ROLL

like a Reinforcement Learning
Algorithm Developer

ROLL: Reinforcement Learning Optimization for Large-scale Learning

——面向Agentic场景的大规模强化学习训练框架

ROLL

like a Reinforcement Learning Algorithm Developer

目录

Part 1：ROLL诞生篇

Part 2：ROLL核心设计篇

Part 3：ROLL实践篇

ROLL

like a Reinforcement Learning Algorithm Developer

Part 1：ROLL诞生篇

LLM与RL：痛点与ROLL的诞生

- 我们设计框架时，该考虑什么问题?
  - 早期的强化学习框架多是为了满足特定研究需求而设计，随着应用场景的不断扩展，框架通过不断叠加功能来满足新需求，这种渐进式演变导致架构变得臃肿，维护成本越来越高。
  - 不同用户群体对框架的期望存在显著差异，框架设计需要在这些诉求之间找到平衡点：通过合理的抽象层次和模块化设计，让不同用户都能高效地使用框架，在保持核心简洁的同时，提供足够的扩展性支持多样化需求。

- ***Rich Training Recipes***
- ***Superior Performance***
- ***Easy Device-Reward Mapping***
- ***…***

Product Developer

- ***Constrained Device Execution***
- ***Pluggable Reasoning Pipeline***
- ***…***

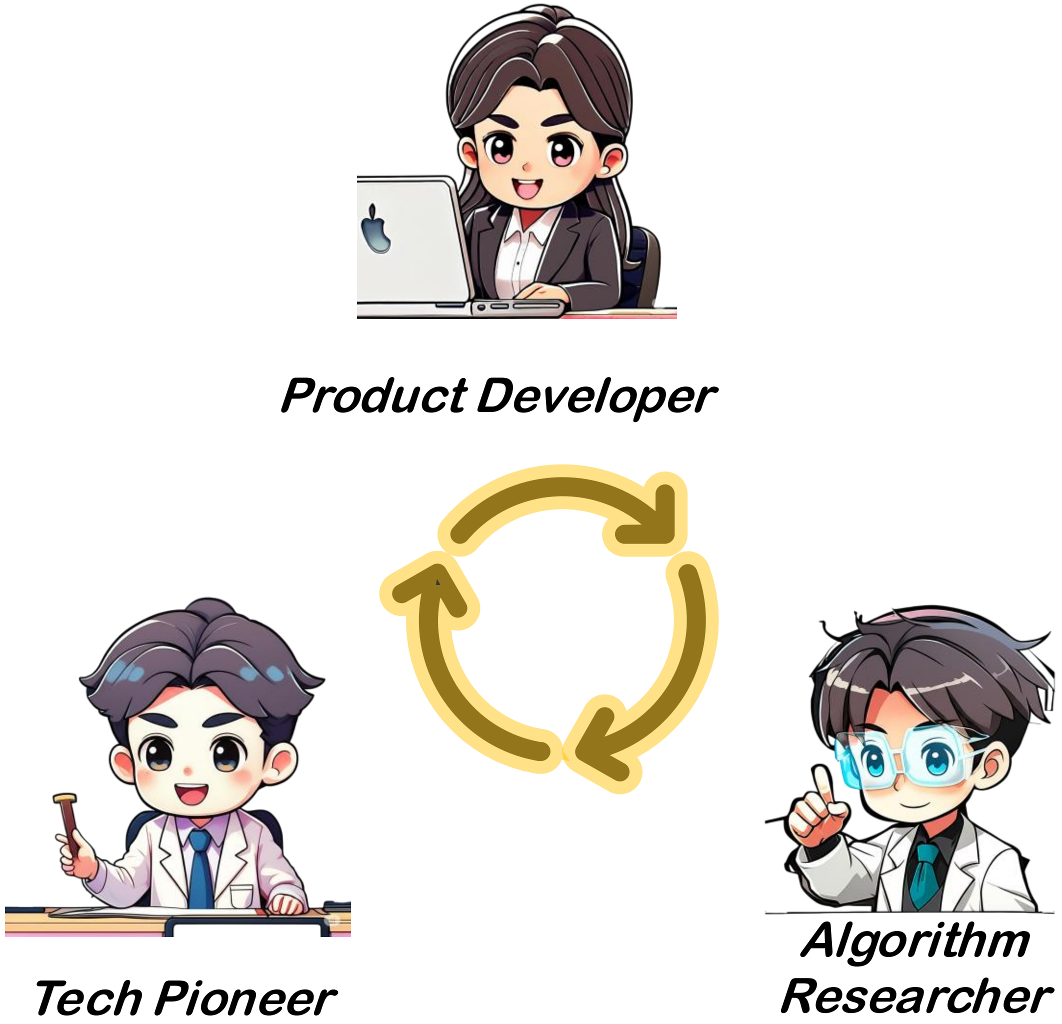
Algorithm Researcher

- ***Fast and Effective***
- ***Scalability and Fault Tolerance***
- ***…***

Tech Pioneer

LLM与RL：痛点与ROLL的诞生

- 同时满足这三类用户是一个“不可能三角”吗？
  - 用户角色往往是动态转换的：
    - 业务算法工程师可能需要进行创新研究来解决特定问题；
    - 学术研究员在推进前沿研究的同时可能要考虑工程落地；
    - 而工业界研究员则经常在基础研究和实际应用间切换。
  - 这种角色的流动性要求框架能够无缝支持不同场景下的需求转换。



LLM与RL：痛点与ROLL的诞生

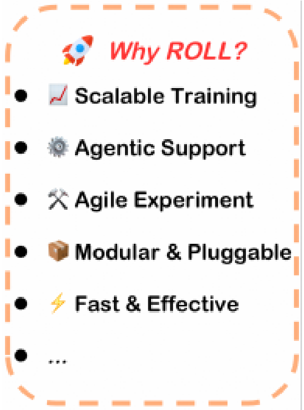
- 在设计框架时，我们的理念是将所有类型的用户都视为“一等公民”，而不是过分偏向某一群体。这意味着框架需要在不同层次上同时满足多种需求：
  - 为业务工程师提供清晰的抽象和便捷的部署能力，
  - 为学术研究员保留足够的灵活性和可定制性
  - 为工业界研究员在研究创新和规模化应用之间搭建桥梁。

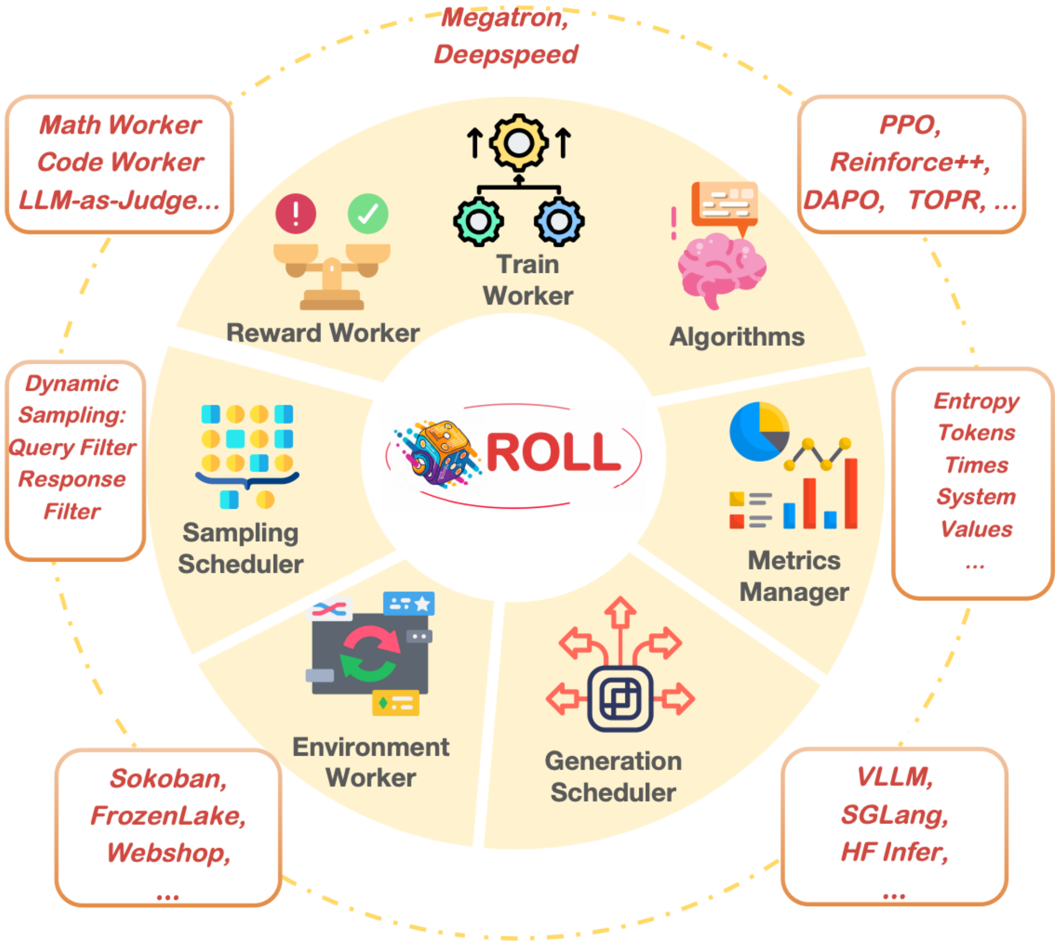
- 通过精心的接口设计和架构规划，我们致力于在易用性和效率之间找到平衡点，让用户能够根据实际需求无障碍地在不同使用模式间切换，真正实现框架对所有用户的普遍友好。

ROLL框架概览：关键特性

➢ 核心定位: 高效、用户友好的强化学习库，专为LLM大规模优化设计

➢ 关键特性:
- **多任务强化学习**：内置丰富的 RL 任务支持，涵盖数学、代码、通用推理、开放式问答、指令遵循等，一套训练循环即可多领域联合优化，采样率与数据权重可灵活动态调整
- **智能体强化学习（Agentic RL）**：原生支持多环境、多角色智能体 - 环境交互（游戏、多轮对话等），并具有灵活的并行化和内置管理功能，异步采样和异步训练
- **算法友好**：提供灵活且丰富的 RL 策略配置，开箱即用地支持 PPO、GRPO、Reinforce++ 等算法



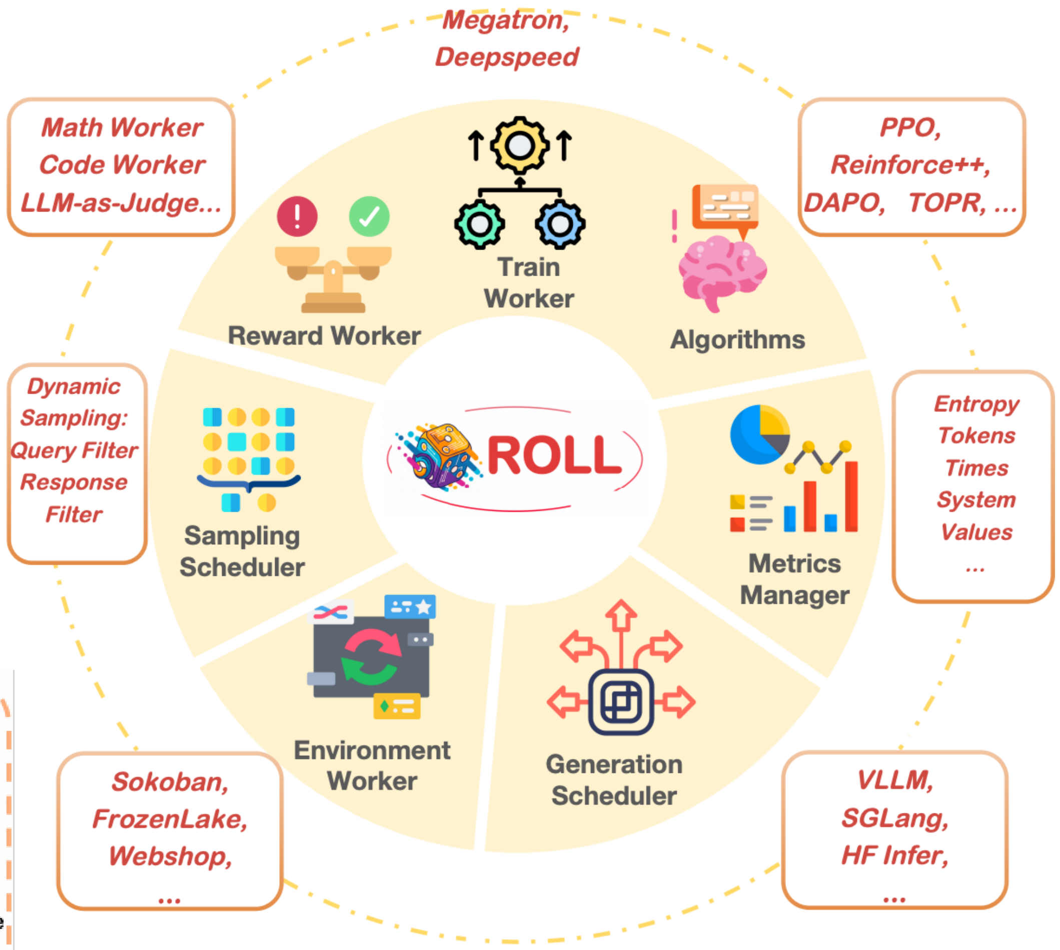


ROLL框架概览：关键特性

- 核心定位: 高效、用户友好的强化学习库，专为LLM大规模优化设计
- 关键特性:
  - **丰富的训推引擎**：基于 Ray 的多角色分布式架构，灵活支持 vLLM、SGLang、Megatron-Core、DeepSpeed 等主流推理 / 训练引擎，从单机到千卡集群均能轻松运行
  - **极致易用与模块化扩展**：Rollout Scheduler、AutoDeviceMapping 等关键模块极大简化 pipeline 开发和调试，支持按需组合套件，后端推理 / 训练引擎自由切换。
  - **样本级调度与动态采样**：样本级 Rollout 生命周期调度机制，支持异步奖励计算、动态采样、按样本裁剪与 EarlyStopping，显著提升训练效率与资源利用率。
  - **可观察性**：集成了swanlab / wandb / tensorboard，支持实时跟踪每个领域、每个策略、每个奖励的性能 —— 从高层概况到细粒度分析

🚀 *Why ROLL?*

- 📈 Scalable Training
- ⚙️ Agentic Support
- ⚒️ Agile Experiment
- 📦 Modular & Pluggable
- ⚡ Fast & Effective
- …



训练效果展示：RLVR (RL with Verifiable Reward)

- ➢ RLVR是什么？ 优化LLM在有可验证答案任务上的性能（数学、代码、通用推理）
- ➢ 实验设置:
  - 数据集：DeepMath、KodCode等，多领域混合训练
  - 模型：Qwen2.5-7B-Base, Qwen3-30B-A3B-Base
  - 算法：PPO Loss + REINFORCE，多验证机制（规则、沙箱、LLM-as-Judge）
- ➢ 核心结果 (图3a, 4a):
  - Qwen2.5-7B-Base: 整体准确率从 0.18 提升至 0.52。数学：0.20->0.53；代码：0.13->0.41
  - Qwen3-30B-A3B-Base: 整体准确率从 0.27 提升至 0.62
- ➢ 稳定性与鲁棒性: 多任务混合训练中模型持续提升，未出现崩溃
- ➢ 多模态支持: Qwen2.5-VL-7B-Instruct在GEOQA_R1V_Train_8K上表现良好 (图4c)。

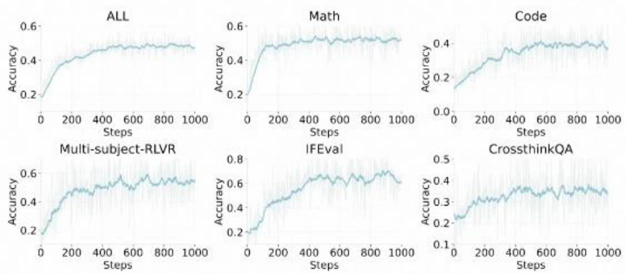


Figure 3: Accuracy Trends Across Different Tasks on Qwen2.5-7B-Base.

