU。智能体 RL 往往受采样轨迹限制（轨迹很长），因此可以给推理分配更多 GPU（例如 3:1）。需要支持连续批处理与在途权重更新。

图示：按 GPU 与时间展示优化步骤 0–3；颜色表示权重随时间变化，阴影标出 GPU 空闲。

Pipeline RL · AReal

异步强化学习

# In-flight Weight Updates

Inference frameworks have evolved to accommodate RL systems needs.

Example from [SGLang Docs](#).

## Pause Generation

Endpoint: `POST /pause_generation`

Request body:

| Field             | Description | Defaults           | Options                                                           |
|-------------------|-------------|--------------------|-------------------------------------------------------------------|
| <code>mode</code> | Pause mode. | <code>abort</code> | <code>abort</code> , <code>retract</code> , <code>in_place</code> |

Modes:

- `abort` : Default behavior, identical to `abort` endpoint with `abort_all` set. Pending requests from `waiting_queue` and `running_queue` will be returned immediately to the caller.
- `retract` : Put engine in “paused” state. Move running requests back to waiting queue. KV cache can be flushed and recomputed later.
- `in_place` : Put engine in “paused” state without changing states of the requests. Running requests rely on availability of KV caches to continue, so any subsequent `flush_cache` call will be unsuccessful.

## Continue Generation

Endpoint: `POST /continue_generation`

在途权重更新。

推理框架不断演进，以满足强化学习系统的需求。

SGLang 文档中的示例：暂停生成。Endpoint: `POST /pause_generation`。请求体字段 `mode`（暂停模式），默认值 `abort`；可选值 `abort`、`retract`、`in_place`。

模式说明：`abort` 为默认行为，与设置 `abort_all` 的 `abort` endpoint 相同；`waiting_queue` 和 `running_queue` 中的待处理请求会立即返回调用方。

`retract`：将引擎置为 `paused` 状态，并将运行中的请求移回 `waiting queue`；KV 缓存可被清空，稍后重新计算。

`in_place`：将引擎置为 `paused` 状态，但不改变请求状态。运行中的请求依赖 KV 缓存继续执行，因此后续调用 `flush_cache` 会失败。

继续生成。Endpoint: `POST /continue_generation`。

示例来自 SGLang Docs。

译注 / 与原文翻译分开

HTTP 方法、路径、字段、枚举值及队列名保留原样。

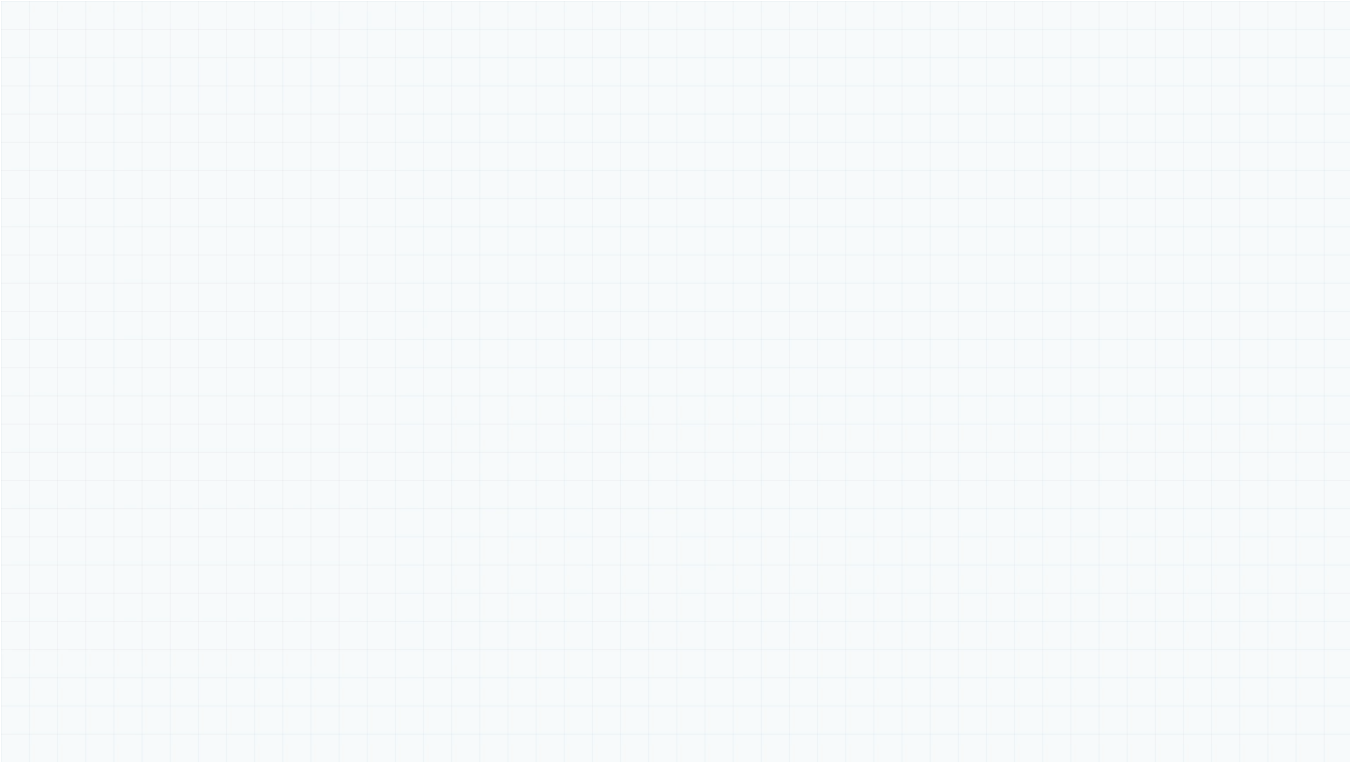
在途权重更新

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Complex Harnesses (Sub-agents, Context compaction, etc.)



