PR #1740 完整报告

radixark/miles

feat (async): support evaluation for fully-async training (dedicated fleet / pause-the-world / external service)

合并时间：2026-08-04 12:23

原文链接：http://prhub.com.cn/radixark/miles/pull/1740

执行摘要

- 一句话：fully-async 评估：共享引擎 / 专用机群 / 外部黑盒三种姿态
- 推荐动作：值得精读。核心设计决策包括：按权重来源切分评估姿态、快照单一 owner 防泄漏、版本钉住与逐引擎读回校验、--eval-sglang-* 继承式参数与 tp-coupled 回退、黑盒契约保持目录进 / 结果出。建议重点阅读 eval_dispatch.py、eval_fleet.py、checkpoint_eval.py 三件套及其 fast 测试，可复用到任何需要异步评估的 RL 训练框架。合并后应补跑一次 head commit 上共享引擎暂停姿态的 GPU 验证。

功能与动机

PR body 明确指出："Fully-async rollout has no evaluation story: the producer never stops, so there is no quiet window to evaluate in, and running eval on the training engines is structurally broken — eval requests get aborted by every weight update, and no well-defined weight version is measured." 同时 "Heavy eval sets also steal training inference capacity even in sync mode"，即重型评估集即使在同步模式下也会挤占训练推理容量。因此需要一种让训练不被中断、且每次评估都钉住确定权重版本的方案。

实现拆解

1. 定义评估姿态与契约：新增 miles/rollout/checkpoint_eval.py，定义 CheckpointEvalFn（黑盒契约：目录进、结果出）、EvalSkip（可归属跳过）、retarget_args（浅拷贝 args 并替换 router 与 GPU 规模）和 is_checkpoint_eval_fn。两种姿态按权重来源切分：共享训练引擎用阻断式调用钉住版本；快照姿态只通过 (rollout_id, HF snapshot dir) 交互，依赖 update_weights_from_disk 加载并把 weight_version=str(rollout_id) 作为版本。args.eval_uses_snapshots 是唯一判别量，由 _resolve_rollout_functions 派生，驱动侧 EvalDispatcher 与 RolloutManager 共享但互不 import。
2. 两种快照后端：job 内专用 eval fleet（--eval-num-gpus）由 miles/ray/rollout/eval_fleet.py 的 EvalFleet.pin 实现——健康探测（无故障容错时调用 RolloutServer.probe_and_mark_dead）、recover、wait_all_engines_alive、逐引擎 update_weights_from_disk 加 get_weight_version 读回校验、路由器就绪探针，成功后返回 GenerateState 交给配置好的 eval fn。黑盒后端（--eval-function-path 指向 CheckpointEvalFn 子类）直接拿到快照目录自行加载与生成，examples/fully_async/external_eval_fn.py 是参考实现：通过

MILES_EXTERNAL_EVAL_GPUS/MILES_EXTERNAL_EVAL_URL 启动或挂接独立 sglang server，并用 _pin 完成 load 后版本读回。两种后端互斥，统一走 RolloutManager._eval_checkpoint 单一路径。