Figure3 a. Dense: Qwen2.5-7B-Base

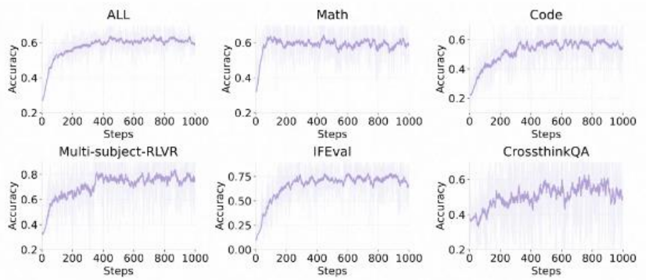


Figure 4: Accuracy Trends Across Different Tasks on Qwen3-30B-A3B-Base.

Figure 4 a. MOE: Qwen3-30B-A3B-Base

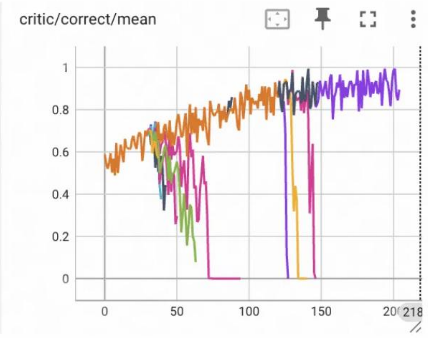


Figure 3b.MOE 200+B
Crash and resume

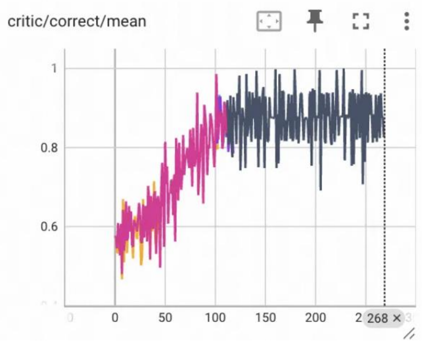


Figure 4b.MOE 200+B
Stable training

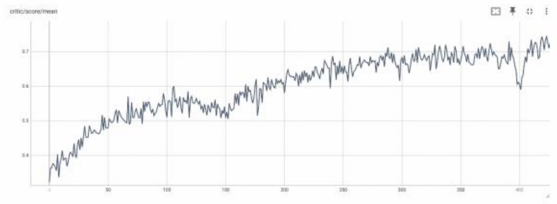
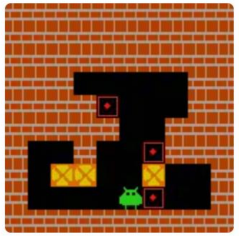


Figure 4 c Qwen2.5-VL-7B-Instruct 训练score曲线

训练效果展示：Agentic RL

- Agentic RL是什么？ 训练LLM作为智能体，在多轮交互环境中完成复杂任务。
- 实验环境:
  - Sokoban (推箱子): 成功率从16.8%提升至26.0% (训练)，有效动作比例提升至73.4%
  - FrozenLake (冰湖): 成功率从16.8%提升至26.0% (训练)，有效动作比例提升至88.8%
  - WebShop (在线购物): 复杂自然语言交互，任务成功率从37%大幅提升至85%+，平均操作数从7次降至4次
- 核心优势:
  - 显著提升任务成功率与执行效率
  - 展现出良好的泛化能力和跨环境迁移能力
  - 可扩展的Agent-Env**异步并行**多轮交互采样，更高效
  - **异步训练**：rollout/训练解耦，扩展性高



Sokoban

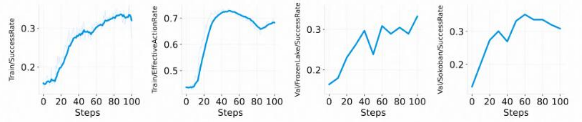
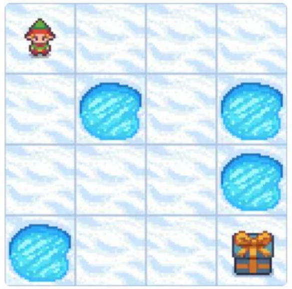


Figure 5: Performance metrics for the SimpleSokoban environment training. *SuccessRate* denotes the success rate of reaching the goal. *EffectiveActionRate* represents the proportion of valid actions executed.



FrozenLake

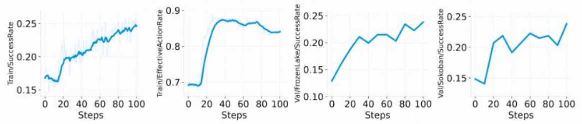
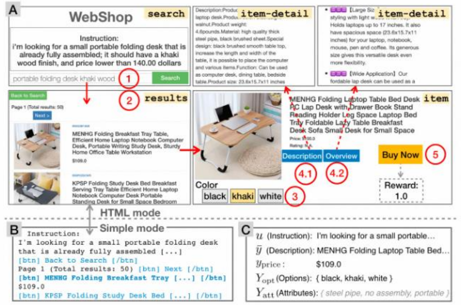


Figure 6: Performance metrics for the FrozenLake environment training.



WebShop

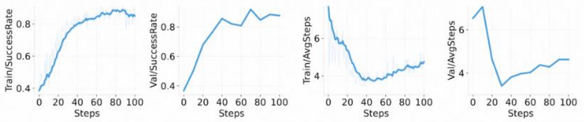


Figure 7: Performance metrics for the WebShop environment training. *AvgSteps* indicates the average number of steps required to complete the task, where fewer steps imply higher action efficiency.

Part 2：ROLL核心设计篇

算法抽象：算法视角的模块化与灵活性

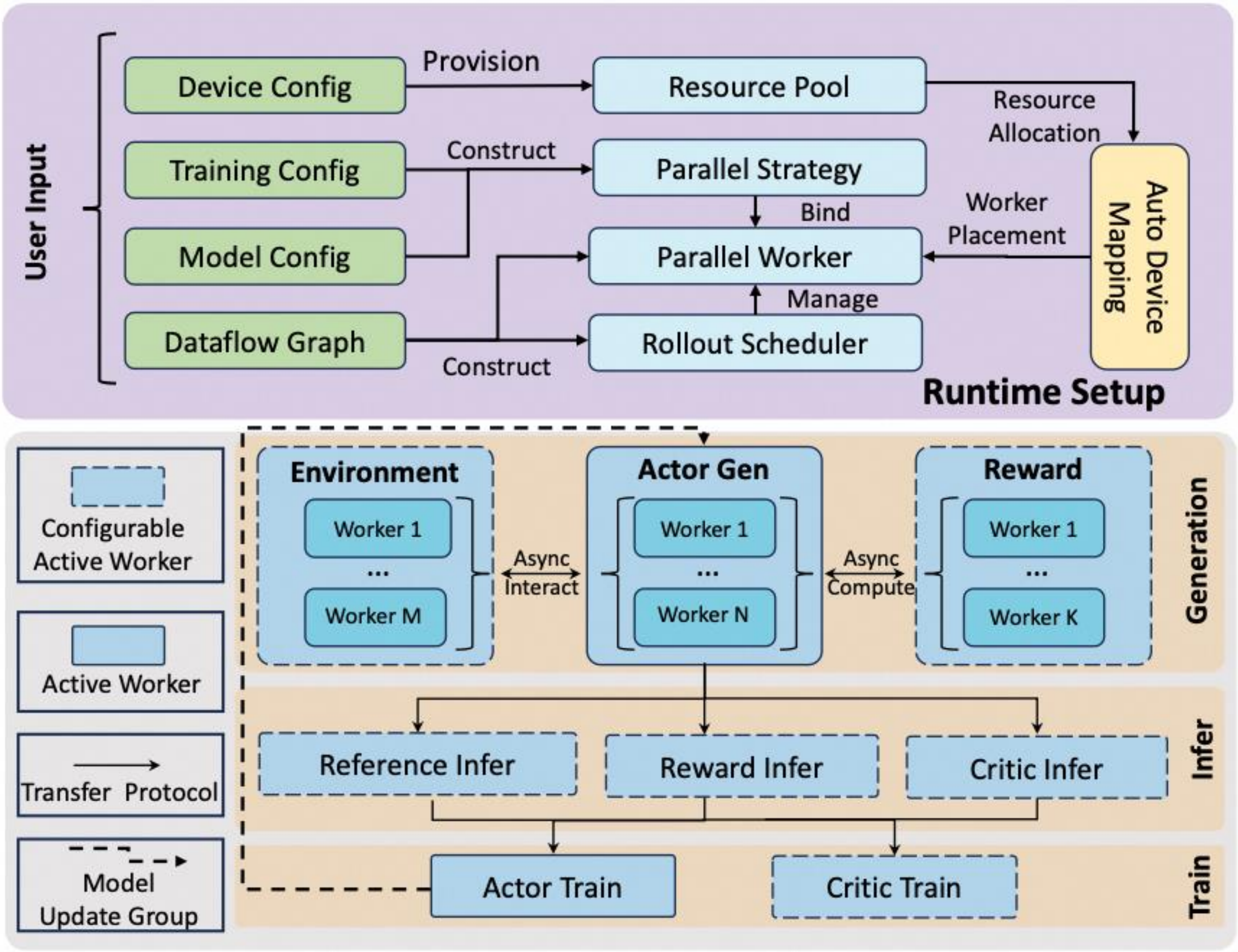
Workflow Overview

- 运行时设置 (Runtime Setup)
  - 根据用户配置，自动分配资源，创建各分布式组件
- 训练迭代 (Training Iteration):
  - 数据生成 (Generation)

    并行交互： Actor、Env或Reward worker**异步并行**协作，高效生成经验数据并计算奖励、过滤
  - 推理 (Infer)

    对数据的前向推理，构造完整训练样本
  - 训练 (Train)

    模型更新： Actor和Critic更新参数，将参数同步回生成模型，实现策略的持续迭代与优化

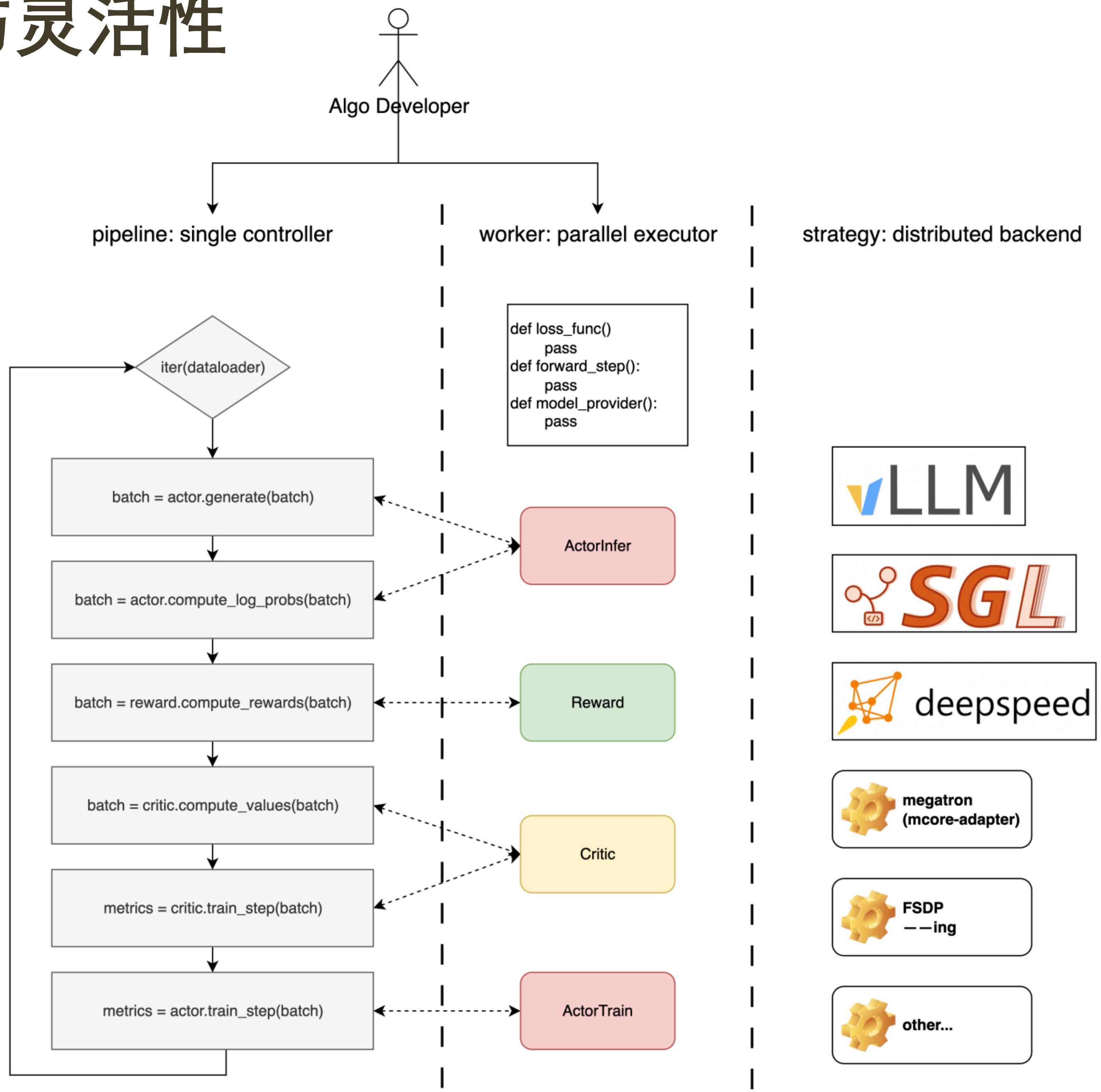


(b) Workflow

算法抽象：算法视角的模块化与灵活性

解耦算法逻辑与分布式工程实现

- Single Controller Pipeline:
  - 单进程视角编排各角色的计算流程
  - 简化调度，提高开发效率
- Worker抽象: 可插拔的模块化组件
  - Actor Worker: 模型生成与策略训练
  - Critic Worker (可选): 状态价值估计
  - Reward Worker: 多种奖励计算（规则验证、代码沙箱、LLM as Judge)
  - Environment Worker:管理与环境的多轮交互