[Recursive Agent Optimization \(RAO\), 2026](#); [Recursive Language Models \(RLM\), 2025](#)

27

复杂执行框架（子智能体、上下文压缩等）。

[Recursive Agent Optimization \(RAO\), 2026](#); [Recursive Language Models \(RLM\), 2025](#)

复杂执行框架（子智能体、上下文压缩等）

系统设计选择

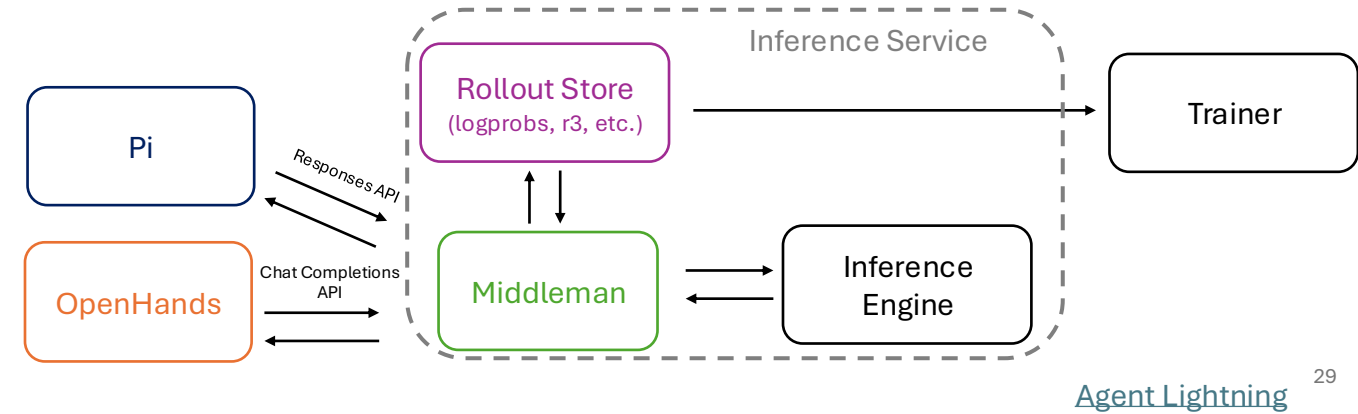
# System Design Choices

28

系统设计选择

# Harness Agnostic Design

- Many popular agent harnesses: Codex, Claude Code, OpenHands, Pi, Hermes, Prime Agent, etc.
- How to support all?
  - Anti-pattern:** Custom rollout/environment code for each harness.
  - Harness-agnostic pattern:** Middleman/Proxy service that emulates inference APIs (chat-completions, responses, anthropic, etc.) → 0 code changes when using new harness



许多常用智能体执行框架：Codex、Claude Code、OpenHands、Pi、Hermes、Prime Agent 等。如何同时支持它们？

反模式：为每个执行框架分别编写采样轨迹和环境代码。

与执行框架无关的模式：使用中间层 / 代理服务模拟推理 API（chat-completions、responses、anthropic 等）→ 使用新执行框架时无需修改代码。

图示：Pi 使用 Responses API，OpenHands 使用 Chat Completions API，二者接入中间层；中间层连接推理引擎，并将 rollout store（logprobs、R3 等）提供给训练器。该服务边界标为 Inference Service。

Agent Lightning

## 与执行框架无关的设计

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Single Program Multiple Data (SPMD)

One Python program runs on every GPU rank (simplified example pseudocode below)

```
rank = dist.get_rank()
train = dist.new_group([0, 1, 2, 3])
infer = dist.new_group([4, 5, 6, 7])

if rank < 4:
    while True:
        batch = next_batch_shard(train)
        forward_backward(batch, train)
        optimizer.step()
        publish_weights_async(train)
else:
    while True:
        refresh_weights_if_ready(infer)
        prompt = next_prompt(infer)
        trace = rollout(prompt, infer)
        enqueue_trajectory(trace, infer)
```

- All ranks**  
Create both groups in the same order.
- Ranks 0–3: training**  
FSDP collectives and optimizer updates
- Ranks 4–7: inference**  
Tensor/Context-parallel decoding and weight refresh

[AReal versions < 1.0](#)

30

Coordination lives in the program running on every rank.  
Straightforward with direct control of everything in a single program, but less flexible.

单程序、多数据 (SPMD)

单个 Python 程序运行在每个 GPU rank 上（以下为简化伪代码）。

```
rank = dist.get_rank()
train = dist.new_group([0, 1, 2, 3])
infer = dist.new_group([4, 5, 6, 7])
if rank < 4:
    while True:
        batch = next_batch_shard(train)
        forward_backward(batch, train)
        optimizer.step()
        publish_weights_async(train)
else:
    while True:
        refresh_weights_if_ready(infer)
        prompt = next_prompt(infer)
        trace = rollout(prompt, infer)
        enqueue_trajectory(trace, infer)
```

所有 rank：以相同顺序创建两个组。Rank 0–3：训练，执行 FSDP collective 和优化器更新。Rank 4–7：推理，执行张量 / 上下文并行解码及权重刷新。

