

PR #2849 完整报告

radixark/miles

[multi-lora v2 7/7] Tinker gateway example and tests

合并时间: 2026-09-16 16:26

原文链接: <http://prhub.com.cn/radixark/miles/pull/2849>

执行摘要

- 一句话: 新增 Tinker 网关示例启动器、SDK 客户端与网关测试套件
- 推荐动作: 值得精读, 但建议带着「契约而非实现」的视角读三处:
examples/multi_lora/serve_qwen3_30b_a3b_tinker.py (单节点 8 卡的资源切分与参数分组, 是复刻其它模型的模板)、miles/tinker/arguments.py (Tinker 能力边界被写死在这里, 扩模型族必看)、tests/fast/tinker/core/test_service.py 与 harness.py (checkpoint 版本化、租户隔离、失败隔离的对外承诺都在这些断言里)。值得关注的设计决策有: 把上游 tinker-cookbook 当作 wire contract 的验收标准; 用 test_layering.py 以导入图约束 core/runtime/server 分层; sampler 保存走不可变目录并支持 pinned version 解析。注意三点: 本 PR 无逐行人工评审 (批准来自共同作者), 如需信任请自行核对最终 tree; run_id 每次调用生成导致重启后旧 checkpoint 路径不可达, 长跑场景务必显式固定; IO 失败终止整个网关的策略是否可接受, 建议结合业务 SLA 单独评估。

功能与动机

PR body 的表述是「Add the Qwen3-30B-A3B Tinker gateway launcher and an SDK marker-memorization client」, 即为前序 6 个 PR 建成的 multi-LoRA / Tinker 网关提供端到端可跑的参考配方与验收标准, 并「Stacked on #2845」。验收口径写得很明确: run_client_recipes.py 的文档字符串称 cookbook 的 sl_loop (SFT) 与 rl_loop (GRPO) 是「the executable definition of the Tinker wire contract; passing them is the gateway's acceptance bar」。此外 body 说明本 PR 还顺带把采样侧的语义收紧:「Existing sampler tests now exercise disk publication, snapshot resolution independently of the training lease, and training continuing after a sampling failure」, 并把 Tinker 的 CLI 契约从公共 parse_args 中抽离成独立模块。

实现拆解

1. 参数注册下沉 (入口层): 新增 miles/tinker/arguments.py, add_tinker_arguments() 用内部闭包统一生成 --tinker-server-host/-server-port/-base-model/-checkpoint-root 以及 train-attn/mlp/unembed 三组 BooleanOptionalAction; configure_tinker_args() 断言 --train-backend megatron、拒绝与 --target-modules/--exclude-modules 并用, 再经 _resolve_target_modules(load_hf_config(...)) 依据 model_type 展开为 q_proj/k_proj/v_proj/o_proj、gate_proj/up_proj/down_proj、lm_head, 写回 args.target_modules 交由通用 LoRA 校验器。配套 miles/utils/arguments.py 删除 19 行

内联注册改为调用该函数 (+3/-19) , serve_tinker.py 同步接线 (+8/-7) 。影响: 所有训练入口的 parse_args 都会走这条新路径。