3. 驱动侧异步派发与快照所有权：miles/ray/rollout/eval_dispatch.py 的 EvalDispatcher 负责有界并发（--eval-max-in-flight）、溢出策略（--eval-overflow-policy backpressure|skip）、快照导出（actor_model.export_hf）与 _retire GC（受 --eval-keep-snapshots 约束）。设计核心是快照单一 owner：导出与回收都在 dispatcher，每个结算点无论成功、跳过还是崩溃都恰好回收一个快照，避免任何失败路径向 --eval-hf-dir（文档建议 /dev/shm）泄漏模型大小目录。
4. 引擎配置与参数继承：miles/backends/sglang_utils/arguments.py 用 _add_prefixed_server_args 注册 --eval-sglang-*（argparse.SUPPRESS 默认 + BooleanOptionalAction），collect_eval_sglang_overrides 收集用户显式覆盖；miles/ray/rollout/rollout_server.py 的 _apply_eval_model_config/_eval_sglang_overrides 把 eval 模型组填入 SglangConfig。TP 取自 --eval-num-gpus-per-engine；当 eval TP 与 rollout TP 不同时，dp/pp/ep/attn_cp 回退 1，避免 tp=1 ep=8 这类 SGLang 启动校验直接拒绝的配置。sglang_overrides 合并被移到 args 派生分支之后，修复了 per-group override 被 --sglang-config 条件分支覆盖的问题。
5. 导出重构与测试配套：miles/backends/megatron_utils/hf_export.py 从 model.py 接收 save_hf_model 并新增 export_hf_model_direct——走 miles 自身 megatron→HF 转换器，导出覆盖范围与权重同步一致；.complete marker 在导出前清理、LoRA adapter 写完后补写；导出 barrier 移入 finally，rank 0 单点失败也不会让其他 rank 卡死直到 NCCL watchdog。测试侧：tests/e2e/megatron/test_qwen3_4b_fully_async_eval.py 以短 Qwen3-4B GRPO 跑覆盖共享 /fleet/external 三种姿态；fast 测试覆盖契约、fleet pin/recover、dispatcher 策略与快照所有权、manager 快照路径、--eval-sglang-* 继承与优先级；文档同步更新 docs/user-guide/fully-async.md 与 CLI 参考。

关键文件：

- miles/rollout/checkpoint_eval.py（模块 评估契约；类别 source；类型 core-logic；符号 CheckpointEvalFn, evaluate_checkpoint, EvalSkip, retarget_args）：新增黑盒评估契约，是快照姿态与外部后端的公共接口，定义 CheckpointEvalFn/EvalSkip/retarget_args。
- miles/ray/rollout/eval_fleet.py（模块 评估机群；类别 source；类型 core-logic；符号 EvalFleet, pin, _pin_fleet, _fleet_args）：实现专用评估机群的核心 pin 逻辑：健康探测、恢复、逐引擎加载与版本读回校验。
- miles/ray/rollout/eval_dispatch.py（模块 派发器；类别 source；类型 core-logic；符号 EvalDispatcher, dispatch, drain, _export）：驱动侧异步派发器，承载快照导出、有界并发、溢出策略与快照唯一 owner 的回收逻辑。
- miles/backends/megatron_utils/hf_export.py（模块 HF 导出；类别 source；类型 dependency-wiring；符号 export_hf_model_direct, save_hf_model, _is_hf_metadata_file, _get_hf_bridge）：HF 导出从 model.py 迁入并新增直连 megatron→HF 导出器，修复导出 barrier 的 rank 0 单点失败问题。
- miles/ray/rollout/rollout_manager.py（模块 管理器；类别 source；类型 core-logic；符号 eval, _eval_checkpoint, report_eval_skip）：新增 _eval_checkpoint 单一路径：fleet

pin 优先，再调用配置的评估 fn，所有失败降级为可归属跳过。

- miles/backends/sglang_utils/arguments.py（模块 参数层；类别 source；类型 configuration；符号 _add_prefixed_server_args, add_sglang_arguments, collect_eval_sglang_overrides）：注册 --eval-sglang-* 继承式参数与布尔 --no- 形式，是 eval fleet 引擎配置的入口。
- miles/ray/rollout/rollout_server.py（模块 引擎服务；类别 source；类型 core-logic；符号 _eval_sglang_overrides, _apply_eval_model_config, _resolve_sglang_config, probe_and_mark_dead）：在 SglangConfig 中追加 eval 模型组，应用 tp-coupled 回退与 eval overrides，并新增 probe_and_mark_dead。
- miles/utils/arguments.py（模块 参数校验；类别 source；类型 configuration）：新增 eval 相关参数与校验矩阵，并修复 fully-async override 之后再解析默认 eval 路径的关键缺陷。
- examples/fully_async/external_eval_fn.py（模块 外部后端；类别 source；类型 core-logic；符号 ExternalSglangEvalFn, evaluate_checkpoint, _pin, dispose）：参考 CheckpointEvalFn 实现：启动或挂接独立 sglang 服务，是外部黑盒姿态的模板。
- examples/fully_async/run_qwen3_5_4b_fully_async_eval.py（模块 示例脚本；类别 source；类型 core-logic；符号 ScriptArgs, prepare, execute, main）：同一脚本通过 --eval-backend fleet|external 切换两种快照后端，是功能示例与 e2e 的参照。
- tests/e2e/megatron/test_qwen3_4b_fully_async_eval.py（模块 评估测试；类别 test；类型 test-coverage；符号 prepare, execute）：覆盖共享 /fleet/external 三种姿态的 GPU e2e，是本次功能的主要验收测试。