协调逻辑位于每个 rank 上运行的程序中。

一个程序直接控制所有组件，做法直观，但灵活性较低。

AReal 1.0 之前的版本

译注 / 与原文翻译分开

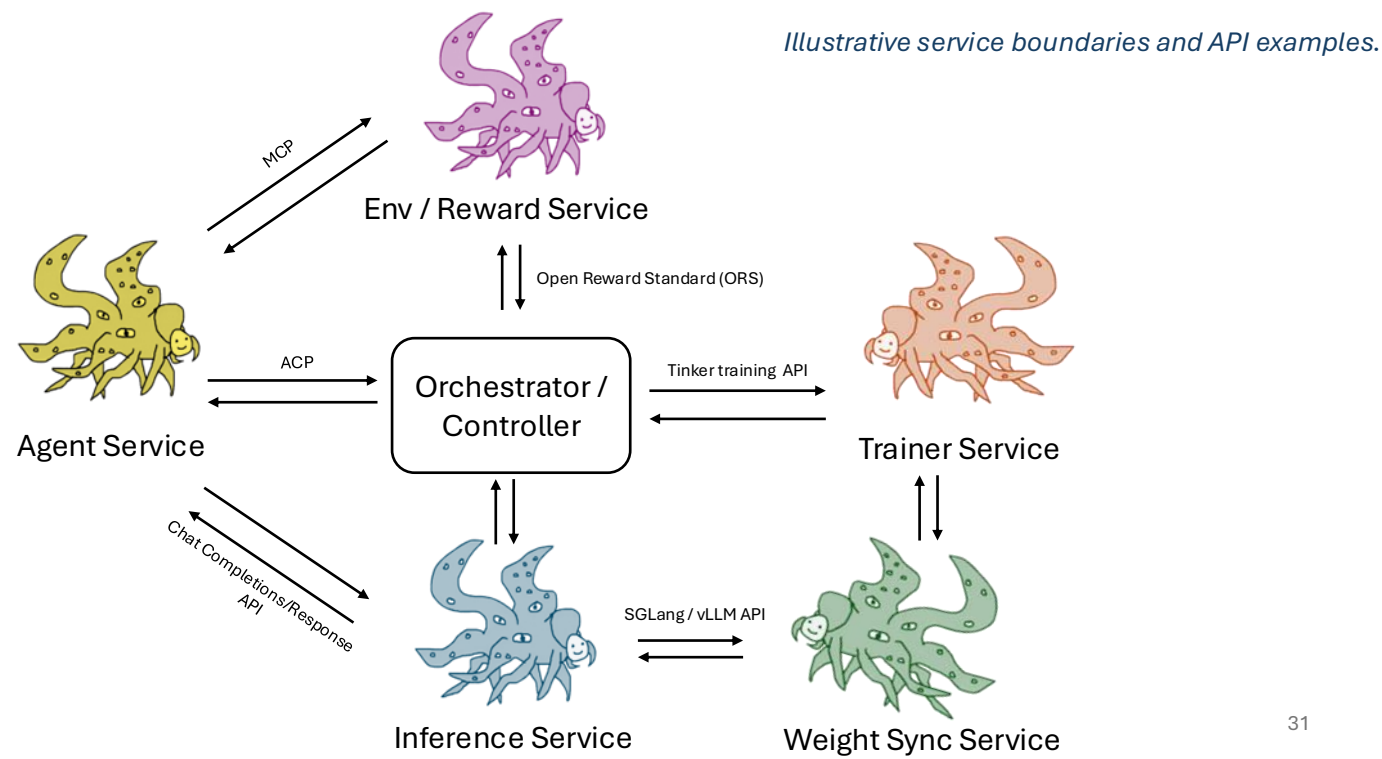
伪代码、函数名和 API 字段保留原样。

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Everything-as-a-service w/ Single Controller



31

一切皆服务，使用单一控制器。

图示为说明性服务边界与 API 示例：环境 / 奖励服务通过 MCP 与 Agent Service 通信；Agent Service 通过 ACP 与 Orchestrator / Controller 通信；环境 / 奖励服务使用 Open Reward Standard (ORS) ；控制器通过 Tinker training API 连接 Trainer Service；Inference Service 通过 SGLang / vLLM API 提供推理；Weight Sync Service 负责权重同步。

一切皆服务，使用单一控制器

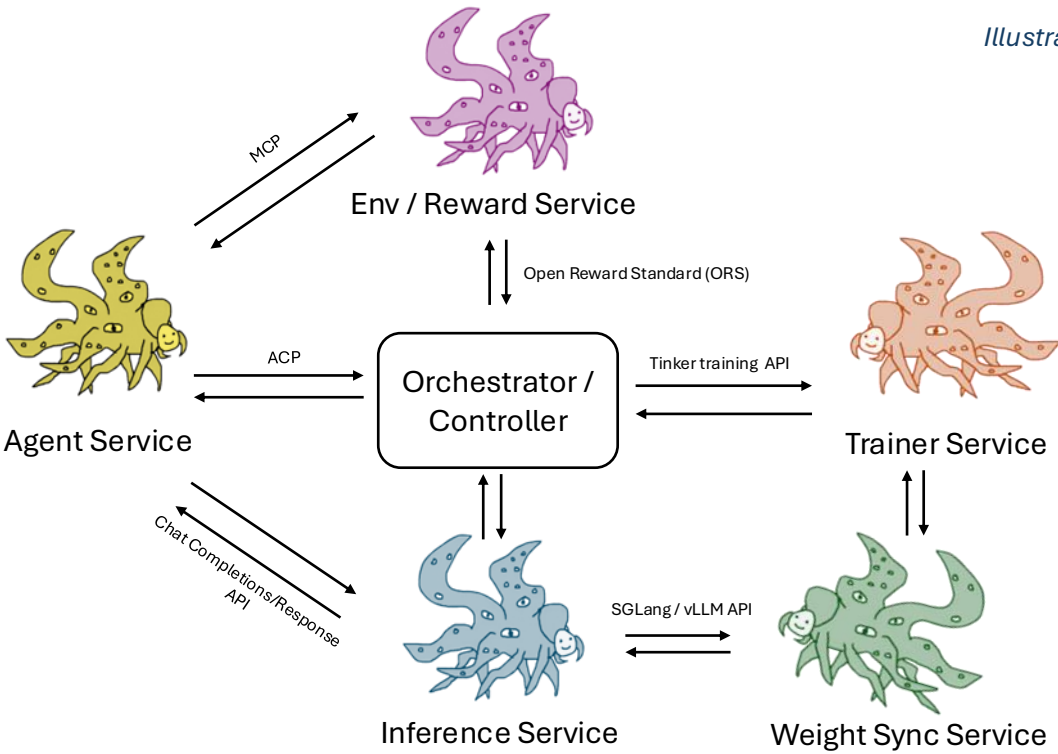
第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Everything-as-a-service w/ Single Controller