2. 示例启动器: examples/multi_lora/serve_qwen3_30b_a3b_tinker.py 用 typer + U.dataclass_cli 提供 prepare (hf download Qwen3-30B-A3B) 与 serve; ScriptArgs.__post_init__ 从 output_dir 派生 save_dir、从 model_dir 派生 hf_checkpoint , run_id 由 U.create_run_id 每次调用生成, 故 checkpoint-root = save_dir/run_id。serve() 按 ckpt/lora/tinker/optimizer/perf/sglang/topology/misc 分组拼装 CLI, 最后交给 U.execute_train(train_script="serve_tinker.py"), 复用统一训练入口承载网关。
3. 客户端与验收口径: examples/multi_lora/run_multi_tenant_example.py 用 tinker SDK 的 api_key (tml- 前缀) 作为租户标识, --mode single|multi 控制并发数, 每个客户端训练不同 marker 并在 greedy 采样中校验「loss 下降 + 自己的 marker 出现」; run_client_recipes.py 直接跑上游 tinker-cookbook 的 sl_loop/rl_loop, 把上游 recipe 当作 wire contract 的通过标准。
4. 测试与测试桩: tests/fast/tinker/**覆盖ore (future/model_queue/scheduler/service) 、runtime 的 datum→batch 翻译与 SGLang 参数映射、server/app 的 HTTP 契约、proto_codec 的 protobuf/JSON 双路一致性; test_layering.py 用导入图断言 core 只依赖 stdlib、只有 runtime 允许 import miles; harness.py 的 FakeBackend 复用 write_checkpoint_dir 落盘并支持 fail_next/fail_on 注入; tests/fast/backends/training_utils/test_checkpoint_io.py 覆盖「进程在 link 替换前后退出 + 后续重试」的发布原子性。原有过时的 sampler-push readiness 用例与 backend fake 方法被删除, fixture 改为通过 run() 停止服务。
5. e2e、文档与快照: tests/e2e/lora/tinker_gateway.py 提供 prepare_gateway/_wait_for_gateway/running_gateway 夹具, test_tinker_cookbook_gateway.py 与 test_tinker_multi_tenant_gateway.py 两条 8 卡 H200 用例由 run-ci-multi-lora 标签选中; README 与生成文档镜像说明共享 checkpoint root、frozen base 加载、不可变 sampler 目录与按需引擎加载; prepare/serve 均有记录型命令快照防 CLI 漂移。

关键文件:

- examples/multi_lora/serve_qwen3_30b_a3b_tinker.py (模块 启动脚本; 类别 source; 类型 core-logic; 符号 ScriptArgs, post_init, prepare, serve) : 本 PR 的主要交付物: Qwen3-30B-A3B Tinker 网关启动器, 定义 8 卡单节点的训练 / 采样切分、adapter 槽位、checkpoint 根推导与完整 CLI 参数分组。
- miles/tinker/arguments.py (模块 网关参数; 类别 source; 类型 core-logic; 符号 add_tinker_arguments, add_argument, configure_tinker_args, _resolve_target_modules) : 新增的 Tinker 参数层与能力边界: 把 --tinker-* 注册集中到一处, 并把 attn/mlp/unembed 开关解析为 target_modules, 同时硬编码 Megatron 后端与 Qwen3 系列模型限制。
- examples/multi_lora/run_multi_tenant_example.py (模块 多租户客户端; 类别 source; 类型 core-logic; 符号 build_datums, run_client, main) : SDK 侧验收客户端: 用 api_key 作为租户标识, 单租户 / 多租户两种模式校验 loss 下降与各自 marker 的隔离性, 是网关「marker 记忆」契约的可执行定义。

- `examples/multi_lora/run_client_recipes.py` (模块 配方客户端; 类别 source; 类型 core-logic; 符号 `run_sft`, `run_rl`, `main`) : 上游 cookbook 验收客户端: 直接调用 `tinker-cookbook` 的 `sl_loop/rl_loop`, 把官方 recipe 能否跑通定义为网关 wire contract 的通过标准。
- `miles/utils/arguments.py` (模块 参数解析; 类别 source; 类型 core-logic; 符号 `parse_args`) : 被删减 19 行、只保留 3 行新增: Tinker 参数注册从公共解析函数中抽离, 改为调用 `add_tinker_arguments`, 是所有训练入口都会经过的共享路径。
- `serve_tinker.py` (模块 服务入口; 类别 source; 类型 dependency-wiring) : 网关进程的入口接线: 随参数层拆分同步调整导入与调用顺序 (+8/-7), 是示例 launcher 与后端之间唯一的连接点。
- `tests/fast/tinker/core/test_service.py` (模块 网关服务测试; 类别 test; 类型 test-coverage; 符号 `_optim_payload`, `test_resubmitted_seq_id_reuses_the_future_and_runs_once`, `test_create_model_is_two_phase`, `test_failed_slot_init_returns_the_slot`) : 体量最大的新增测试 (850 行), 把请求去重、两阶段建模型、槽位回收、barrier 顺序、checkpoint 版本发布、路径穿越拒绝、租户与失败隔离等网关语义固化成可执行断言。
- `tests/fast/tinker/harness.py` (模块 测试桩; 类别 test; 类型 test-coverage; 符号 `FakeBackend`, `_record`, `named`, `trainer_dead`) : 全部失败路径用例的基石: `FakeBackend` 记录调用并支持 `fail_next/fail_on` 注入, 复用 `write_checkpoint_dir` 真实落盘, 避免测试自造一套 IO 语义。
- `tests/e2e/lora/tinker_gateway.py` (模块 网关夹具; 类别 test; 类型 test-coverage; 符号 `prepare_gateway`, `_wait_for_gateway`, `running_gateway`) : e2e 夹具: `prepare_gateway` 拉镜像 / 准备环境、`_wait_for_gateway` 就绪探测、`running_gateway` 生命周期管理, 供 cookbook 与 multi-tenant 两条 8 卡用例复用。