关键符号：CheckpointEvalFn.evaluate_checkpoint, EvalSkip, retarget_args, EvalFleet.pin, EvalDispatcher.dispatch, RolloutManager._eval_checkpoint, FullyAsyncRolloutFn._call_eval, export_hf_model_direct, save_hf_model, collect_eval_sglang_overrides, _apply_eval_model_config, ExternalSglangEvalFn

关键源码片段

miles/rollout/checkpoint_eval.py

新增黑盒评估契约，是快照姿态与外部后端的公共接口，定义 CheckpointEvalFn/EvalSkip/retarget_args。

```
# miles/rollout/checkpoint_eval.py —— 黑盒评估后端契约。
# 训练侧负责：导出快照、异步派发、结果记在快照对应 step、之后回收目录；
# 后端负责：在 __init__ 里准备服务，在 evaluate_checkpoint 里消费快照目录。
import abc
import copy
import logging
from argparse import Namespace

from miles.rollout.base_types import RolloutFnEvalInput, RolloutFnEvalOutput, RolloutFnInput

logger = logging.getLogger(__name__)
```

```
class EvalSkip(Exception):
    # 抛出自定义异常即可让该评估点被登记为 eval/skipped_{reason}，
    # 而不是被当作崩溃处理 —— 训练循环永远不会因此中断。
    def __init__(self, reason: str):
        super().__init__(reason)
        self.reason = reason


def retarget_args(args: Namespace, router_ip, router_port, num_gpus: int, num_gpus_per_
engine: int) -> Namespace:
    # 浅拷贝 args，只替换路由器地址与 GPU 规模；下游 GenerateState 与
    # 生成函数都从 args 读取这些字段，因此同一套评估代码可无修改地
    # 指向另一组引擎（共享引擎 / fleet / 外部服务共用这一份逻辑）。
    eval_args = copy.copy(args)
    eval_args.sglang_router_ip = router_ip
    eval_args.sglang_router_port = router_port
    eval_args.rollout_num_gpus = num_gpus
    eval_args.rollout_num_gpus_per_engine = num_gpus_per_engine
    return eval_args


class CheckpointEvalFn(abc.ABC):
    # 黑盒契约：目录进、结果出，中间是后端自己的事。
    @abc.abstractmethod
    async def evaluate_checkpoint(self, checkpoint_dir: str, input: RolloutFnEvalInput) ->
    RolloutFnEvalOutput:
        ...

    async def __call__(self, input: RolloutFnInput) -> RolloutFnEvalOutput:
        # 只服务 eval；训练 fn 必须留在 --rollout-function-path。
        assert input.evaluation, "CheckpointEvalFn only serves eval; keep the train fn on --rollout-
        function-path"
        assert input.hf_dir is not None, (
            "no snapshot was dispatched — checkpoint eval fns require train_async.py "
            "and a snapshot source (--eval-hf-dir or --save-hf)"
        )
        return await self.evaluate_checkpoint(input.hf_dir, input)

    def dispose(self) -> None:
        # 可选钩子：销毁 __init__ 里启动的进程 / 客户端，默认 no-op。
        ...
```