Illustrative service boundaries and API examples.



Separation of concerns (decoupling):

- Each of these components is fairly complex and specialized.
- Instead all living in the same codebase:
  - Lightweight orchestrator (CPU training loop) → researchers can easily prototype different algorithms without complex dependencies.
  - Project maintenance, updates and releases don't need to be synchronized.

32

说明性服务边界与 API 示例。

关注点分离（解耦）： 这些组件都相当复杂且各自专门化。

与其把所有组件放进同一个代码库，不如使用轻量级编排器（CPU 训练循环）；研究人员可以轻松试验不同算法，而不必处理复杂依赖。

项目维护、更新与发布不必同步进行。

图中服务与协议： Agent Service — ACP — Orchestrator / Controller; Env / Reward Service — Open Reward Standard (ORS) ; Controller — Tinker training API — Trainer Service; Inference Service — SGLang / vLLM API — Trainer Service; Weight Sync Service.

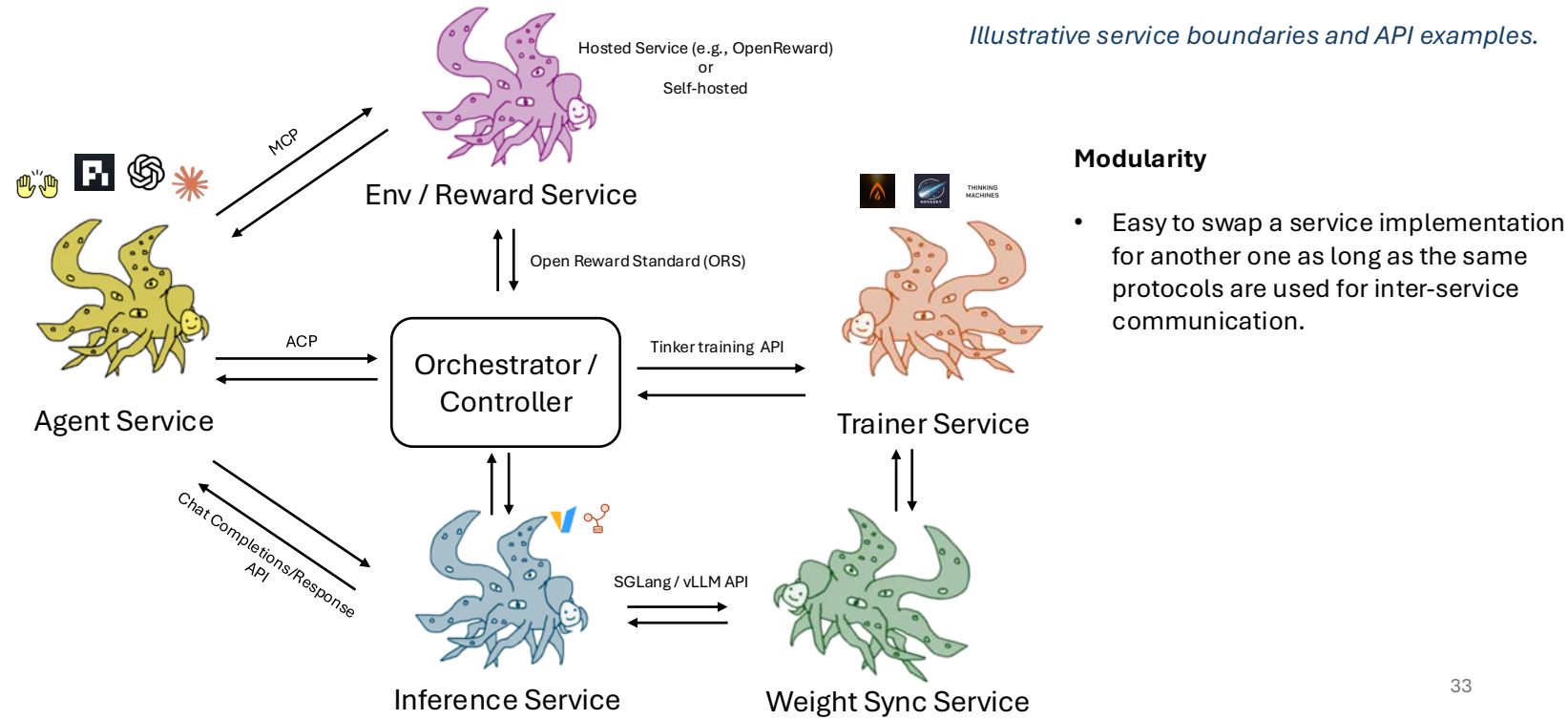
一切皆服务，使用单一控制器

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Everything-as-a-service w/ Single Controller