关键符号: `ScriptArgs.post_init`, `serve`, `prepare`, `add_tinker_arguments`, `configure_tinker_args`, `_resolve_target_modules`, `build_datums`, `run_client`, `run_sft`, `run_rl`, `FakeBackend._record`, `FakeBackend.named`, `created_model`, `await_settled`, `prepare_gateway`, `running_gateway`

关键源码片段

`examples/multi_lora/serve_qwen3_30b_a3b_tinker.py`

本 PR 的主要交付物: Qwen3-30B-A3B Tinker 网关启动器, 定义 8 卡单节点的训练 / 采样切分、adapter 槽位、checkpoint 根推导与完整 CLI 参数分组。

```
@dataclass
class ScriptArgs(U.ExecuteTrainConfig):
    # run_id 每次调用由 U.create_run_id 生成, checkpoint 根目录固定为 save_dir/run_
    # id: 重启即换目录, 不会覆盖旧权重
    run_id: str = field(default_factory=U.create_run_id)

    hf_checkpoint: str | None = None
    model_type: str = "qwen3-30B-A3B"
    model_dir: str = "/root/models"
    save_dir: str | None = None
```

```

megatron_path: str = "/root/Megatron-LM"

# 一台 8 卡节点切成 4 卡训练 (TP2 + EP4) 与 4 卡采样, adapter 槽位池同时容纳 4 个租户
num_gpus_per_node: int = 8
actor_num_gpus: int = 4
rollout_num_gpus: int = 4
tp: int = 2
ep: int = 4

lora_rank: int = 32 # 槽位池上界; 单客户端 rank 由 SDK 传入, 不得超过它
lora_alpha: int = 64
n_adapters: int = 4
tinker_train_attn: bool = True
tinker_train_mlp: bool = True
tinker_train_unembed: bool = True # lm_head 也参与 LoRA, 示例的 marker 记忆任务依赖它

def __post_init__(self):
    # 未显式传参时从 output_dir 推导 checkpoint 根, 并把 HF 权重落到 model_dir
    if self.save_dir is None:
        self.save_dir = f"{self.output_dir}/checkpoints"
    if self.hf_checkpoint is None:
        self.hf_checkpoint = f"{self.model_dir}/Qwen3-30B-A3B"

@app.command()
@U.dataclass_cli
def serve(args: ScriptArgs):
    """Serve the Tinker gateway (idles until clients connect)."""
    ckpt_args = f"--hf-checkpoint {args.hf_checkpoint} --megatron-to-hf-mode bridge "

    lora_args = (
        f"--lora-rank {args.lora_rank} --lora-alpha {args.lora_alpha} --lora-dropout 0.0 "
        "--no-gradient-accumulation-fusion " # 槽位共享基座权重, 保持梯度路径显式可控
        f"--multi-lora-n-adapters {args.n_adapters} "
    )

    # checkpoint 根与 run_id 绑定, 每次 serve 得到独立的 tinker:// 命名空间
    tinker_args = f"--tinker-server-port {args.tinker_port} " f"--tinker-checkpoint-root {args.save_dir}/{args.run_id}"

    # 三组开关最终由 miles/tinker/arguments.py 的 _resolve_target_modules 展开成 target_modules
    for group in ("attn", "mlp", "unembed"):
        enabled = getattr(args, f"tinker_train_{group}")
        tinker_args += f"--{' ' if enabled else 'no-'}tinker-train-{group}"

    # 这里只是初始配置; 真实的 AdamParams 由每个 optim_step 请求携带
    optimizer_args = "--optimizer adam --lr 1e-4 "