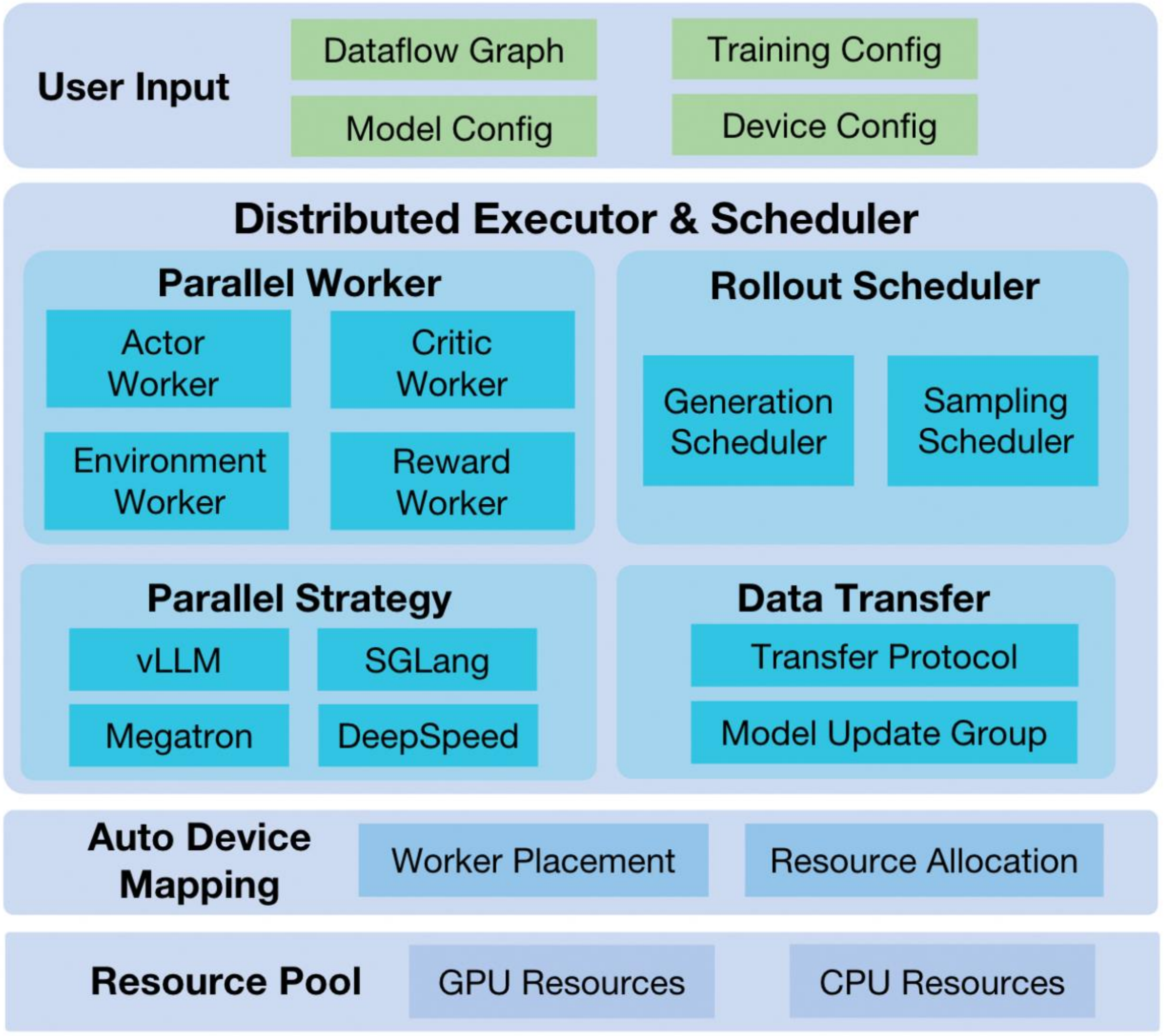


框架架构：面向分布式与高性能的LLM执行体系

模块化与可扩展的分布式架构

核心理念：构架面向LLM的分布式计算基石

- **高度模块化与可扩展性**： ROLL在算法抽象之上，构建了一套灵活高效的分布式执行架构，无缝集成多种先进LLM推理与训练引擎
- **广泛适用场景**： 从单机部署到大规模GPU集群，ROLL均能提供卓越的性能支持，确保了其在多样化应用场景中的适应性与扩展性
- **引擎无缝切换**： 支持训练(DeepSpeed、Megatron、FSDP[ing])、推理(vLLM、SGLang)的无缝切换，并扩展了高效的GPU offload/reload实现，充分发挥并行计算优势



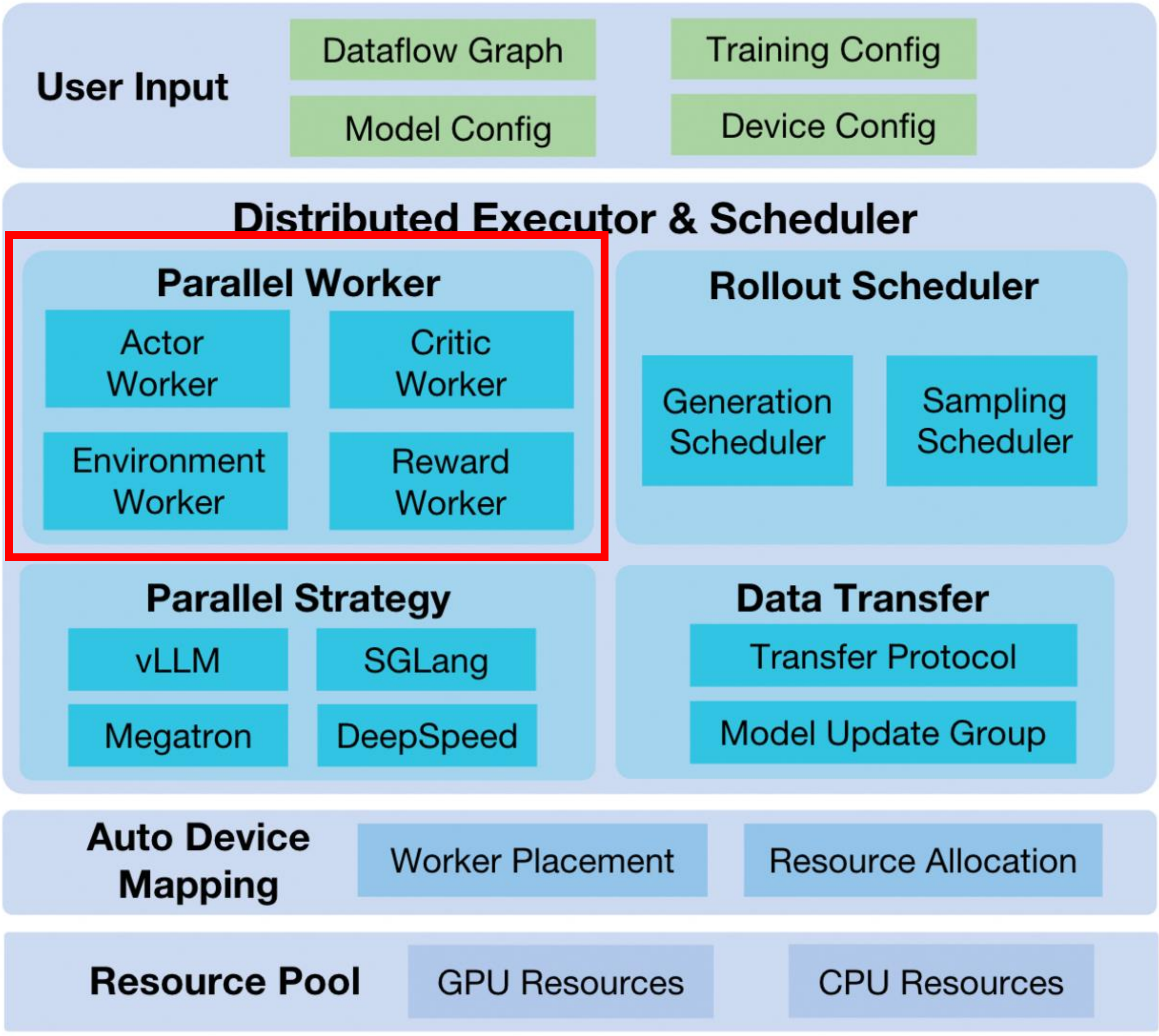
(a) Architecture

框架架构：面向分布式与高性能的LLM执行体系

模块化与可扩展的分布式架构

关键执行单元：Parallel Worker

- ➢ 资源持有单元
  - ROLL中的基本资源管理单位，每个Parallel Worker持有一组Ray PlacementGroup资源
  - 用户在Worker内自定义业务逻辑，单进程视角编程、分布式执行
- ➢ Cluster协同管理
  - 通过Cluster层对各角色（如Actor、Critic、Environment、Reward）的Workers进行统一管理
  - Pipeline里描述各角色Cluster协同计算过程，Worker执行具体分布式计算



(a) Architecture

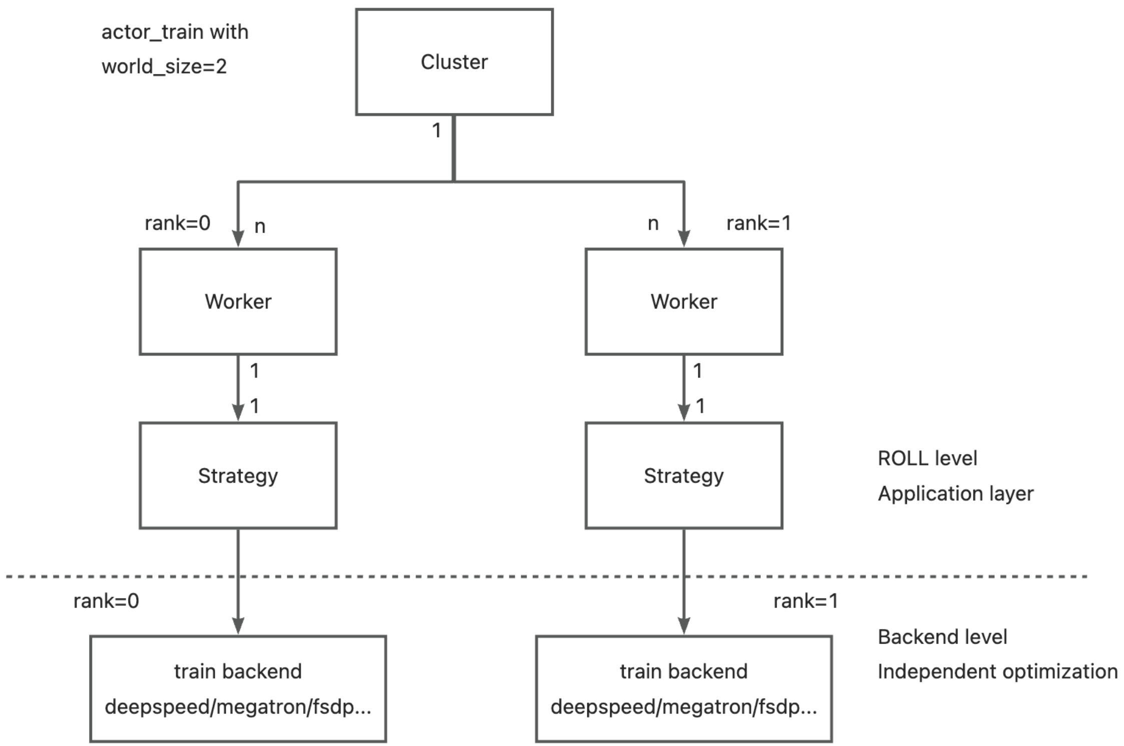
框架架构：面向分布式与高性能的LLM执行体系

模块化与可扩展的分布式架构

核心引擎集成：Parallel Strategy

- 训练: 无缝集成MegatronCore和DeepSpeed，支持5D并行（DP, PP, TP, CP, EP），结合ZeRO/Offload降低显存
- 推理: 整合vLLM和SGLang，支持TP/EP/PP

```
class TrainStrategy(InferenceStrategy):  xiongshaopan.xsp +1 *
    def __init__(self, worker: "Worker"):...

    def setup_collective_group(self, model_update_name, comm_plan, backend="nccl"):...

    def train_step(  xiongshaopan.xsp +1
        self,
        batch: DataProto,
        loss_func: Callable[[DataProto, torch.Tensor], Tuple[torch.Tensor, Dict[str, torch.Tensor]]],
    ):...

    def model_update(self, *args, **kwargs):...

    def save_checkpoint(self, *args, **kwargs):...

    def load_checkpoint(self, *args, **kwargs):  5 usages (1 dynamic)  new *
        pass
```



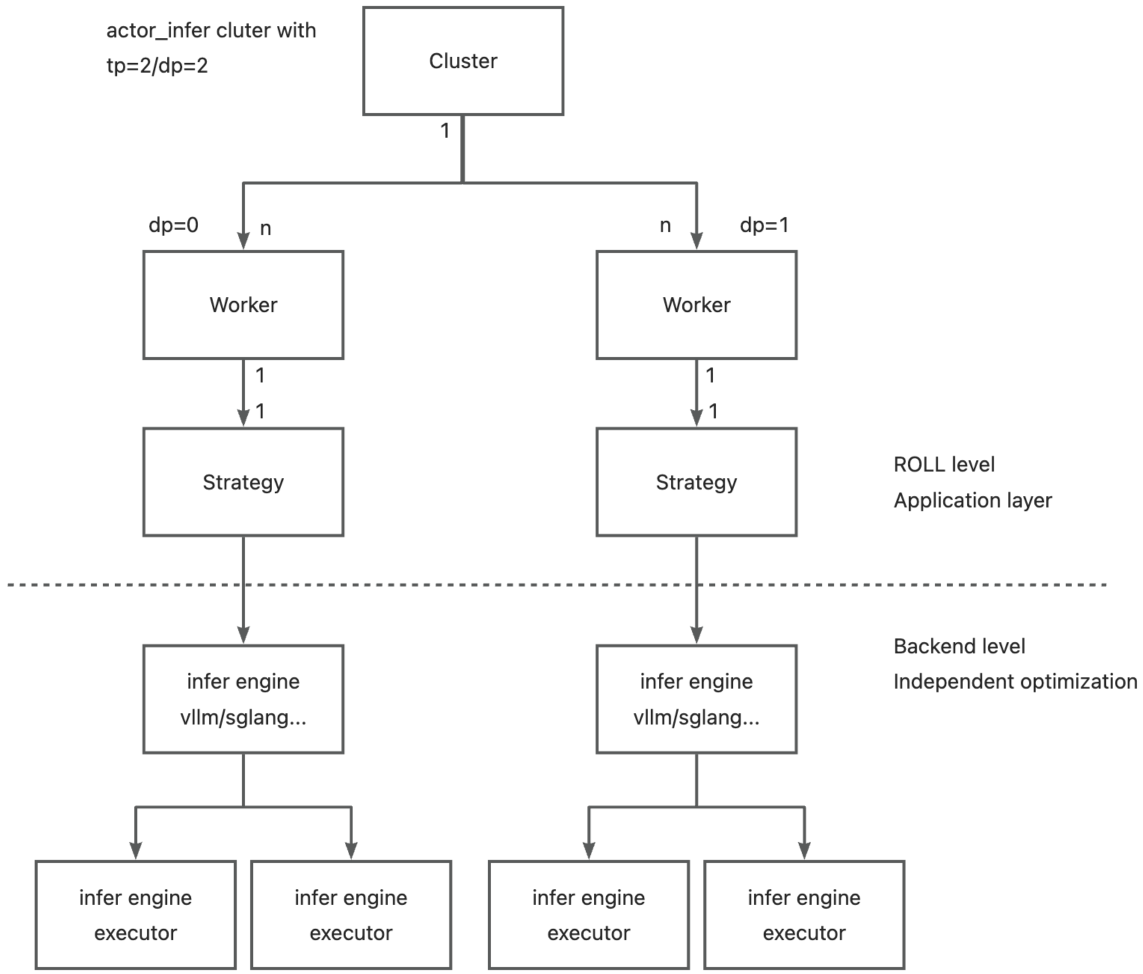
框架架构：面向分布式与高性能的LLM执行体系

模块化与可扩展的分布式架构

核心引擎集成：Parallel Strategy

- 训练: 无缝集成MegatronCore和DeepSpeed，支持5D并行（DP, PP, TP, CP, EP），结合ZeRO/Offload降低显存
- 推理: 整合vLLM和SGLang，支持TP/EP/PP