33

说明性服务边界与 API 示例。

环境 / 奖励服务可使用托管服务（例如 OpenReward），也可自行托管。

模块化：只要服务间通信使用相同协议，就可以轻松将某项服务实现替换为另一种实现。

图中服务与协议：Agent Service — ACP — Orchestrator / Controller; Env / Reward Service — MCP 与 Open Reward Standard (ORS) ; Controller — Tinker training API — Trainer Service; Inference Service — SGLang / vLLM API; Weight Sync Service。

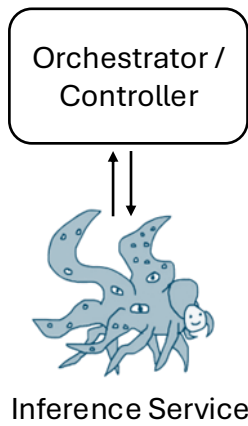
一切皆服务，使用单一控制器

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Elastic Scale Up



34

弹性扩容。

编排器 / 控制器负责管理推理服务；二者之间双向通信。

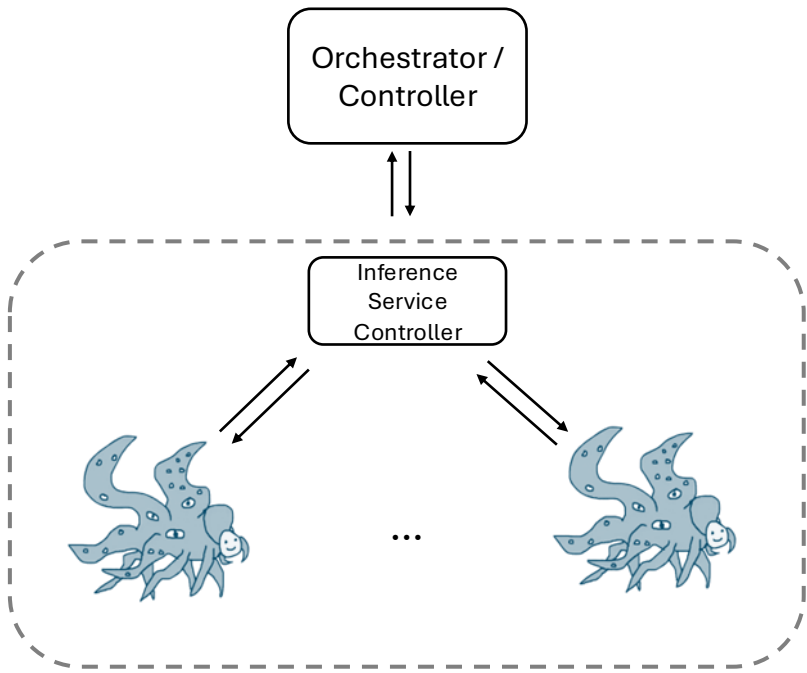
弹性扩容

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Elastic Scale Up



35

弹性扩容。

编排器 / 控制器管理推理服务控制器；推理服务控制器可以管理多个推理服务实例（图中以省略号表示）。

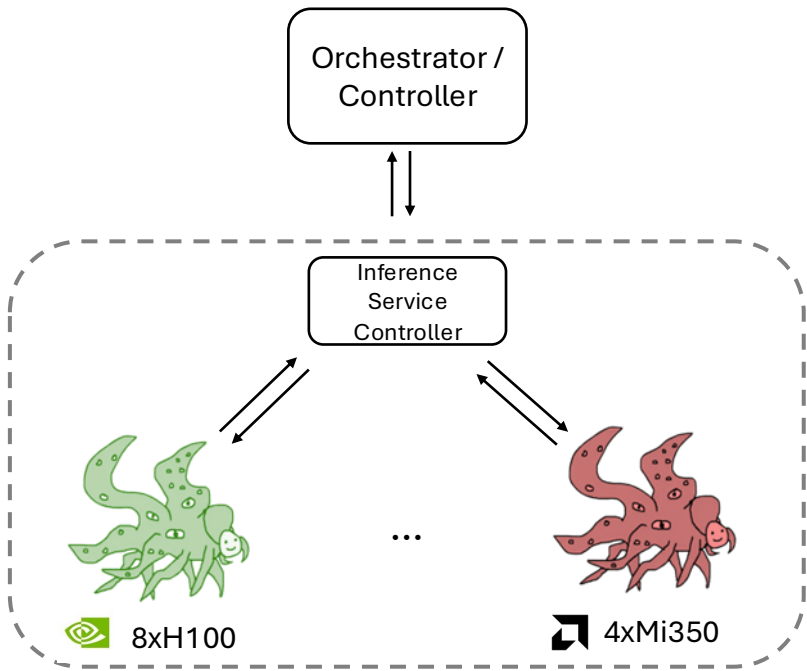
弹性扩容

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Multi-cluster / Heterogeneous Compute