```

```

perf_args = (
    f"--tensor-model-parallel-size {args.tp} --sequence-parallel "
    "--pipeline-model-parallel-size 1 --context-parallel-size 1 "
    f"--expert-model-parallel-size {args.ep} --expert-tensor-parallel-size 1 "
    "--use-dynamic-batch-size --max-tokens-per-gpu 8192 "
)
sglang_args = (
    f"--rollout-num-gpus-per-engine {args.rollout_num_gpus_per_engine} "
    f"--sglang-mem-fraction-static {args.sglang_mem_fraction_static} "
    "--sglang-lora-backend triton "
)
topology_args = (
    f"--actor-num-nodes 1 --actor-num-gpus-per-node {args.actor_num_gpus} "
    f"--rollout-num-gpus {args.rollout_num_gpus} "
)
misc_args = (
    "--attention-dropout 0.0 --hidden-dropout 0.0 "
    "--accumulate-allreduce-grads-in-fp32 --attention-softmax-in-fp32 --attention-backend flash "
)

train_args = (
    f"{ckpt_args} {lora_args} {tinker_args} {optimizer_args} "
    f"{perf_args} {sglang_args} {topology_args} {misc_args} {args.extra_args} "
)

# 复用统一训练入口承载网关进程，train_script 指向 serve_tinker.py
U.execute_train(
    train_args=train_args,
    config=args,
    num_gpus_per_node=args.num_gpus_per_node,
    megatron_model_type=args.model_type,
    train_script="serve_tinker.py",
    megatron_path=args.megatron_path,
)

```

miles/tinker/arguments.py

新增的 Tinker 参数层与能力边界：把 --tinker-* 注册集中到一处，并把 attn/mlp/unembed 开关解析为 target_modules，同时硬编码 Megatron 后端与 Qwen3 系列模型限制。

```
import argparse
```

```
from miles.utils.hf_config import load_hf_config
```

```
def add_tinker_arguments(parser):
```

```
    """把所有 --tinker-* 开关集中注册，供 miles/utils/arguments.py 的 parse_args 调用。"""
```

```
    group = parser.add_argument_group("Tinker")
```

```

def add_argument(name, **kwargs):
    return group.add_argument(f"--tinker-{name}", **kwargs)

add_argument("server-host", default="0.0.0.0")
add_argument("server-port", type=int, default=10613)
add_argument("base-model", help="网关对外宣告的模型名（默认取 --hf-checkpoint）")
add_argument("checkpoint-root", help="tinker:// 的 checkpoint 根目录（默认 <save>/tinker）")
# BooleanOptionalAction 同时派生 --tinker-train-attn 与 --no-tinker-train-attn
add_argument("train-attn", action=argparse.BooleanOptionalAction, default=True)
add_argument("train-mlp", action=argparse.BooleanOptionalAction, default=True)
add_argument("train-unembed", action=argparse.BooleanOptionalAction, default=True)
return parser

def configure_tinker_args(args):
    # Tinker 网关依赖 Megatron 的 MultiLoRA 槽位与分布式 checkpoint IO
    assert args.train_backend == "megatron", "Tinker requires the Megatron backend"
    # 统一由 train-attn/mlp/unembed 推导 target_modules，禁止与通用 LoRA 过滤参数混用
    assert (
        args.target_modules is None and args.exclude_modules is None
    ), "Tinker uses --tinker-train-attn/mlp/unembed; --target-modules and --exclude-modules are not supported"
    modules = _resolve_target_modules(
        load_hf_config(args.hf_checkpoint),
        train_attn=args.tinker_train_attn,
        train_mlp=args.tinker_train_mlp,
        train_unembed=args.tinker_train_unembed,
    )
    # 交给通用 LoRA 校验器，在 trainer / engine 初始化前完成解析与校验
    args.target_modules = ",".join(modules)

def _resolve_target_modules(hf_config, *, train_attn, train_mlp, train_unembed):
    # 每个新模型族都需要自己的 attention/MLP 完整映射，此处只覆盖 Qwen3 系列
    assert hf_config.model_type in (
        "qwen3",
        "qwen3_moe",
    ), f"Tinker target layout is not defined for model_type={hf_config.model_type!r}"
    modules = []
    if train_attn:
        modules.extend(("q_proj", "k_proj", "v_proj", "o_proj"))
    if train_mlp:
        modules.extend(("gate_proj", "up_proj", "down_proj"))
    if train_unembed:
        modules.append("lm_head") # unembedding 也要适配，示例的 marker 记忆依赖这一项
    assert modules, "Tinker requires at least one trainable LoRA module group"
    return modules