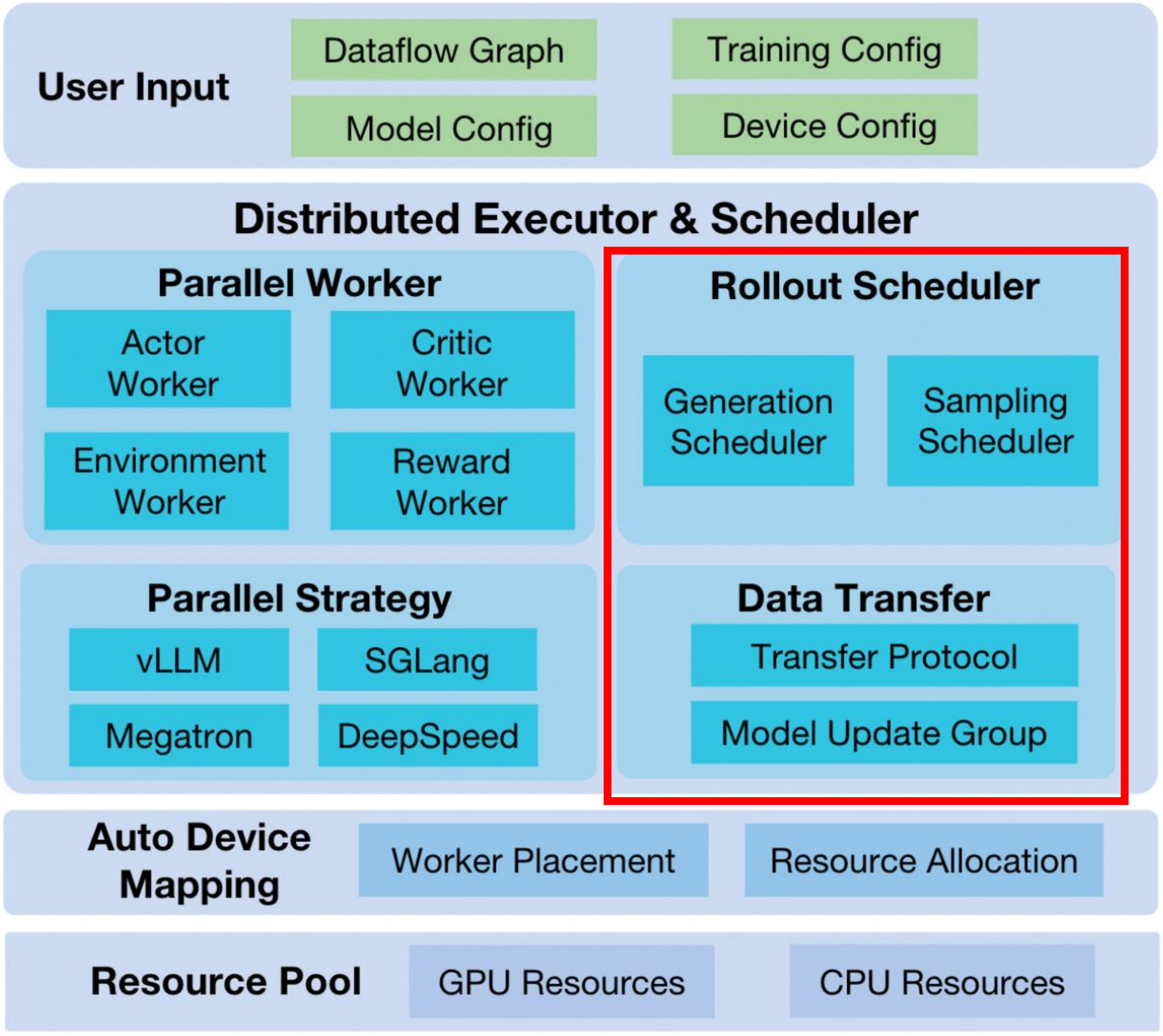
```python
class InferenceStrategy(ABC):
    strategy_name = None

    def __init__(self, worker: "Worker"):...

    def initialize(self, *args, **kwargs):
        raise NotImplementedError
    def forward_step(
        self,
        batch: DataProto,
        forward_func: Callable[[DataProto, torch.Tensor], Tuple[torch.Tensor, Dict[str, torch.Tensor]]],
    ) -> Dict[str, torch.Tensor]:...
    def generate(self, *args, **kwargs):...
    def add_request(self, command, data: DataProto, *args, **kwargs):...

    # 参数同步相关接口
    def broadcast_bucket(self, model_update_name, src_pp_rank, meta_infos, bucket_size):...
    def setup_collective_group(self, model_update_name, comm_plan, backend="nccl"):...

    # offload/load 相关接口
    def load_states(self):...
    def offload_states(self, *args, **kwargs):
        raise NotImplementedError
```



框架架构：面向分布式与高性能的LLM执行体系

模块化与可扩展的分布式架构

- **Rollout Scheduler:** 样本级异步并行rollout，动态负载均衡、异步执行、灵活任务路由
- **Data Transfer:** ModelUpdateGroup高效参数同步



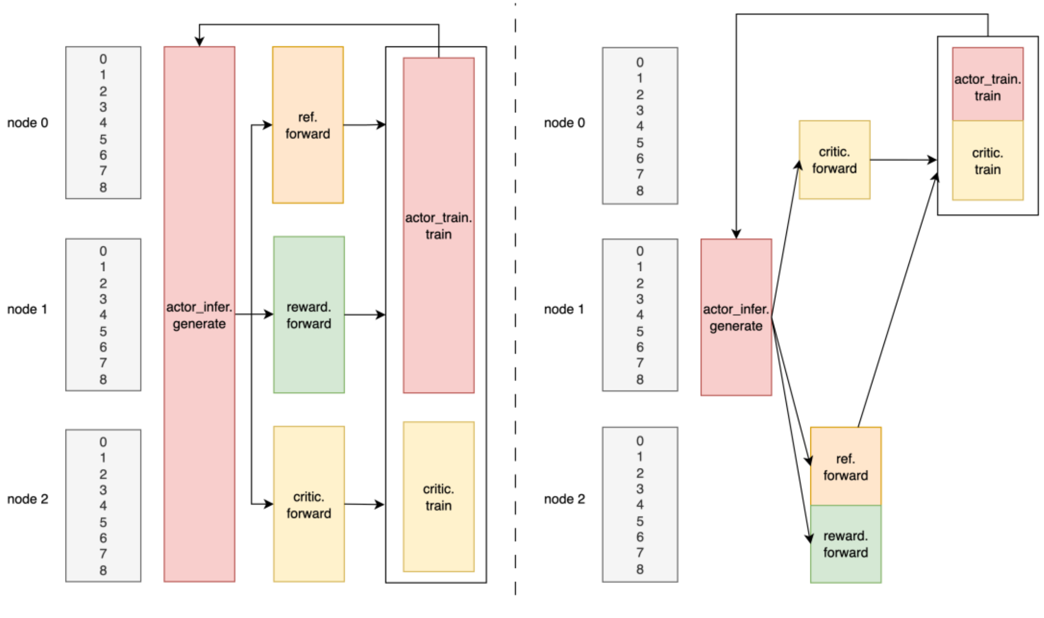
(a) Architecture

框架架构：面向分布式与高性能的LLM执行体系

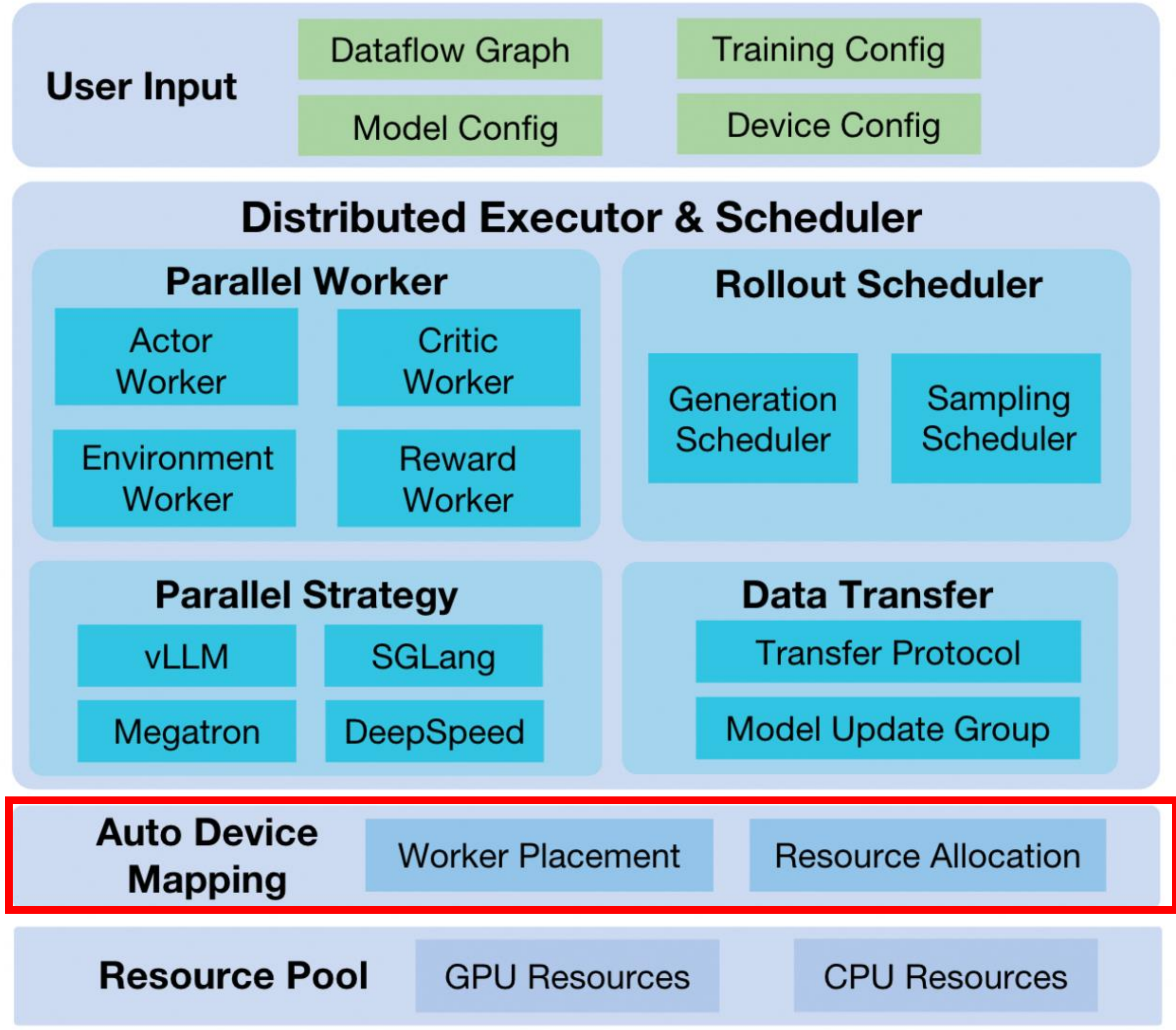
模块化与可扩展的分布式架构

灵活资源管理：AutoDeviceMapping

- 灵活的资源管理，支持用户自定义设备映射，支持共置(colocated)和分离部署(disaggregated)