miles/ray/rollout/eval_fleet.py

实现专用评估机群的核心 pin 逻辑：健康探测、恢复、逐引擎加载与版本读回校验。

```
# miles/ray/rollout/eval_fleet.py —— job 内专用评估机群（--eval-num-gpus）。
# 机群只负责"把权重送到位"：pin 加载快照、逐引擎读回版本并确认一致，
# 然后把 GenerateState 交还，由配置好的评估 fn 去生成 —— 与训练引擎上的路径完全一致。
import asyncio
```

```
import logging
from argparse import Namespace

from miles.rollout.checkpoint_eval import EvalSkip, retarget_args
from miles.rollout.inference_rollout.inference_rollout_common import GenerateState
from miles.utils.http_utils import wait_http_ok

logger = logging.getLogger(__name__)
EVAL_WEIGHT_LOAD_TIMEOUT_SECS = 600.0


class EvalFleet:
    def __init__(self, args: Namespace, *, srv):
        self.args = args
        self._srv = srv
        self._state = GenerateState(self._fleet_args())

    async def pin(self, checkpoint_dir: str, weight_version: str) -> GenerateState:
        # 健康检查三段式：无健康监控时自己探测，recover 重启死引擎，等全部复活。
        try:
            if not self.args.use_fault_tolerance:
                await self._srv.probe_and_mark_dead()
            await self._srv.recover()
            await self._srv.wait_all_engines_alive()
        except Exception as e:
            logger.warning(f"Eval fleet unhealthy: {e}")
            raise EvalSkip("unhealthy") from e

        if not await self._pin_fleet(checkpoint_dir, weight_version):
            raise EvalSkip("pin_violation")

        # 引擎复活后路由器可能还在 503，用单 token 探针确认路由可用再放行。
        try:
            await self._wait_router_ready()
        except Exception as e:
            logger.warning(f"Eval router not ready: {e}")
            raise EvalSkip("unhealthy") from e
        return self._state

    def _fleet_args(self) -> Namespace:
        router_ip, router_port = self.args.sglang_model_routers["eval"]
        return retarget_args(
            self.args, router_ip, router_port, self.args.eval_num_gpus, self.args.eval_num_gpus_
            per_engine
        )

    async def _pin_fleet(self, checkpoint_dir: str, weight_version: str, *, retries: int = 2) -> bool:
        # 路由器在引擎间做负载均衡，任何一台版本不一致都会造成混合版本；
        # 所以逐台 update_weights_from_disk 后必须全部 get_weight_version 一致才算成功。
```

```
        actors = [e.actor_handle for e in self._srv.engines]
        versions: list = []
        for attempt in range(retries):
            try:
                await asyncio.wait_for(
                    asyncio.gather(
                        *[a.update_weights_from_disk.remote(checkpoint_dir, weight_version=weight_
                        version) for a in actors]
                    ),
                    timeout=EVAL_WEIGHT_LOAD_TIMEOUT_SECS,
                )
                versions = await asyncio.wait_for(
                    asyncio.gather(*[a.get_weight_version.remote() for a in actors]),
                    timeout=EVAL_WEIGHT_LOAD_TIMEOUT_SECS,
                )
            except Exception as e:
                logger.warning(f"Weight pin to {checkpoint_dir} failed (attempt {attempt + 1}/{retries}
                ): {e}")
                continue
            if versions and all(str(v) == weight_version for v in versions):
                return True
        logger.warning(f"Failed to pin weight_version={weight_version} to {checkpoint_dir} (got
        {versions})")
        return False
```