```

[examples/multi_lora/run_multi_tenant_example.py](#)

SDK 侧验收客户端：用 `api_key` 作为租户标识，单租户 / 多租户两种模式校验 loss 下降与各自 marker 的隔离性，是网关「marker 记忆」契约的可执行定义。

```
async def run_client(index: int, args) -> dict:
    marker = MARKERS[index]
    tag = f"client-{index}"
    # tml- 前缀是 SDK 强制要求；网关把 api_key 直接当成租户 id，用来隔离 future 与 adapter 槽位
    service = tinker.ServiceClient(base_url=args.base_url, api_key=f"tml-smoke-{tag}")
    training = await service.create_lora_training_client_async(base_model=args.base_model,
                                                             rank=args.lora_rank)
    tokenizer = training.get_tokenizer()

    datums = build_datums(tokenizer, marker)
    target_tokens = sum(datum.loss_fn_inputs["target_tokens"].shape[0] for datum in datums)

    losses = []
    for _ in range(args.steps):
        # 先并行发出两个请求，网关按 seq_id 排序并把 optim_step 当作 barrier，等前面所有
        # forward_backward 完成
        fb_future = await training.forward_backward_async(datums, loss_fn="cross_entropy")
        optim_future = await training.optim_step_async(types.AdamParams(learning_rate=args.lr))
        fb = await fb_future
        await optim_future
        losses.append(fb.metrics["loss:sum"] / target_tokens)

    # 保存后立刻用返回的 tinker:// 路径建采样客户端，验证「保存版本 → 采样解析」链路
    save_future = await training.save_weights_for_sampler_async(name="final")
    sampler_path = (await save_future).path
    sampler = await service.create_sampling_client_async(model_path=sampler_path)

    response = await sampler.sample_async(
        prompt=types.ModelInput.from_ints(tokenizer.encode(PROMPT)),
        num_samples=1,
        sampling_params=types.SamplingParams(max_tokens=args.max_tokens, temperature=0.0),
    )
    completion = tokenizer.decode(response.sequences[0].tokens)

    # 双重判据：loss 必须下降，且 greedy 输出必须出现本客户端自己的
    # marker，以此暴露跨租户串扰
    ok = losses[-1] < losses[0] and marker in completion
    print(
        f"[{tag}] {'PASS' if ok else 'FAIL'} loss {losses[0]:.3f} -> {losses[-1]:.3f}, "
        f"marker {marker!r}, completion {completion!r}"
    )
    return {"tag": tag, "ok": ok}

async def main(args):
    # --mode single 只跑一个租户（快速自检），--mode multi 并发跑多个租户（隔离性验收）
```

```

n_clients = 1 if args.mode == "single" else args.clients
assert n_clients <= len(MARKERS), f"at most {len(MARKERS)} clients"
results = await asyncio.gather(*(run_client(i, args) for i in range(n_clients)))
failed = [r["tag"] for r in results if not r["ok"]]
assert not failed, f"failed clients: {failed}"
print(f"all {n_clients} client(s) passed")