colocated device_mapping　　disaggregated device_mapping

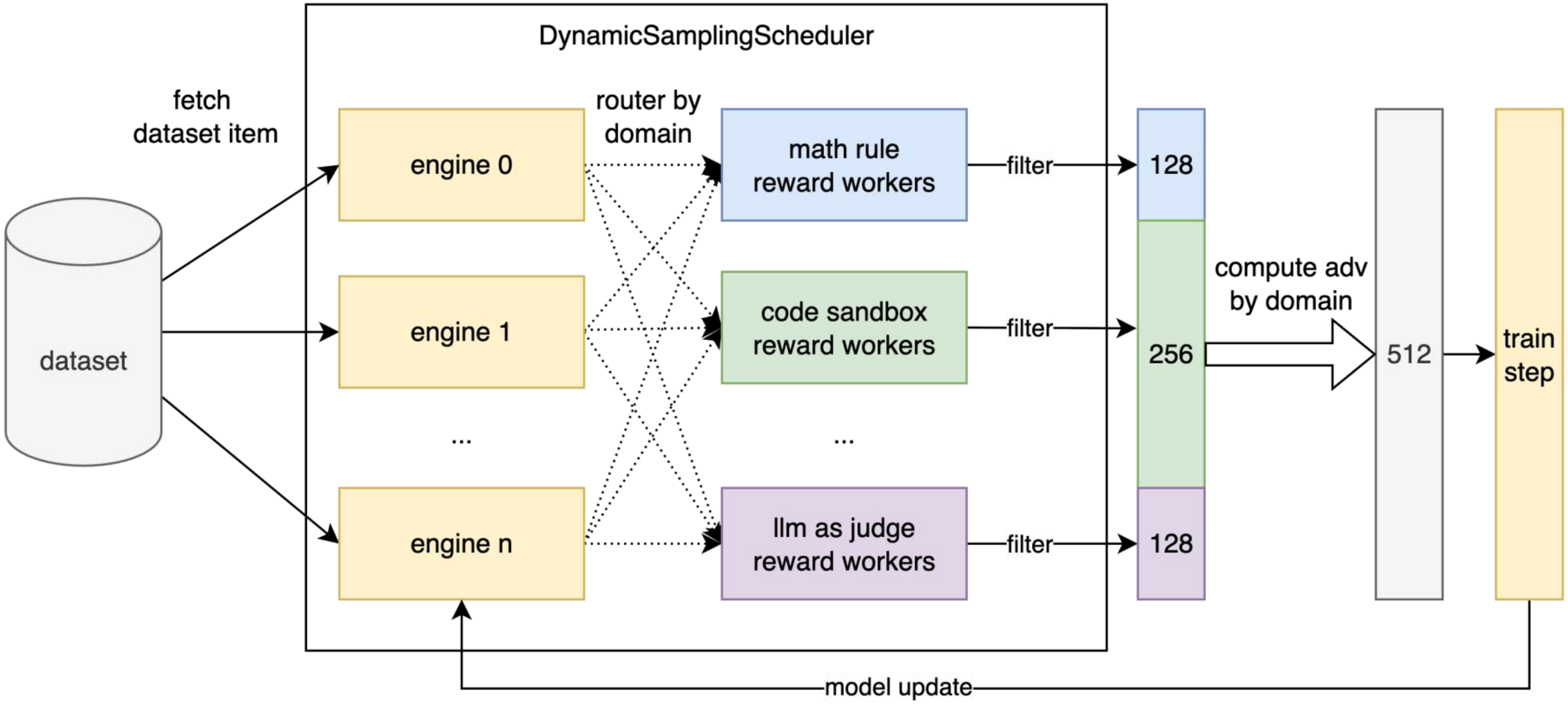


(a) Architecture

RLVR 多任务训练与异步奖励计算

多任务联合训练

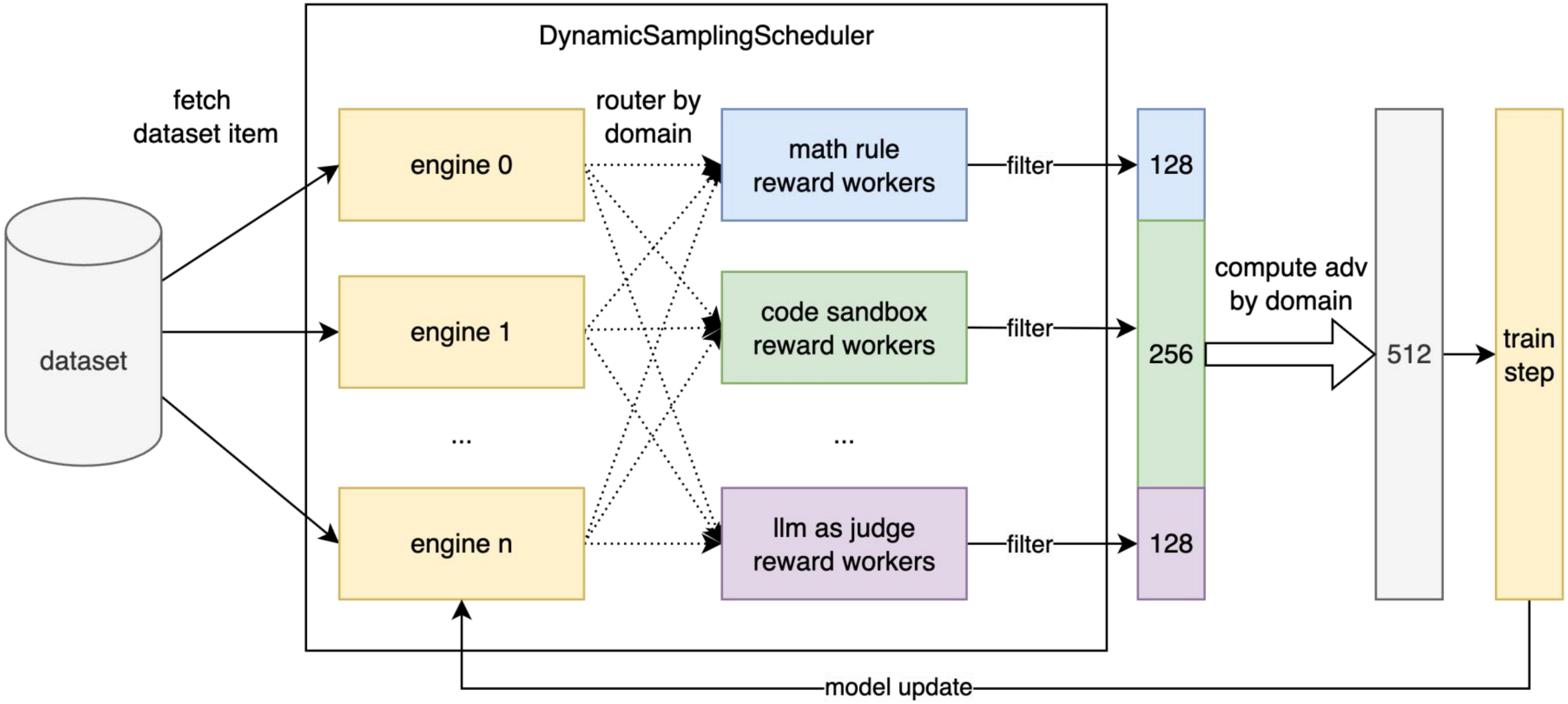
- ROLL内置丰富的RL任务支持，涵盖数学、代码、通用推理、开放式问答、指令遵循等
- 一个训练循环即可多领域联合优化，采样率与数据权重可灵活动态调整



RLVR 多任务训练与异步奖励计算

多任务联合训练

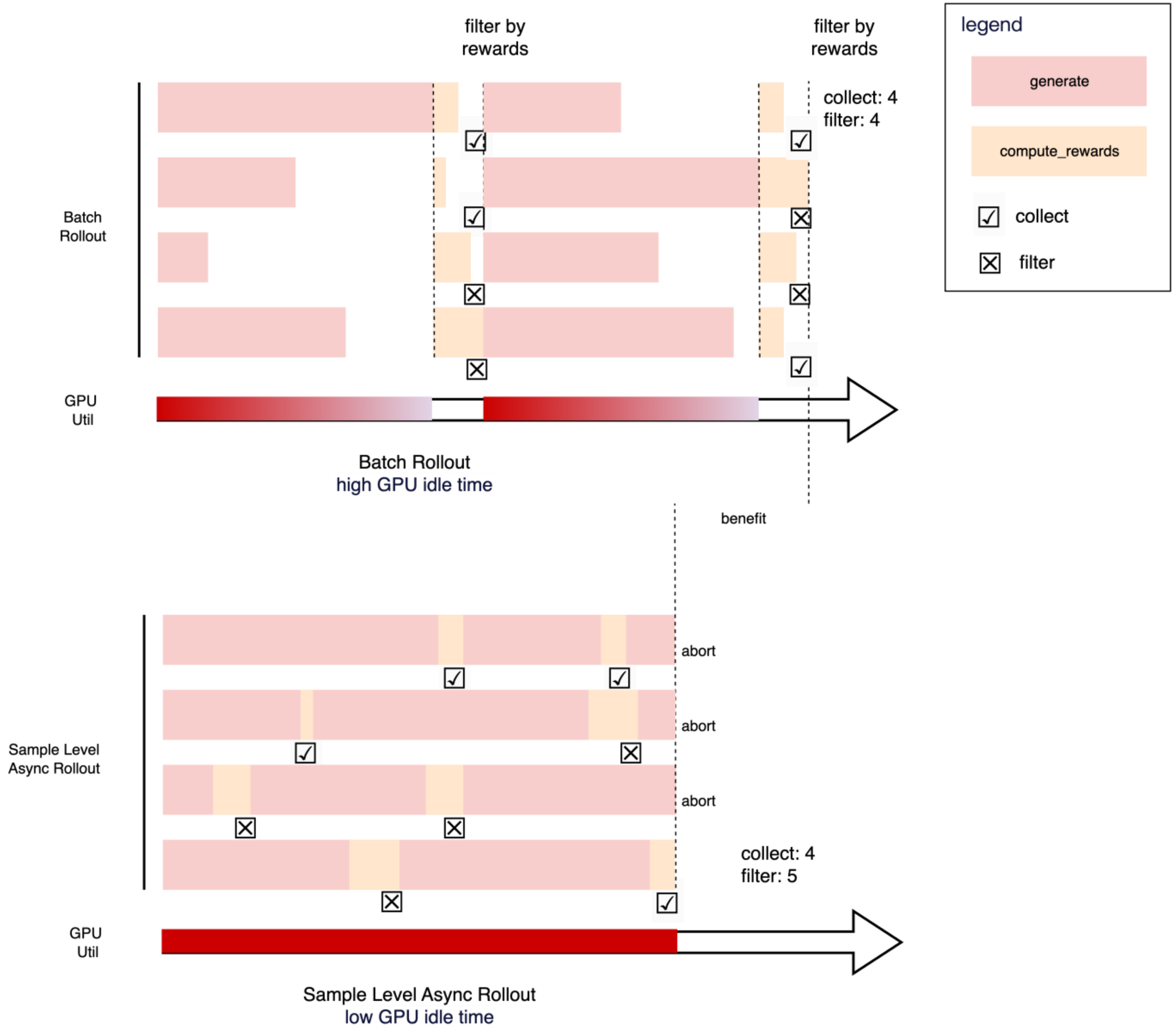
1. 数据抽取 (Dataset): 从多领域数据集中fetch prompt
2. response生成：actor_infer engine并行生成response
3. Reward计算与过滤：根据Prompt所属domain，路由到Reward Worker，并动态过滤
4. Adv计算：按domain计算adv
5. 模型训练与更新



RLVR 多任务训练与异步奖励计算

异步奖励计算 与 动态过滤

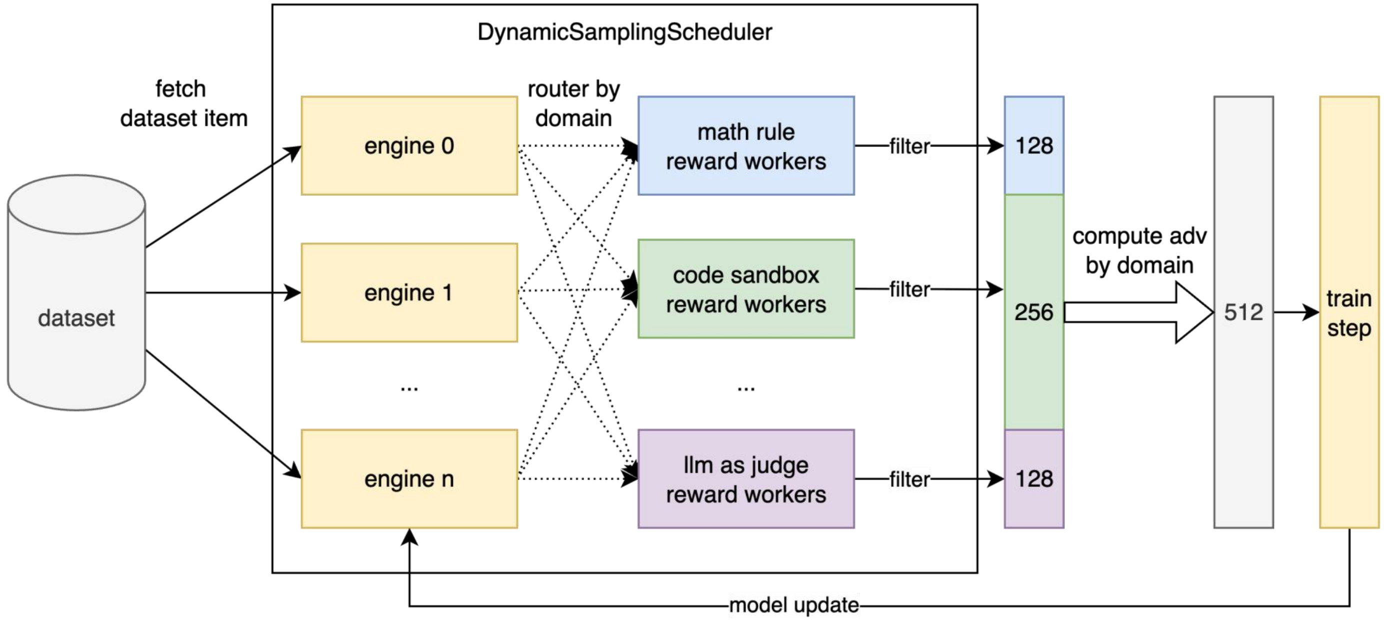
- ➢ 传统的batch rollout
  - 生成、奖励计算、过滤 串行进行
  - 大量时间等待，导致GPU资源闲置
- ➢ ROLL: 样本级async rollout，压榨GPU效率
  - DynamicSamplingScheduler管理Prompt生成，一旦生成完毕，立即调用对应reward Worker异步计算reward。(对应图中 generate 和 compute_rewards 的重叠与连续)



RLVR 多任务训练与异步奖励计算

多任务联合训练——实践

- 灵活的reward配置与路由
  - YAML配置： 通过rewards定义各领域专属Reward Worker
  - 动态实例化： worker_cls自动创建
  - 自动数据路由： 根据数据tag（或domain），路由Reward Worker计算




```yaml
rewards:
  math_rule:
    worker_cls: roll.pipeline.rlvr.rewards.math_rule_reward_worker.MathRuleRewardWorker
    tag_included: [deepmath_103k, aime]
  code_sandbox:
    worker_cls: roll.pipeline.rlvr.rewards.code_sandbox_reward_worker.CodeSandboxRewardWorker
    tag_included: [KodCode]
  llm_judge:
    worker_cls: roll.pipeline.rlvr.rewards.llm_judge_reward_worker.LLMJudgeRewardWorker
    tag_included: [RLVR]
```