36

多集群 / 异构计算。

编排器 / 控制器管理推理服务控制器；推理服务可运行于不同集群或不同类型的计算设备。

图示硬件：8×H100；4×Mi350。

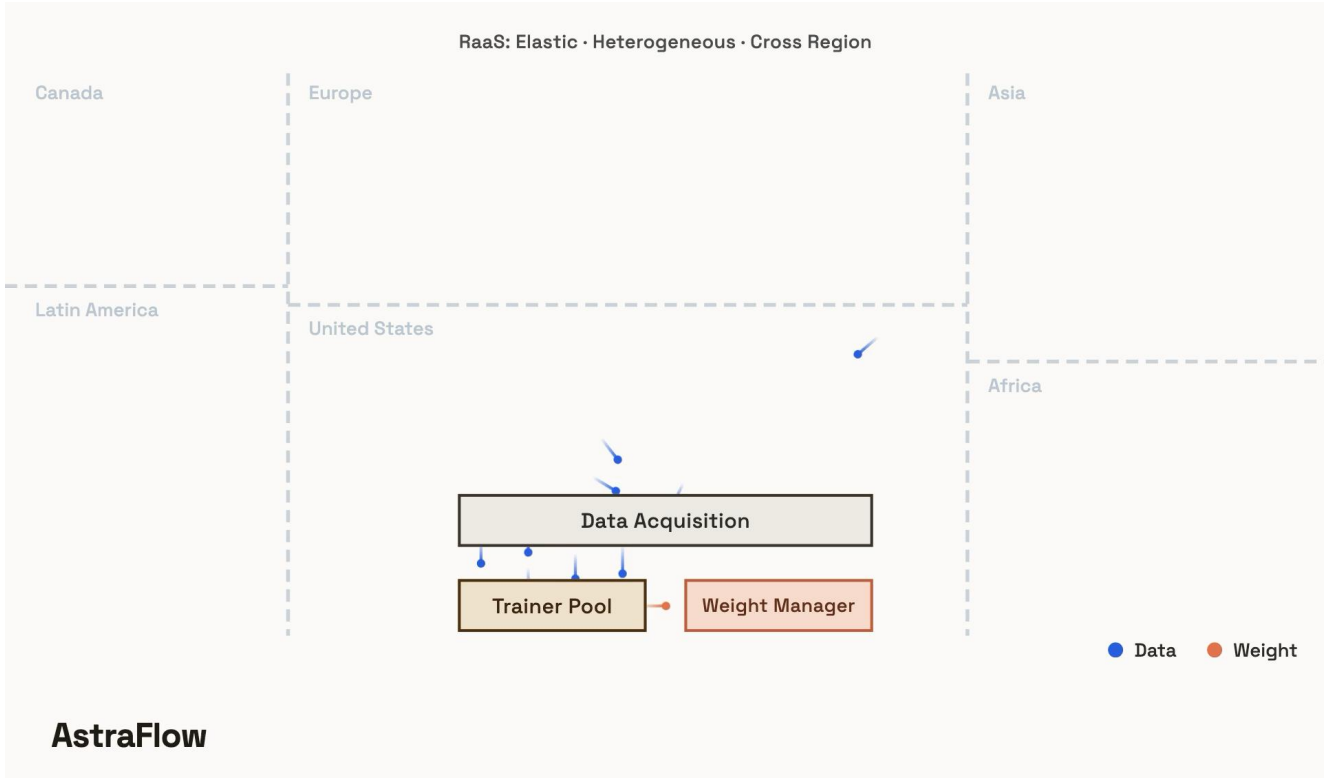
多集群 / 异构计算

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Example: AstraFlow (Haizhong Zheng et. al, 2026)



37

示例：AstraFlow (Haizhong Zheng 等, 2026) 。

RaaS：弹性 · 异构 · 跨区域。区域包括加拿大、欧洲、亚洲、拉丁美洲、美国和非洲。

数据采集 → Trainer Pool（训练器池）与 Weight Manager（权重管理器）；图例：蓝色表示数据，橙色表示权重。

AstraFlow

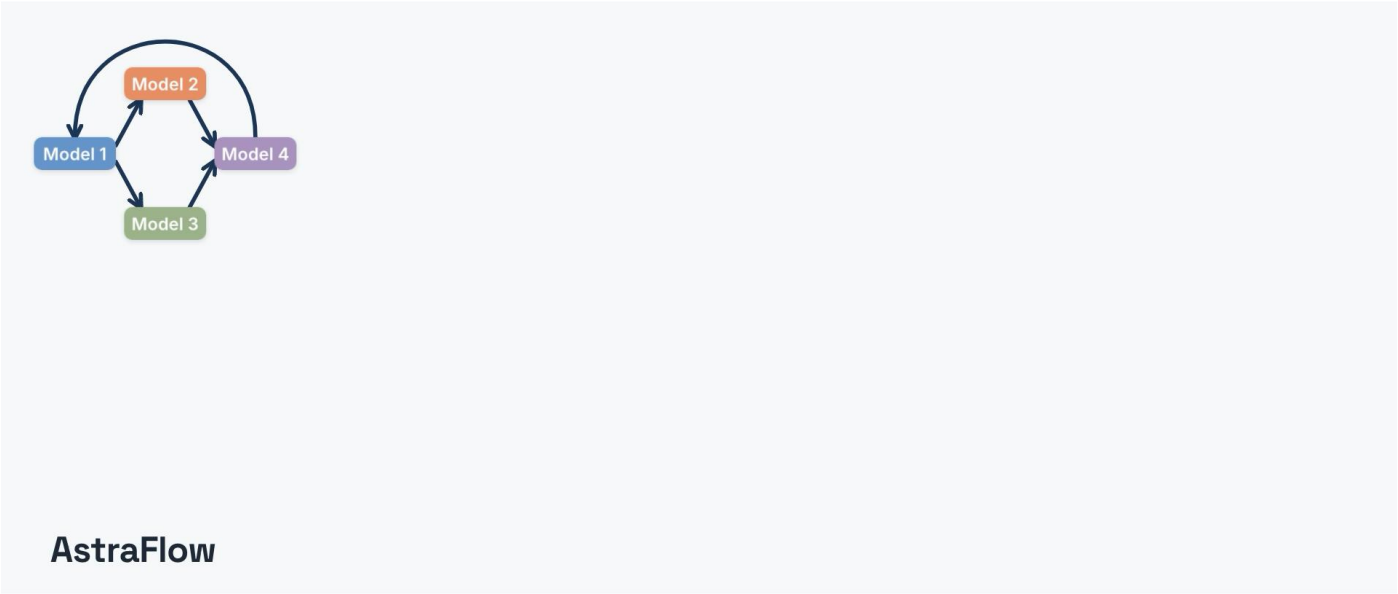
译注 / 与原文翻译分开

图中各区域名与组件名按可辨文字转译。

示例：AstraFlow

# Multi-policy Training

Microservice architecture makes it easy to train multiple *different* models simulataneously (e.g., for multi-agent rollouts with heterogenous policies)