miles/ray/rollout/eval_dispatch.py

驱动侧异步派发器，承载快照导出、有界并发、溢出策略与快照唯一 owner 的回收逻辑。

```
# miles/ray/rollout/eval_dispatch.py —— 驱动侧异步派发器（不阻塞训练循环）。
# 设计核心：快照的“导出”与“回收”由同一方持有，任何失败路径都不会泄漏
# 模型大小的目录；每个评估点无论成功、跳过还是崩溃，都恰好回收一个快照。
import logging
import os
import shutil
import time
from collections import deque

import ray

logger = logging.getLogger(__name__)


class EvalDispatcher:
    def __init__(self, args, actor_model, rollout_manager):
        self.args = args
        self.actor_model = actor_model
        self.rollout_manager = rollout_manager
        self.pending: deque[tuple[int, ray.ObjectRef, str | None]] = deque()
        self._exported: list[str] = []
```

```
async def dispatch(self, rollout_id: int, hf_dir: str | None = None, force: bool = False) ->
None:
    # 非快照姿态保持原有共享引擎调用形状（阻断式，让 FullyAsyncRolloutFn 自己暂停
    producer）。
    if not self.args.eval_uses_snapshots:
        await self.rollout_manager.eval.remote(rollout_id)
        return

    await self._reap_finished()
    if len(self.pending) >= self.args.eval_max_in_flight:
        if self.args.eval_overflow_policy == "skip" and not force:
            await self.rollout_manager.report_eval_skip.remote(rollout_id, "busy")
            return
        await self._settle(*self.pending.popleft()) # backpressure：先结算最老的

    export_time = None
    exported_dir = None
    require_marker = hf_dir is None # 调用方自带的目录（如训练前基线）没有 .complete
    if hf_dir is None:
        if self.args.eval_hf_dir is None:
            hf_dir = self.args.save_hf.format(rollout_id=rollout_id)
        else:
            hf_dir = os.path.join(self.args.eval_hf_dir, f"step_{rollout_id}")
            try:
                export_time = await self._export(rollout_id, hf_dir)
            except Exception as e:
                logger.error(f"HF snapshot export for eval {rollout_id} failed: {e}")
                shutil.rmtree(hf_dir, ignore_errors=True)
                await self.rollout_manager.report_eval_skip.remote(rollout_id, "export_failed")
                return
            exported_dir = hf_dir

    ref = self.rollout_manager.eval.remote(
        rollout_id, hf_dir=hf_dir, export_time_seconds=export_time, require_marker=require_
        marker
    )
    self.pending.append((rollout_id, ref, exported_dir))

async def _settle(self, rollout_id: int, ref, exported_dir: str | None) -> None:
    try:
        await ref
    except Exception:
        logger.exception(f"Async eval for rollout {rollout_id} raised")
        await self.rollout_manager.report_eval_skip.remote(rollout_id, "crashed")
    finally:
        self._retire(exported_dir)

def _retire(self, exported_dir: str | None) -> None:
    if exported_dir is None:
```

```
            return
        self._exported.append(exported_dir)
        while len(self._exported) > self.args.eval_keep_snapshots:
            victim = self._exported.pop(0)
            shutil.rmtree(victim, ignore_errors=True)
            logger.info(f"GC'd consumed eval snapshot {victim}")
```

评论区精华

Review 中最有价值的交锋来自 ashtonchew 对共享引擎 eval 契约的核对：resolve_rollout_function_paths() 在 fully-async override 之前计算默认 eval path，导致 --fully-async 默认得到 (FullyAsyncRolloutFn, InferenceRolloutFn) 两个不同路径，RolloutManager 因而构造了第二个 InferenceRolloutFn 并绕过 FullyAsyncRolloutFn._call_eval() 的 producer pause。他建议的有界修复是 eval_path = args.eval_function_path or rollout_path（在 fully-async override 之后解析），并已用 resolver 测试复现；该问题由提交 b0aa5e5 修复。此外 gemini-code-assist 提出三处健壮性建议：args.hf_checkpoint 为 Hub ID 时 iterdir() 会崩溃（最终代码改为 is_dir() 判断后仅 warning）、用 ValueError/FileNotFoundError 替代 assert（避免 -O 下被 strip）、探针避免捕获过宽异常。