RLVR 多任务训练与异步奖励计算

多任务联合训练——实践

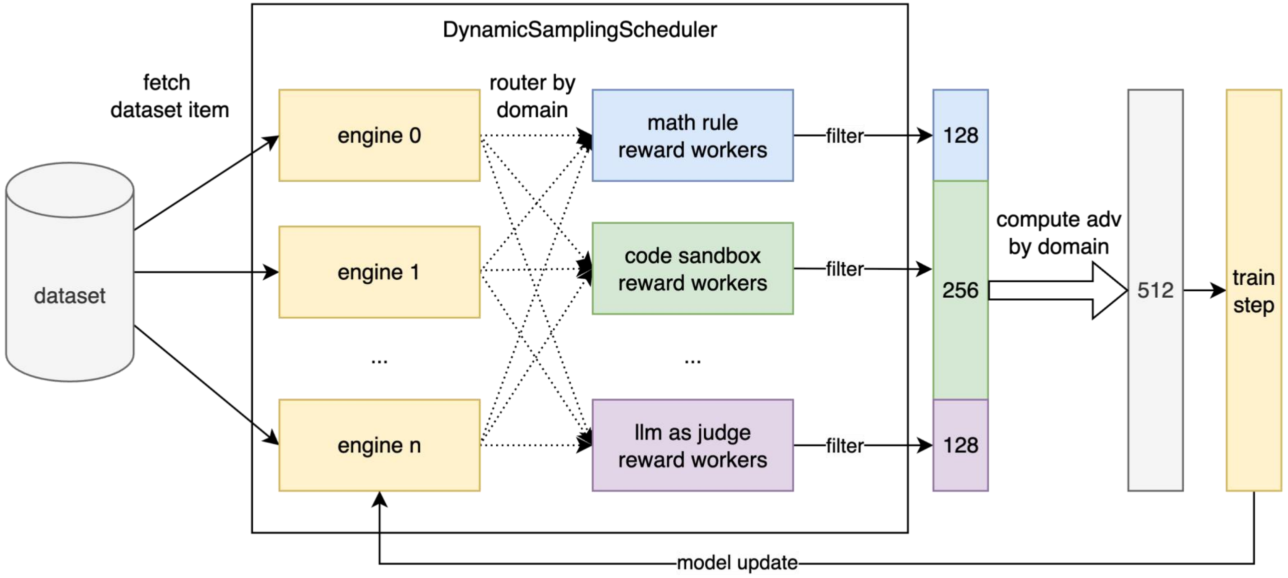
- 动态采样过滤（DynamicSamplingScheduler）
  - 域独立调度： 为每个域配备专属的DynamicSamplingScheduler
  - 样本级生成与奖励计算： scheduler均衡向actor_infer发送prompt请求，并对应domain Reward Worker计算reward
  - 动态过滤：自定义函数对response进行动态过滤

```
def query_filter_fn(data_list: List[DataProto], config: RLVRConfig) -> bool:   1 usage   xiongshaopan.xsp +1
    """
    各domain的过滤规则可以自定义
    """
    response_level_rewards = [data.batch["response_level_rewards"] for data in data_list]
    if len(response_level_rewards) == 1:
        return True
    rewards = torch.cat(response_level_rewards, dim=0)

    domain = data_list[0].non_tensor_batch["domain"][0]
    query_filter_config = config.rewards[domain].query_filter_config

    if query_filter_config.type == "no_filter":
        return True
    elif query_filter_config.type == "mean_filter":
        threshold_up = query_filter_config.filter_args.get("threshold_up", math.inf)
        threshold_down = query_filter_config.filter_args.get("threshold_down", -1)
        if torch.mean(rewards) <= threshold_down or torch.mean(rewards) >= threshold_up:
            return False
    elif query_filter_config.type == "std_filter":
        std_threshold = query_filter_config.filter_args.get("std_threshold", -1)
        if torch.std(rewards) <= std_threshold:
            return False
    return True
```

RLVR 多任务训练与异步奖励计算

多任务联合训练——实践

- 灵活的domain batch size分布控制
  - 保持全局rollout_batch_size不变
  - 基于domain_interleave_probs参数，灵活分配每个域的Batch Size比例
    - 示例：数学40%，代码30%，LLM Judge 10%...
  - [进阶玩法]根据任务训练情况动态调整，提升训练效果



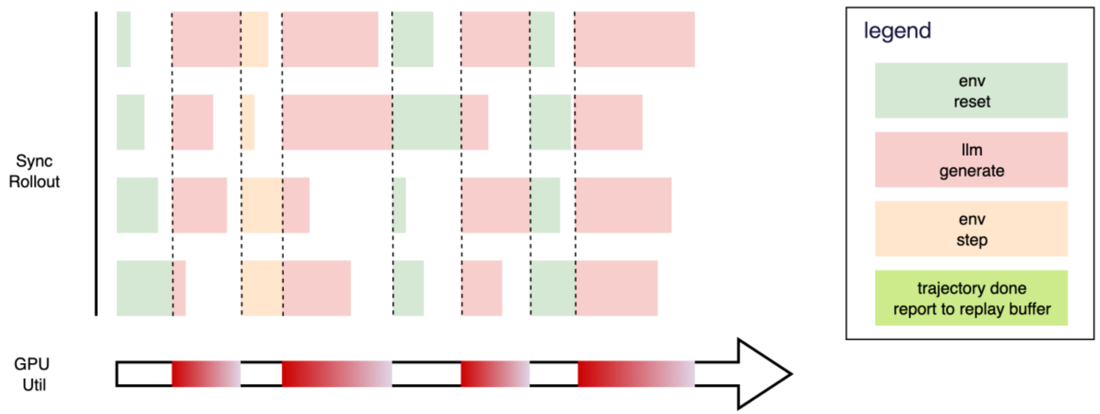

```yaml
domain_interleave_probs:
  math_rule: 0.4
  code_sandbox: 0.3
  llm_judge: 0.1
  crossthinkqa: 0.1
  ifeval: 0.1
```

```python
scheduler_refs = {}
for domain, scheduler in self.generate_schedulers.items():
    scheduler_refs[domain] = scheduler.get_batch.remote(data=batch,
                                                        batch_size=self.domain_batch_size[domain])
ray.get(list(scheduler_refs.values()))
```

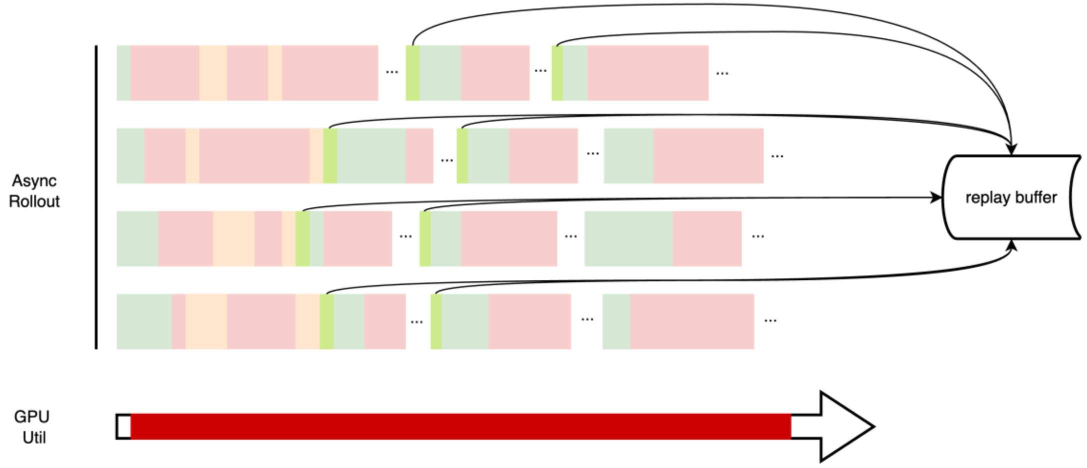
Agentic 异步并行rollout与异步训练

异步并行rollout

- Batch Rollout
  - 执行模式: 同步、批次化执行
  - 核心特点:
    - barrier: 各阶段强制等待最慢的环境完成
    - 资源利用率低: 导致大量 GPU空闲时间
    - 性能瓶颈: 整体速度受最慢环境限制
- ROLL: Async Parallel Rollout
  - 执行模式: 解耦、基于env粒度的并行执行
  - 核心特点:
    - 无barrier: 各环境独立推进，不互相等待
    - 高资源利用率: GPU持续工作
    - 整体速度更快: 减少总Rollout时间
    - 灵活性高: 适应异构环境和动态控制



Sync Rollout:
Barriers at each stage, leading to high GPU idle time



Async Rollout:
Parallel rollouts are executed per env, ensuring high GPU utilization

Agentic 异步并行rollout与异步训练

异步并行rollout

- 多轮交互EnvManager
  - 核心: 异步并行Rollout的基石
  - 职责: 管理单个环境 (BaseEnv) 的rollout
  - 特性: 自包含，支持本地调试与并行部署
- 核心功能与运行机制
  - run_rollout_loop: 执行完整Rollout（环境交互、LLM决策）直至采样结束
  - LLM决策: 通过LLMProxy生成action
  - 环境步进: 应用动作，收集环境反馈
  - 多EnvManager并行执行，发送完整轨迹至队列 (GroupQueueManager)

```python
class TrajEnvManager(BaseEnvManager):  xiongshaopan.xsp *

    def run_rollout_loop(self, data: DataProto):  4 usages (2 dynamic)  xiongshaopan.xsp *
        self.episode_id = 0
        self.group_seed = data.meta_info['seed'] + self.env_config['group_seed']
        rollout_cache: RolloutCache = self.reset()
        start_step = self.current_step

        while self.running:

            lm_output: DataProto = self.make_decision(rollout_cache)
            stop_reason = lm_output.meta_info.pop("stop_reason")

            if stop_reason == GenerateStopReason.FINISH:
                rollout_cache: RolloutCache = self.step(lm_output)

            if self.running and (rollout_cache.terminated or stop_reason == GenerateStopReason.MAX_LENGTH):

                rollout: DataProto = self.formulate_rollouts(rollout_cache)

                traj_group_id = f"{self.env_config['group_id']}_{self.episode_id}_{self.group_seed}"
                rollout.non_tensor_batch["traj_group_id"] = np.array( object: [traj_group_id], dtype=object)
                ray.get(self.output_queue.put.remote(self.env_config['group_id'], self.episode_id, start_step, rollout))

                self.rollout_cache = None
                if not self.running or self.episode_id >= self.worker_config.max_traj_per_env:
                    self.logger.debug(
                        f"env_id: {self.env_config['env_id']} max_traj_per_env {self.worker_config.max_traj_per_env} reached, stopping rollout loop")
                    break

                rollout_cache = self.reset()
```

Agentic 异步并行rollout与异步训练

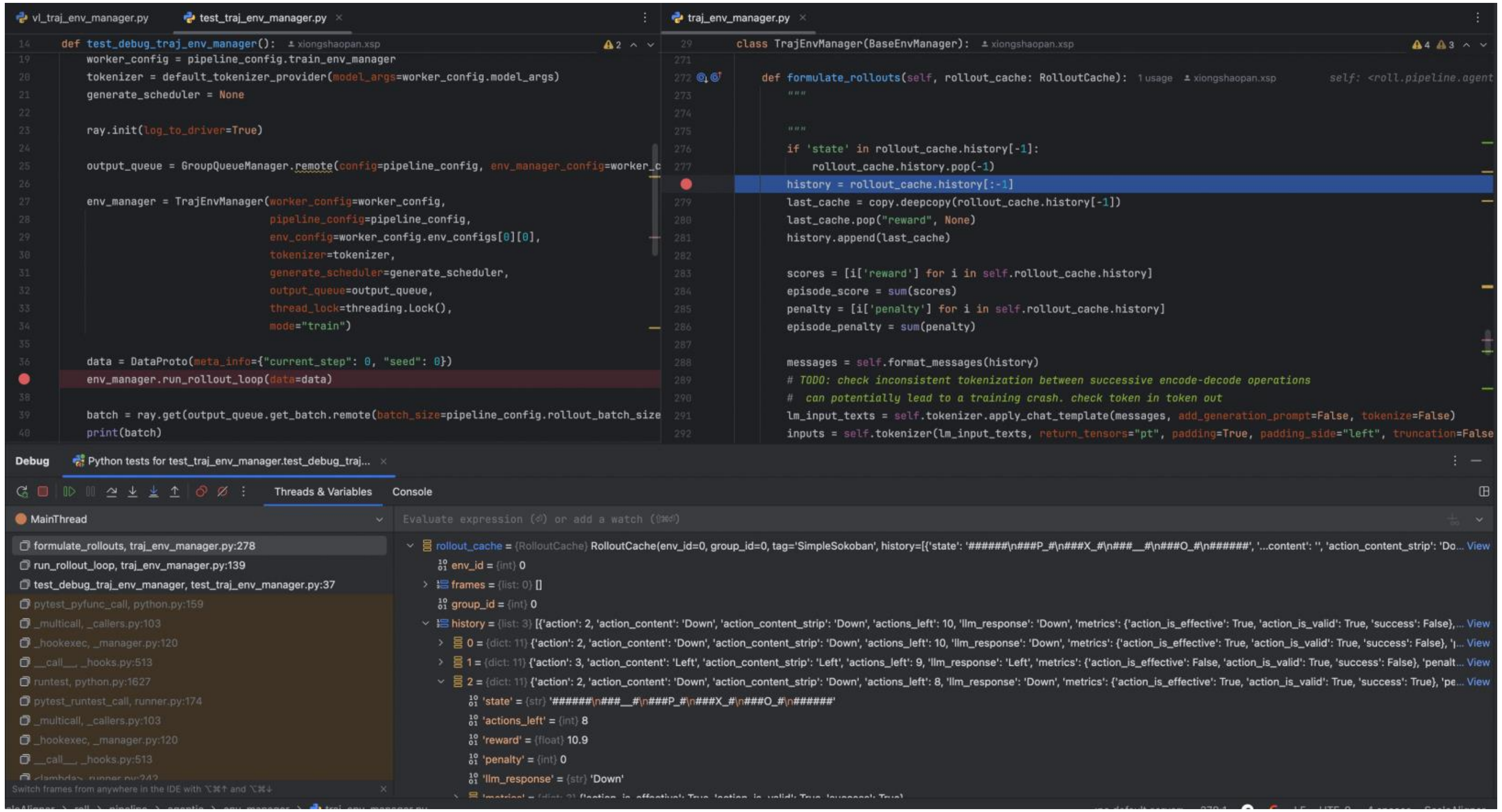
异步并行rollout

- EnvManager: 多轮交互的本地调试
- 痛点：多轮交互调试之殇
  - **高昂代价:** 调试LLM与环境多轮交互，需启动分布式，耗时且资源巨大
  - **效率低下:** 新环境、Prompt格式调试迭代周期长，成本高
  - **复杂难解:** 分布式环境下多轮交互问题排查困难