38

多策略训练。

微服务架构让同时训练多个不同模型变得容易（例如，为使用异构策略的多智能体采样轨迹进行训练）。

图示模型：Model 1、Model 2、Model 3、Model 4；模型之间以箭头表示交互。

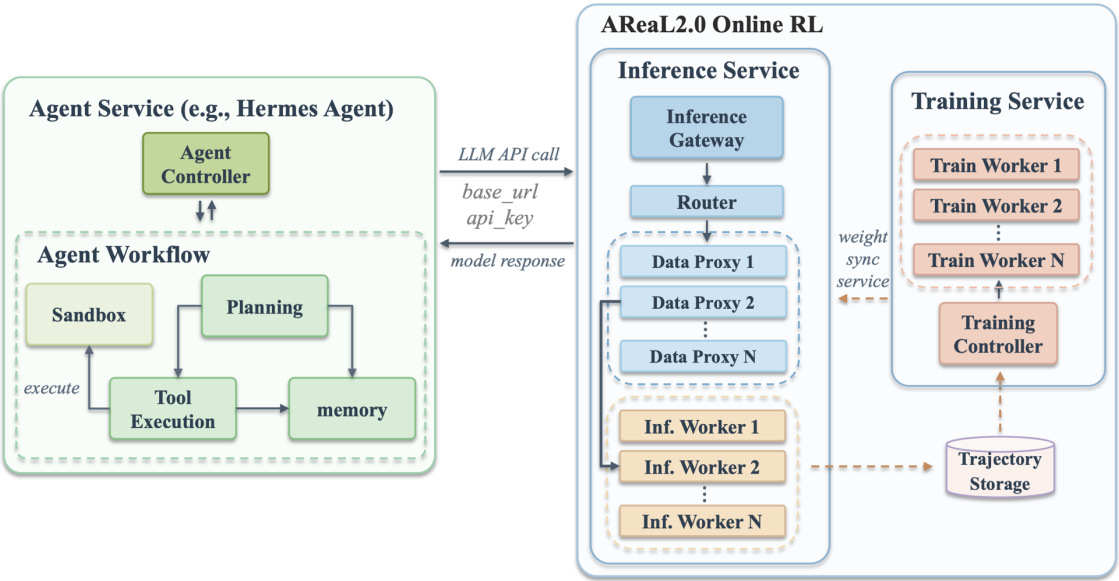
多策略训练

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分

Online RL (Training after Deployment)



Online RL in production at Cursor:

- [Cursor Tab](#)
- [Composer](#) (new model checkpoint from real user interaction every 5 hours)

AReal 2.0, 2026

39

在线强化学习（部署后训练）。

图示：Agent Service（例如 Hermes Agent）包含 Agent Controller；Agent Workflow 包含 Sandbox、Planning、Tool Execution 和 memory。智能体通过 LLM API call (base\_url、api\_key) 连接 AReal 2.0 Online RL，接收 model response。

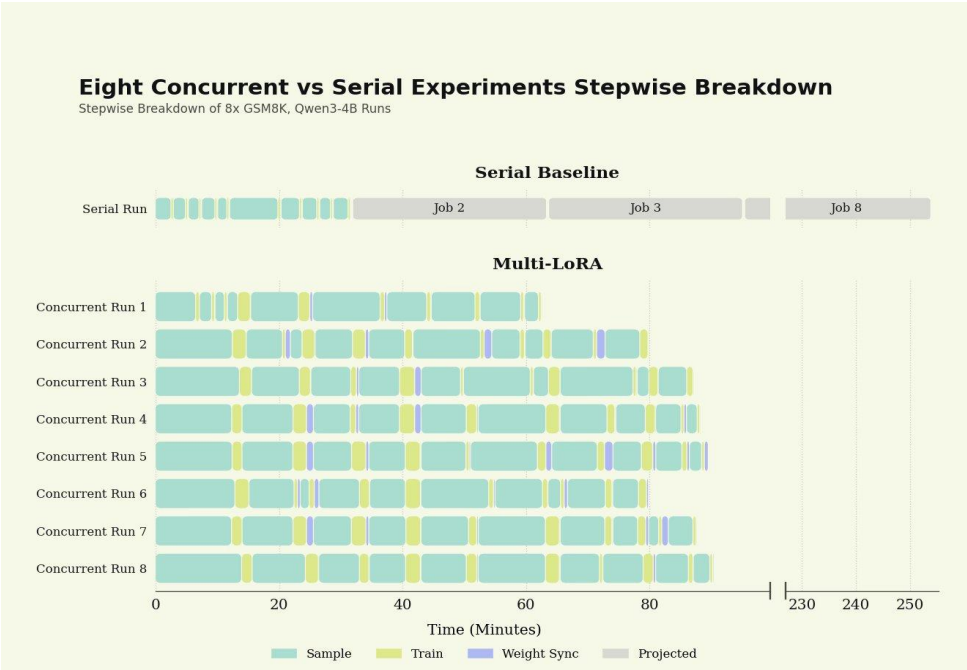
AReal 2.0 Online RL 包含 Inference Service（Inference Gateway、Router、Data Proxy 1/2/N、Inf. Worker 1/2/N）、Training Service（Train Worker 1/2/N、Training Controller）和 Trajectory Storage；推理服务与训练服务之间有 weight sync service。

Cursor 的生产环境在线 RL：Cursor Tab；Composer（每5小时根据真实用户交互产生一个新的模型检查点）。

AReal 2.0, 2026

在线强化学习（部署后训练）

# Multi-Tenant, Multi-LoRA Training



- 8 concurrent runs beat 8 serial runs in throughput (but with e2e time per run).
- Each concurrent LoRA run can be a different user of your training service.
- Requires special kernels Grouped-GEMM/SGMV kernels.