- 共享引擎 eval 路径解析失控：默认 eval_path 未经过 fully-async override (correctness): 采用有界修复：eval_path = args.eval_function_path or rollout_path，放在 fully-async override 之后；显式 --eval-function-path 不受影响。
- 快照所有权分散导致失败路径泄漏模型目录 (design): 快照单一 owner 收敛到 EvalDispatcher：每个结算点无论结果都恰好回收一个快照，_retire 按 eval_keep_snapshots 做 GC。
- eval TP 不同时继承 dp/pp/ep/attn_cp 导致引擎启动校验失败 (correctness): eval TP 与 rollout TP 不同时，tp-coupled 的四个维度回退为 1，用户可用 --eval-sglang-* 显式覆盖。
- args.hf_checkpoint 为 Hub ID 时 iterdir 崩溃 (correctness): 最终 export_hf_model_direct 先判断 base_checkpoint.is_dir()，非本地目录时仅记录 warning 并跳过 metadata 拷贝。
- 用 assert 做参数校验在 -O 下会被剥离 (design): Review 中提出但合并前未系统性替换；当前以 assert 为主，风险较低但值得后续统一处理。

风险与影响

- 风险：
  1. 共享引擎暂停姿态未经验证：PR body 自述 "These runs predate the rework and need repeating on the head commit. In particular the shared-engine numbers were produced by code in which the producer pause could not run, so that posture is effectively unverified"。合并时该姿态的实际暂停效果仍缺 GPU 验证。
  2. 跨模块重构回归：save_hf_model 从 model.py 迁入 hf_export.py，任何仍从旧路径 import 的插件或脚本会断裂；_compute_server_args 的 overrides 合并顺序修复会影响所有使用 --sglang-config 的多组（含 PD/ 多 LoRA）配置。

3. 外部依赖耦合：retract 暂停模式需要 sgl-project/sglang#31962 与 #1750 才能可靠工作；外部服务姿态依赖 sglang 的 update_weights_from_disk/model_info 版本读回语义。
4. 快照竞争与资源：快照在 /dev/shm 上 staging，eval_keep_snapshots + eval_max_in_flight 个模型大小目录可能超过容器 shm 配额；fleet pin 与 router 就绪之间存在引擎复活后的 503 窗口，依赖单 token 探针兜底。
5. CI 覆盖盲区：GPU e2e 依赖 run-ci-eval label 才能被执行，labelless 的绿跑并不覆盖该测试；合并后共用引擎姿态验证不足构成监控盲区。 - 影响：对用户：fully-async 训练首次获得完整评估方案，且不中断训练；新增 --eval-num-gpus、--eval-function-path、--eval-sglang-*、--eval-max-in-flight、--eval-overflow-policy 等参数。对系统：默认（同步训练且不配置快照姿态）路径保持原有调用形状不变，eval_rollout_single_dataset 的评分语义被单独 revert 回失败样本计 0.0，避免静默移动既有 eval 曲线；hf_export 迁移与参数解析顺序修复是潜在行为变化点，需要关注 --sglang-config 用户。对团队：提供了可复用的黑盒评估契约与参考实现，外部评测服务可用同一套训练 args，无需手工复制配置。影响程度中高，主要集中在 fully-async 用户群与 --sglang-config 多组配置。 - 风险标记：核心路径变更，共享引擎暂停未经验证，快照生命周期复杂度，外部 sglang 依赖，跨模块导出重构，GPU e2e 依赖 CI label

关联脉络

- PR #1717 Rewrite fully-async rollout as FullyAsyncRolloutFn on the class-based rollout API: 本 PR 的共享引擎评估直接挂接 FullyAsyncRolloutFn._call_eval() 的 producer 暂停语义，并复用其类式 API 与状态实例。
- PR #1716 Move fully-async rollout from examples into miles/rollout: fully-async rollout 迁入核心库后，本 PR 在其基础上补充完整的评估链路，二者属于同一功能演进线。