Agentic 异步并行rollout与异步训练

异步并行rollout

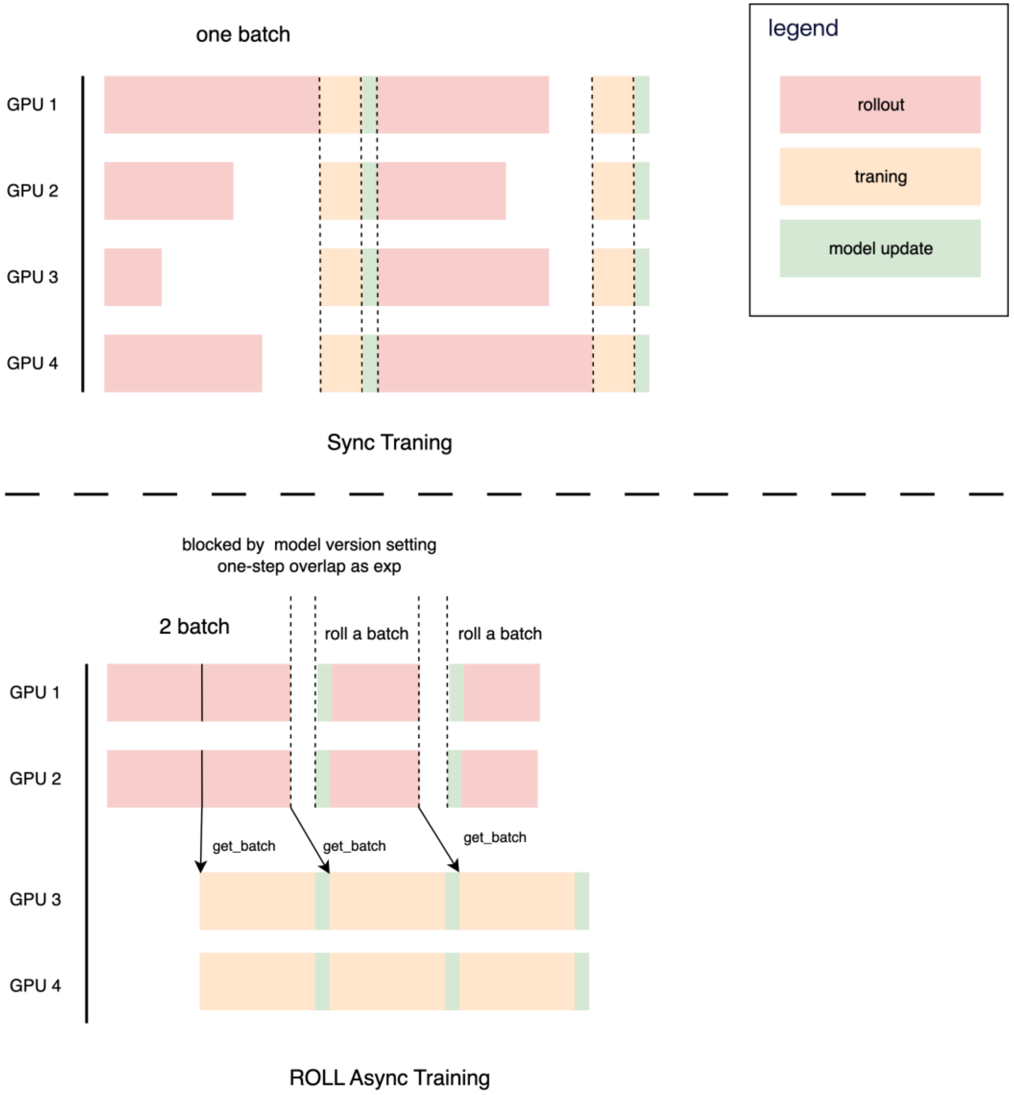
- ➢ EnvManager: 多轮交互的本地调试
- ➢ 解决方案：EnvManager的“自包含”设计
  - 核心: EnvManager 独立封装RL采样逻辑 (Env、LLM交互、数据收集)
  - 关键机制：
    - ☐ LLMProxy：实现可灵活切换，训练使用policy model，调试使用random/openai_chat_api
    - ☐ 开发者可在本地直接实例化EnvManager，轻松运行与调



Agentic 异步并行rollout与异步训练

异步训练

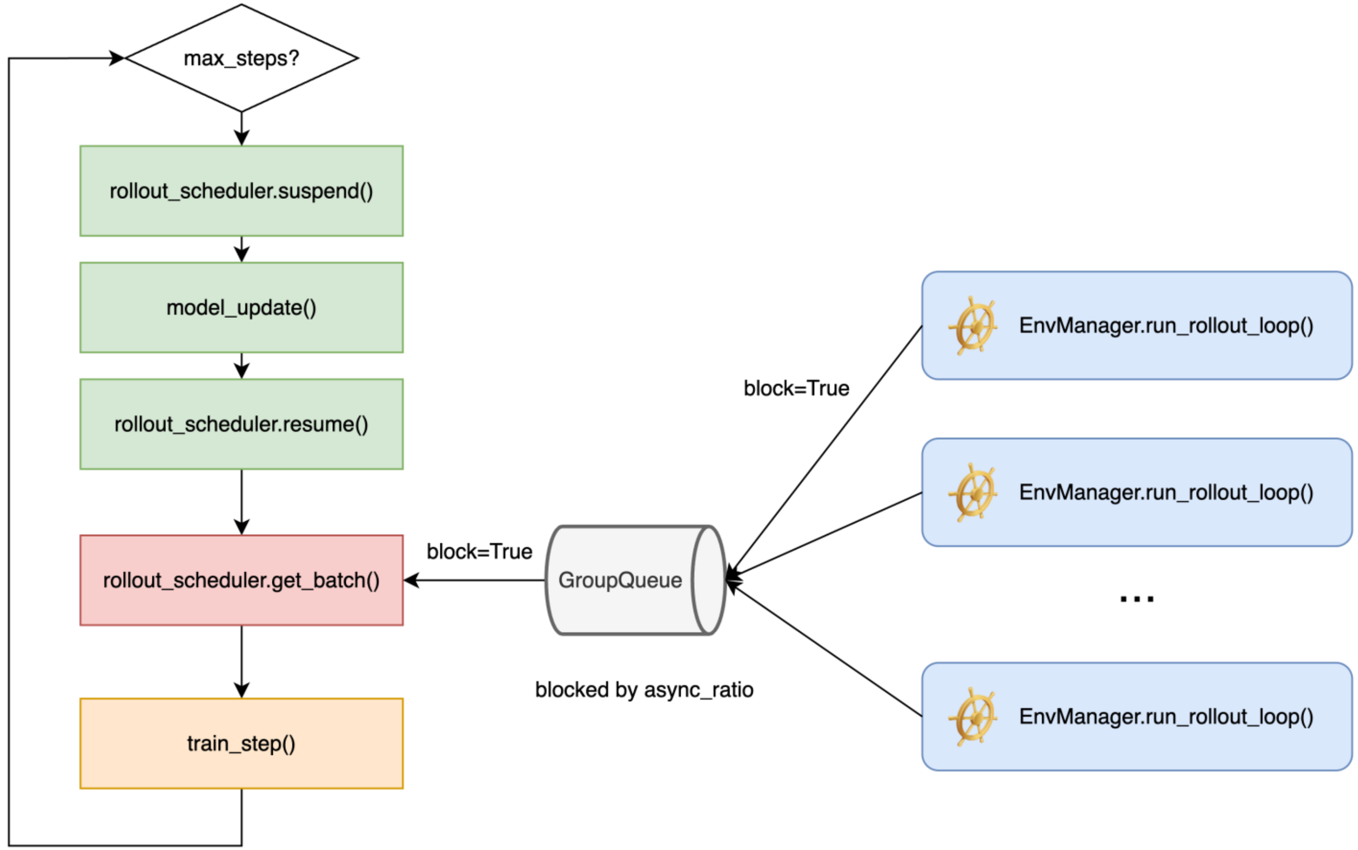
- ➢ Sync training
  - 多轮rollout、训练串行进行
  - 生成长尾问题严重，GPU空闲，利用率低
- ➢ ROLL: Async training
  - rollout/训练解耦，扩展性高
  - 消除等待，GPU持续工作



Agentic 异步并行rollout与异步训练

ROLL: Async training 实现

- 异步并行rollout (EnvManager)
    - EnvManager.run_rollout_loop() 并行运行
    - 异步生成agent与env交互轨迹
- 数据缓冲与解耦 (GroupQueue)
    - 各EnvManager 将轨迹推送到 GroupQueue
- 异步训练循环 (Training Loop)
    - suspend(): 停止rollout，准备模型更新
    - model_update()): actor_train -> actor_infer
    - resume(): 新模型恢复rollout
    - get_batch(): block式从GroupQueue取数据
    - train_step(): 模型训练



Part 3：ROLL实践篇

易用性与灵活性：满足你的个性化需求

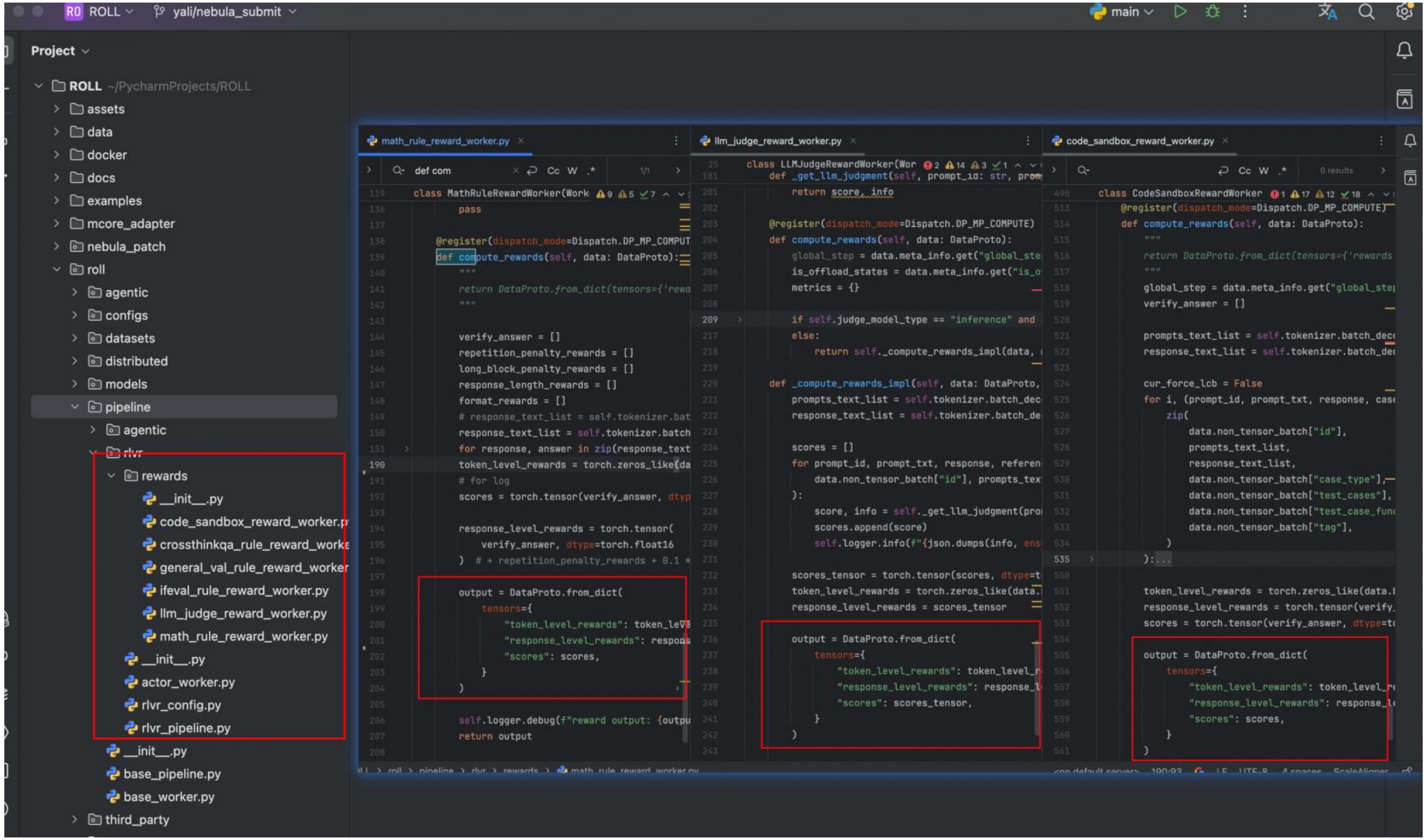
- 自定义Pipeline:
  - 算法同学可直接DIY RL计算流，编排Worker的计算流程
  - 内置rlvr_pipeline和agentic_pipeline作为标准模板
  - 专注于业务逻辑，无需关心底层分布式细节

pipeline demo

```python
class PipelineDemo:
    def __init__(self):
        self.actor_train: Any = Cluster(role_type=TrainRole, resource_pool=4, config=dict(dp=2,
        self.actor_infer: Any = Cluster(role_type=TrainRole, resource_pool=4, config=dict(dp=2,
        self.reference: Any = Cluster(role_type=InferRole, resource_pool=2, config=dict(dp=2, tp
        self.reward: Any = Cluster(role_type=InferRole, resource_pool=4, config=dict(dp=2, tp=2)
        self.critic: Any = Cluster(role_type=InferRole, resource_pool=4, config=dict(dp=2, tp=1)

        self.actor_train.initialize()
        self.actor_infer.initialize()
        self.reference.initialize()
        self.reward.initialize()
        self.critic.initialize()

        self.model_update_group = ModelUpdateGroup(src_cluster=self.actor_train, tgt_cluster=sel

        num_samples = 64
        seq_length = 8
        batch_size = 8
        dataset = CustomDataset(num_samples=num_samples, seq_length=seq_length)
        self.data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

    def run(self):
        step = 0
        for batch_data in self.data_loader:
            batch = DataProto.covert(batch=batch_data)
            batch = self.actor_infer.generate(batch=batch)
            batch = self.reference.compute_logprobs(batch=batch)
            batch = self.reward.forward(batch=batch)
            batch = self.critic.forward(batch=batch)
            batch = compute_advantage(batch)
            self.actor_train.train_step(batch=batch)
            self.critic.train_step(batch=batch)
            self.model_update_group.model_update()
            self.do_checkpoint(step=step)
            step += 1
```