多租户、多 LoRA 训练。

图表：8次并行实验与串行实验的逐步分解；每种方式运行8次 GSM8K、Qwen3-4B。横轴为时间（分钟），串行基线图显示 Job 2、Job 3、Job 8；多 LoRA 图显示并发运行1–8。图例：Sample、Train、Weight Sync、Projected。

8个并行运行的吞吐量高于8个串行运行（但每个运行的端到端时间更长）。  
每个并发 LoRA 运行都可以对应训练服务的不同用户。

需要专用内核：Grouped-GEMM / SGMV kernels。

SkyRL + Trajectory Labs：Multi-LoRA Training for Continual Learning, 2026；Punica：Multi-tenant LoRA Serving, 2023

译注 / 与原文翻译分开

图中各阶段条形的具体时长未标注数字，未作推断。

SkyRL + Trajectory Labs: Multi-LoRA Training for Continual Learning, 2026; Punica: Multi-tenant LoRA Serving, 2023

多租户、多 LoRA 训练

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



Tinker

- Pioneered multi-tenant, multi-LoRA RL training and the decoupled training service design.
- Clean API design:
  - Training client per training run.
  - Sampling client per LoRA weight checkpoint.

|                                                                        |                                                   |                                                             |                                        |
|------------------------------------------------------------------------|---------------------------------------------------|-------------------------------------------------------------|----------------------------------------|
|                                                                        |                                                   |                                                             |                                        |
| forward_backward                                                       | optim_step                                        | sample                                                      | save_state                             |
| Perform a forward pass and a backward pass, accumulating the gradient. | Update weights based on the accumulated gradient. | Generate tokens for interaction, evaluation, or RL actions. | Save training progress for resumption. |

率先实现多租户、多 LoRA 强化学习训练和解耦式训练服务设计。

简洁的 API 设计：每次训练运行使用一个 training client；每个 LoRA 权重检查点使用一个 sampling client。

API 操作：forward\_backward——执行前向和反向计算，并累积梯度；optim\_step——根据累积梯度更新权重；sample——为交互、评测或 RL 动作生成词元；save\_state——保存训练进度，以便恢复。

译注 / 与原文翻译分开

API 方法名保留原样。

Tinker

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

ZH / 中文译文 · 第 1/1 部分



Thank you, Questions?

Next Class: Sandboxing and Credential Management

谢谢，有问题吗？

谢谢，有问题吗？

下节课：沙箱与凭据管理

第 12 讲 · 强化学习系统

EN / 英文官方原页 · 保持原样

# Some Useful Metrics to Track/Debug

| Metric                                | What it tells you                                                     | Warning sign                                                              |
|---------------------------------------|-----------------------------------------------------------------------|---------------------------------------------------------------------------|
| Reward / pass@1                       | Is the policy improving on the training tasks?                        | Flat for many steps, or rising while held-out evals fall (reward hacking) |
| pass@k                                | Coverage: can any of k samples solve it?                              | Falls while pass@1 rises: diversity is collapsing                         |
| Entropy                               | How much the policy still explores                                    | Sudden collapse toward 0, or a jump (degenerate text)                     |
| importance_weight_max                 | Worst token ratio, e.g. $\pi_{\text{train}}$ / $\pi_{\text{rollout}}$ | Spikes: mismatch or stale data; a few tokens dominate the update          |
| Grad norm                             | Size of each update before clipping                                   | Spikes ahead of divergence; steady upward drift                           |
| Average staleness                     | How many weight versions old the data is                              | Creeping up: rollouts can't keep pace with training                       |
| Trainer MFU                           | How well the trainer GPUs are used                                    | Drops from padding, imbalance or small microbatches                       |
| Timing: rollout, update, stall, total | Where the wall-clock time of a step goes                              | Stall > 0: the trainer is waiting on rollouts                             |

## 值得追踪 / 调试的一些指标

ZH / 中文译文 · 第 1/1 部分

表头：指标 | 说明什么 | 警告信号。

Reward / pass@1：策略在训练任务上是否有进步？警告：许多步骤内指标持平，或该指标上升而留出评测下降（奖励投机）。

pass@k：覆盖率——k 个样本中是否有任何一个能解决任务？警告：pass@1 上升时 pass@k 下降，说明多样性正在崩塌。

Entropy：策略还在探索多少？警告：突然降至0附近，或突然升高（文本退化）。

importance\_weight\_max：最差词元比率，例如  $\pi_{\text{train}}$  /  $\pi_{\text{rollout}}$ 。警告：尖峰表示数据不匹配或陈旧；少数词元主导了更新。

Grad norm：裁剪前每次更新的大小。警告：在发散前出现尖峰，或持续向上漂移。

Average staleness：数据落后当前权重版本多少代。警告：数值不断升高，表示采样轨迹跟不上训练。

Trainer MFU：训练 GPU 的利用效率。警告：填充、负载不平衡或微批次过小会让指标下降。

Timing: rollout, update, stall, total：每个步骤的墙钟时间分布。警告：stall > 0 表示训练端正在等待采样轨迹。

译注 / 与原文翻译分开

指标名称、数学符号和变量保留原样。