```

评论区精华

本 PR 的 review 讨论几乎为空：claude[bot] 只提示「This repository is configured for manual code reviews」，没有触发实际评审；yushengsu-thu（共同作者）给出 APPROVED，review_comments_count = 0，2792 行 / 40 个文件只有 1 个 commit。真正有信息量的是 issue 评论里的 CI 线程：作者连续 4 次 /rerun-test (test_tinker_multi_tenant_gateway.py、test_tinker_cookbook_gateway.py、test_qwen3_5_35b_a3b_lora_ci.py、test_nemotron_3_ultra_4layer_ci.py)，首轮四条全部在 39-48 秒内以「The run never started: resolving the test file's execution plan failed」失败，第二轮分别在 stage-c-8-gpu-h200/h100 上以 7m18s / 7m41s / 16m14s / 24m43s 通过；另有一条与 multi-lora 无关的既有失败 test_deepseek_v4_flash_4layer_ci.py (stage-c-4-gpu-h200, 20m11s)。此外 PR body 自己记录了若干设计取舍（IO 失败终止共享 cell、不做自动恢复、GPU 收敛验证延后），这些是理解实现边界的关键。

- 评审门禁：只有共同作者的 APPROVED，没有逐行讨论 (other): 2792 行 / 40 文件 / 1 个 commit 的变更在没有第三方逐行评审的情况下合入，批准来自协作方本人。
- 新增 e2e 用例首轮「执行计划解析失败」(testing): 失败发生在测试选择 / 执行计划解析阶段而非被测代码，属于 CI 侧问题；但暴露了新启用用例与门禁链路的稳定性依赖，需要重跑才能拿到有效结果。
- checkpoint/IO 失败即终止共享 trainer cell 的取舍 (design): 选择了 fail-fast: 任何租户的磁盘 / 导出失败都会终止整个网关，换取状态一致性；自动恢复被显式排除在本次范围外，多租户可用性风险留给后续迭代。
- Tinker 的能力边界被写死在参数层 (design): 边界清晰但扩展成本高：新增模型族或换后端必须改这段代码；示例 launcher 也据此把 tinker-train-attn/mlp/unembed 作为唯一开关面。

风险与影响

- 风险：1) 共享参数解析路径被改动：miles/utils/arguments.py 删除 19 行内联 Tinker 注册改调 add_tinker_arguments，这是所有训练入口共用的 parse_args 路径；若仍有脚本或测试依赖旧的内联参数位置 / 默认值，可能出现静默回归（本次提供的材料中未见针对该文件的独立测试证据）。2) 能力边界由 assert 硬编码：configure_tinker_args 断言 train_backend == megatron，_resolve_target_modules 只接受 model_type ∈ {qwen3, qwen3_moe}，换模型族或换后端会直接 AssertionError 而非可读错误；Tinker 与 --target-modules/--exclude-modules 互斥也不可绕过。3) 示例 launcher 的 run_id 由 U.create_run_id 每次调用生成，checkpoint 根 save_dir/run_id 随之变化：重启后旧的 tinker:// 路径不再可达，目录会持续累积；对示例是刻意的隔离，但用于长跑或生产需要显式固定 run_id/save_dir（能否用 CLI 覆盖 run_id 需按 U.dataclass_cli 的实际行为确认）。