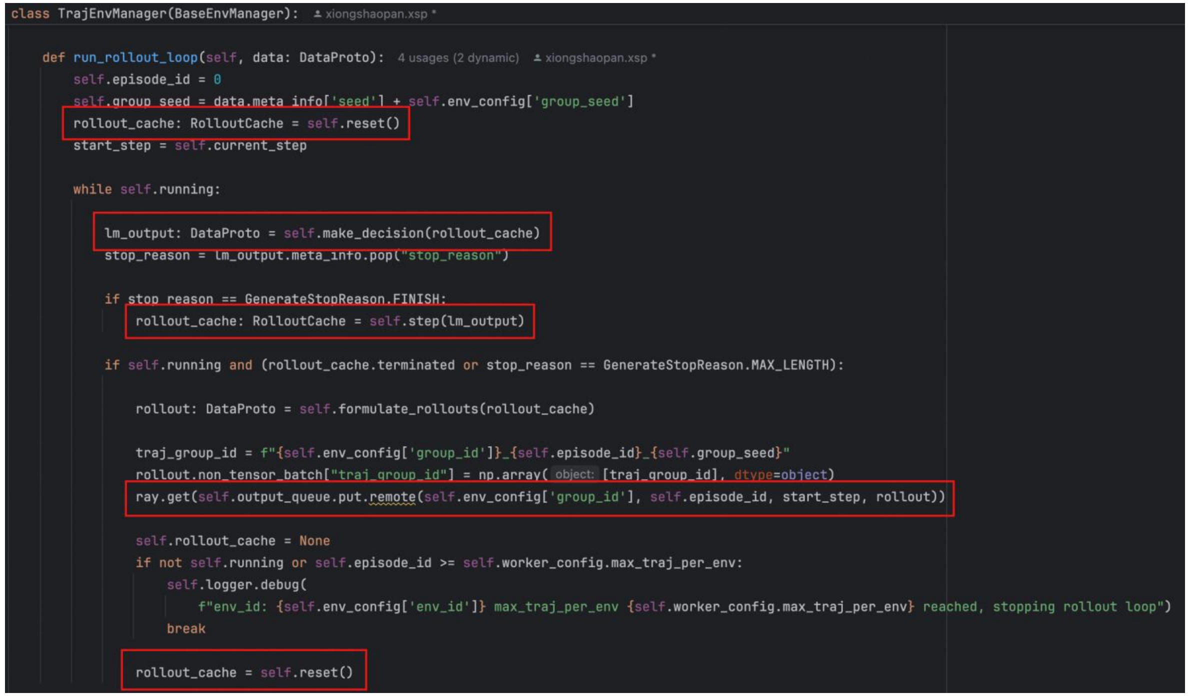
易用性与灵活性：满足你的个性化需求

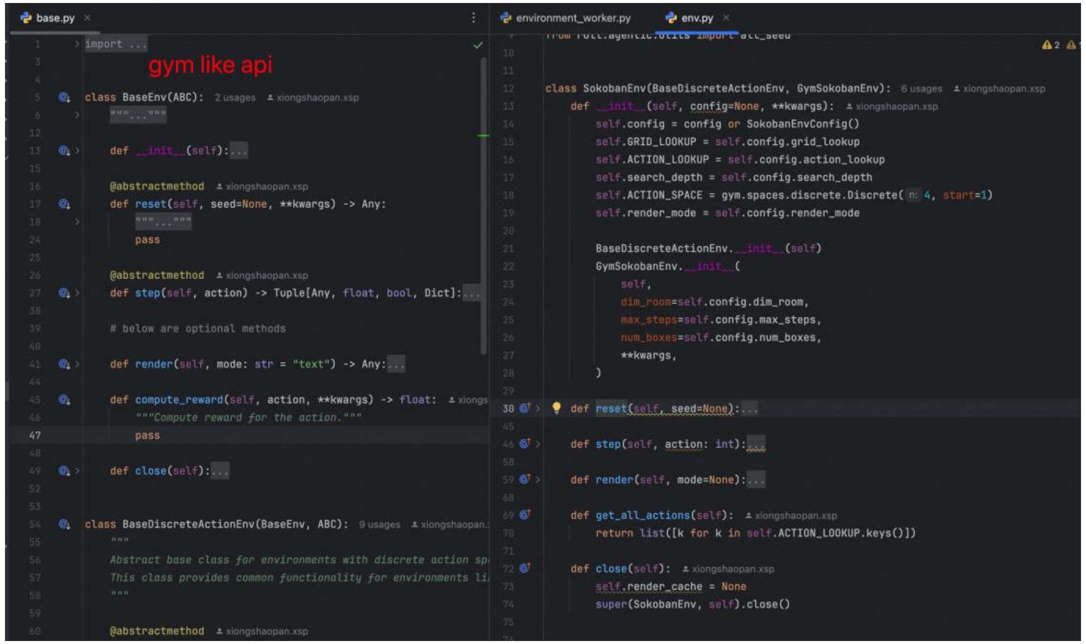
- 自定义Reward
  - 内置多种常用奖励计算方式（数学规则、代码沙箱、LLM as Judge等）
  - 提供清晰接口，轻松扩展自定义奖励逻辑，满足特定业务需求



易用性与灵活性：满足你的个性化需求

- 自定义Environment与多轮交互
  - 提供Gym-like标准接口，快速构建和扩展自定义环境
  - 支持Ray Actor和线程级部署，灵活选择
  - 用户可控的RL Rollout Loop: 自由编写Agent与Env交互逻辑，包括多轮对话、工具调用等
  - 极致的开发效率：多轮交互可本地调试





代码实操

用户配置 .yaml

➢ Model Scope 模型下载配置

rlvr_qwen2.5_7B_megatron_vllm_8gpus_model_scope.yaml

```yaml
#    - baseline


model_download_type: MODELSCOPE

  llm_judge:
    # NOTE: llm as judge 也需要gpu, 不能和actor infer共享gpu
    worker_cls: roll.pipeline.rlvr.rewards.llm_judge_reward_worker.LLMJudgeRewardWorker
    judge_prompt: Qwen2.5-7B-Instruct-RLVR-prompt
    judge_model_type: inference
    tag_included: [RLVR]
    model_args:
      model_name_or_path: AI-ModelScope/Qwen2.5-7B-Instruct-RLVR
      disable_gradient_checkpointing: true
      dtype: bf16
      model_type: trl
```

➢ Swanlab 实验配置

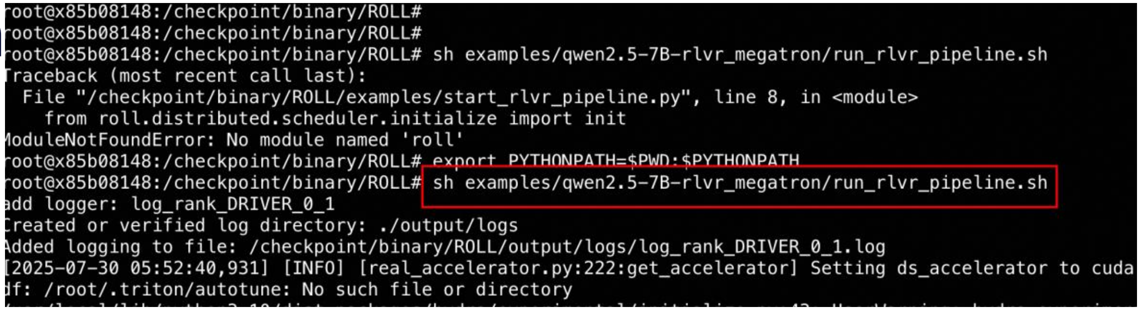
```yaml
track_with: swanlab
tracker_kwargs:
  login_kwargs:
    api_key: your_api_key
  project: roll-rlvr
  logdir: debug
  experiment_name: roll-rlvr-examples
  tags:
    - roll
    - rlvr
    - debug
```

运行环境

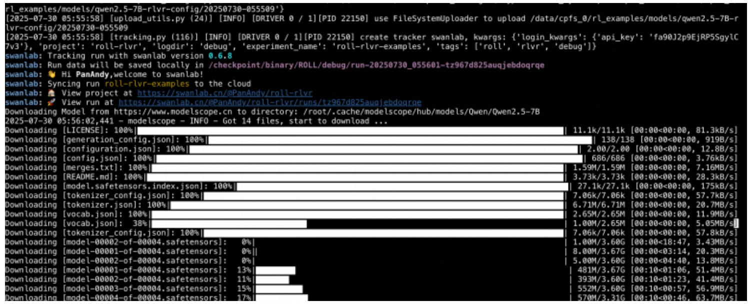
- Python3.10
- Torch 2.6.0
- 镜像：roll-registry.cn-hangzhou.cr.aliyuncs.com/roll/pytorch:nvcr-24.05-py3-torch260-vllm084

代码实操

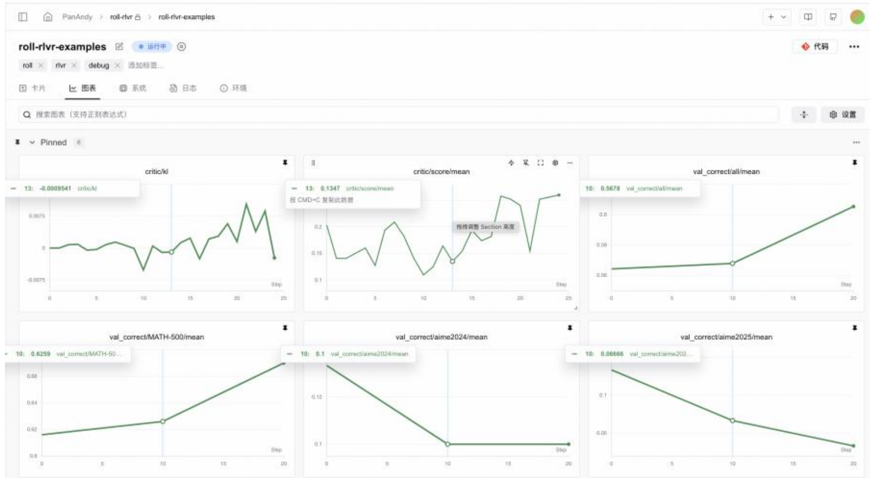
任务启动：sh run_pipeline.sh



- sh run_rlvr_pipeline.sh 一键启动



- model scope 模型下载
- SwanLab 初始化



- SwanLab 实验监控

代码实操

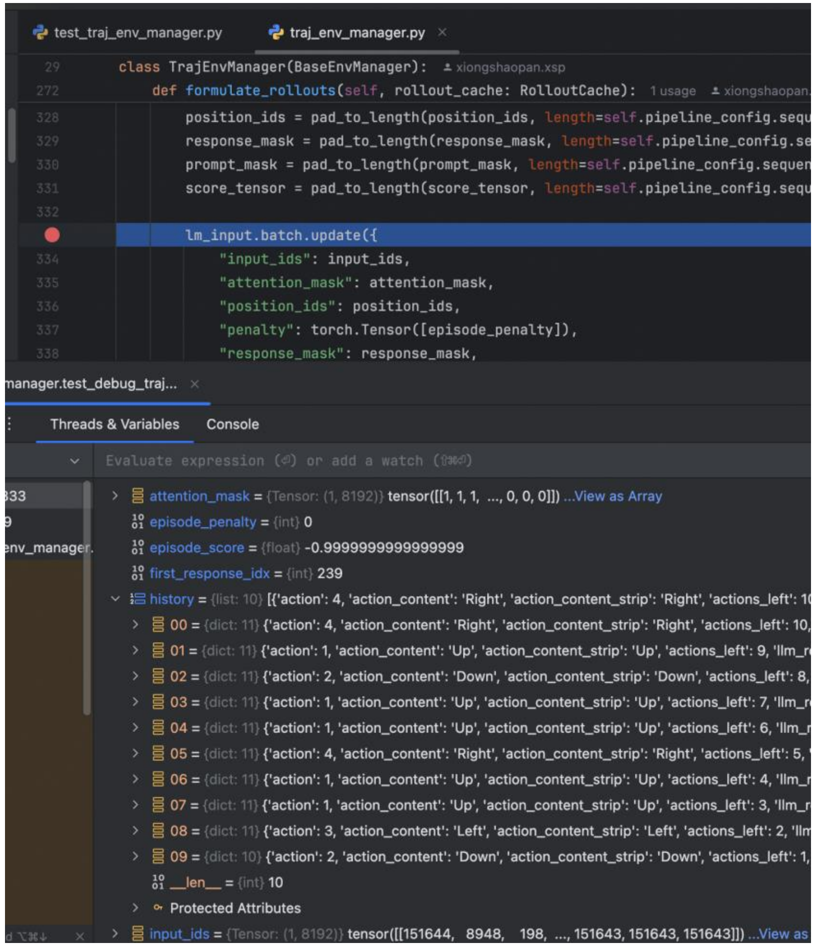
多轮交互EnvManager本地调试

本地依赖安装(mac)：

test_traj_env_manager.py

```
"""
usage:

conda create -n python310_torch260_em  python=3.10

pip3 install torch torchvision torchaudio py-cpuinfo
pip install -r requirements_em_local_debug.txt

python tests/agentic/env_manager/test_traj_env_manager.py
"""
import threading
```

IDE debug/单步调试：



ROLL！！！

LitePPO: http://arxiv.org/abs/2508.08221

- **大规模消融实验**:
  - 基于ROLL 框架，覆盖 4/8B Base & Instruct、三档难度数据、两大奖励尺度，完整复现并剖析 8 类主流技巧
- **LitePPO**:
  - 仅组合 “Group-level mean + Batch-level std Advantage Norm” 与 “Token-level Loss” 两项技巧，在 6 个数学基准上平均超越 GRPO/DAPO，且训练曲线更稳定
- **技巧指南**:
  - Norm：Group-mean/Batch-std 最稳健；奖励集中时去掉 std
  - Clip-Higher：只对已对齐模型有效，小模型存在 “缩放律”
  - Token-level Loss：对 Base 模型必用，对 Instruct 模型反而略差
  - Overlong Filtering：短/中长度任务增益显著，长尾推理作用有限

ROLL Future Life Lab 2025-08-12

arXiv:2508.08221v1 [cs.LG] 11 Aug 2025

Part I: Tricks or Traps?
A Deep Dive into RL for LLM Reasoning

Zihe Liu[*♡α], Jiashun Liu[*◇α], Yancheng He[*α], Weixun Wang[*†α], Jiaheng Liu[Ω], Ling Pan[◇], Xinyu Hu[α¶], Shaopan Xiong[α], Ju Huang[α], Jian Hu[♣], Shengyi Huang[‡], Siran Yang[α], Jiamang Wang[α], Wenbo Su[α], Bo Zheng[α]

[α]Alibaba Group [♡]Beijing Jiaotong University
[◇]Hong Kong University of Science and Technology [Ω]Nanjing University
[¶]Peking University [♣]OpenRLHF [‡]CleanRL

Abstract

Reinforcement learning for LLM reasoning has rapidly emerged as a prominent research area, marked by a significant surge in related studies on both algorithmic innovations and practical applications. Despite this progress, several critical challenges remain, including the absence of standardized guidelines for employing RL techniques and a fragmented understanding of their underlying mechanisms. Additionally, inconsistent experimental settings, variations in training data, and differences in model initialization have led to conflicting conclusions, obscuring the key characteristics of these techniques and creating confusion among practitioners when selecting appropriate techniques. This paper systematically reviews widely adopted RL techniques through rigorous reproductions and isolated evaluations within a unified open-source framework. We analyze the internal mechanisms, applicable scenarios, and core principles of each technique through fine-grained experiments, including datasets of varying difficulty, model sizes, and architectures. Based on these insights, we present clear guidelines for selecting RL techniques tailored to specific setups, and provide a reliable roadmap for practitioners navigating the RL for the LLM domain. Finally, we reveal that a minimalist combination of two techniques can unlock the learning capability of critic-free policies using vanilla PPO loss. The results demonstrate that our simple combination consistently improves performance, surpassing strategies like GRPO and DAPO.

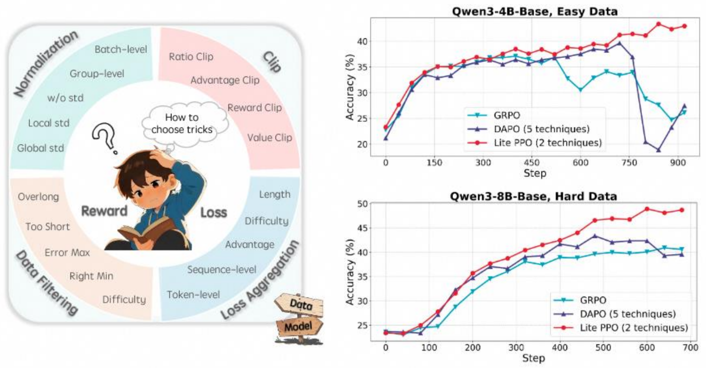


Figure 1: **Left**: The proliferation of RL optimization techniques, coupled with diverse initialized models and data, has raised barriers to practical adoption. **Right**: We establish detailed application guidelines via dissecting internal mechanisms of widely-used tricks, and introduce Lite PPO, a minimalist two-technique combination that enhances learning capacity in critic-free policies with vanilla PPO loss. The average accuracy is calculated across six mathematical benchmarks.

* Equal Contribution. † Corresponding to: Weixun Wang <weixun.wwx@taobao.com>.

ROLL
like a Reinforcement Learning Algorithm Developer

ROLL！！！

- 行动起来： 加入ROLL社区，一起探索LLM RL的未来!
  - GitHub: https://github.com/alibaba/ROLL
  - Paper: https://arxiv.org/abs/2506.06122

扫码进ROLL项目交流群

欢迎加入

欢迎加入

欢迎加入

非常感谢