- 4) 可用性策略激进: body 明确「Checkpoint load/save/export execution failures, including filesystem errors, terminate the shared trainer cell and server」, 即单租户的一次 IO 故障会终止整个网关, 且「Automatic training recovery/replay remains outside the current implementation」——多租户场景下这是最大的可用性风险。
- 5) 测试以 fake 为主: FakeBackend 不覆盖真实 Megatron/ 路由 / 引擎行为, body 亦承认该验证「does not validate forward/backward numerics, topology changes, MoE/TP/PPLayouts, server restart, or failure recovery」, 并记录了本地 launcher gate 362 passed / 30 failed / 122 errors。
- 6) CI 成本: 新增两条 8 卡 H200 用例 (7-8 分钟量级) 并挂 run-ci-multi-lora, 叠加 run-ci-image / bypass-fastfail 标签, 门禁执行时间与资源占用上升。
- 7) 说明与代码可能不一致: body 称「No new test files are added」, 而文件清单显示 tests/fast/tinker/** 与若干 e2e 文件均为新增, 描述疑似沿用了栈内早期版本, 需要以最终 tree 为准。
- 影响: 对使用者: multi-LoRA / Tinker 用户首次拿到可直接运行的参考配方和两个客户端——SDK marker 记忆 (单租户与并发租户) 与上游 cookbook 的 SFT/GRPO 验收, 正确性判据 (loss 下降 + 自己的 marker 被记住、跨租户互不串扰) 可原样复用。对系统: 源码侧集中在示例、miles/tinker/arguments.py 新增与 miles/utils/arguments.py、serve_tinker.py 的接线, 训练数值路径未改; 但参数层是公共入口, 任何以 Tinker 方式启动的训练脚本都受影响, 且 Tinker 与 Megatron 后端强绑定。对团队: 测试套件把网关的内部契约 (seq_id 排序、optim_step barrier、批次打包边界、checkpoint 版本发布、路径穿越拒绝、403/410 语义、core 层不 import miles) 固化成可执行的规格, 后续改网关会先撞测试; 新模型族需要在这里扩展 target modules 映射。对 CI/ 运维: 新增两条 8 卡 H200 用例并挂标签选择, GPU 排队与时长增加; checkpoint 根随 run_id 变化与「IO 失败终止共享 cell」两条语义需要运维侧明确接受。
 - 风险标记: 共享参数解析路径改动, 示例 run_id 每次生成, 仅支持 qwen3/qwen3_moe, IO 失败级联终止网关, GPU 端到端验证依赖重跑, 无逐行人工评审

关联脉络

- PR #2845 multi-lora v2 栈基 PR (标题未在上下文中给出): PR body 明确「Stacked on #2845」, 本 PR 的示例与测试建立在该栈的网关实现之上, 参数层与服务层的边界由它定义。
- PR #3198 multi-lora v2 栈内关联 PR (标题未在上下文中给出): body 称「the final #3198 tree exactly matches the reviewed implementation」, slot checkpoint 续训验证与广谱 GPU 选择都落在 #3198, 本 PR 的最终形态需与之一致。
- PR #3183 [AMD][LoRA] Add a Qwen3-4B LoRA RL launcher: 同样是「启动器脚本 + 记录型命令快照」的组合 (scripts/amd/run_qwen3_4b_lora.py、tests/snapshots/launch_scripts/**), 与本 PR 的 launcher/ 快照做法同源, 可作为写法参照。
- PR #3245 fix(megatron): respect --no-load-optim when resuming a LoRA adapter: 同属 LoRA 槽位 checkpoint 保存 / 恢复正确性主线; 本 PR 的 save/load slot 参数与 optimizer 状态断言与之呼应。

- PR #3083 fix(megatron): respect --no-save-optim when saving checkpoints: LoRA checkpoint 语义的另一半（保存侧），与本 PR 中 sampler 发布、checkpoint 版本化测试构成同一可靠性脉络。
- PR #3256 e2e: run the DSv4 megatron impl CI on the cudnn DSA backend: 本 PR 期间报错的 test_deepseek_v4_flash_4layer_ci.py (stage-c-4-gpu-h200, 20m11s) 正属于该 CUDA 用例线，失败原因很可能与其后端迁移相关而非本 PR 改动。