

PR #2841 完整报告

radixark/miles

[multi-lora v2 2/7] Per-batch loss selection and Tinker losses

合并时间: 2026-09-16 16:25

原文链接: <http://prhub.com.cn/radixark/miles/pull/2841>

执行摘要

- 一句话: 新增 Tinker 每批损失选择、5 种目标函数与 per-datum 输出
- 推荐动作: 建议精读。最值得关注的三处设计决策: (1) `tinker_losses.py` 的「服务端只做 raw sum、归一化交给客户端」模型, 以及 `loss_normalizer` 与执行 batch size 解耦的理由; (2) `_sum_loss_and_outputs` 在“无监督 token 的 microbatch”上用 `logits.sum(dtype=torch.float32) * 0` 保持计算图连通的 NaN 防御写法, 以及 per-datum 输出 detach 到 CPU 以规避 recompute 重复追加的设计; (3) `forward_only` 复用同一条 loss pass 而不是另起实现, 这是消除双实现漂移的关键取舍。阅读时建议同时对照 review 中“零掩码测试无法验证目标函数”与“per_datum 命名空间污染”两条评论, 它们分别解释了最终测试矩阵与 `loss.py` 守卫的由来; 并把 nanjiangwill 关于 O(KN) 全池同步的评论作为独立待办跟踪。

功能与动机

Tinker 协议允许客户端按 work unit 指定目标函数, 而 miles 原有实现只有一个由 `args.loss_type` 决定的全局损失, 无法在同一 trainer 上服务不同 `loss_fn` 的请求。PR body 说明: 需要 “Select loss_fn and loss_fn_config per work unit”, 并提供 “cross-entropy, importance-sampling, PPO, CISPO, and DRO losses”; 归一化由客户端负责, “client loss inputs provide normalization, including custom-loss gradients represented as weights = -dL/dlogprob”; 同时要避免全局收集器带来的重复输出——“There is no module-global output collector, so loss recomputation cannot append duplicate outputs”。配套地, “Optimizer requests also apply grad_clip_norm per slot”, 并且 “loss_normalizer is separate from the execution batch size, so Tinker accumulation retains raw sums”。本 PR 与 @yushengsu-thu 共同完成, 堆叠在 #2840 之上。

实现拆解

1. 新增 Tinker 损失族与注册表 (`miles/backends/training_utils/loss_hub/tinker_losses.py`, 新增 163 行) - 5 个目标函数: `cross_entropy_loss_function`、`importance_sampling_loss_function`、`ppo_loss_function`、`cispo_loss_function`、`dro_loss_function`, 统一签名 (`args, batch, logits, sum_of_sample_mean`) -> (`loss, log`)。
 - 共享辅助: `_target_logprobs` 把显式 `target_tokens` 拼到序列尾部当作 gather label; `_as_tensor_like` 统一 dtype/device; `_response_masks` 保证 DP padding datum 的全零掩码不进入目标; `_sum_loss_and_outputs` 做 raw sum 聚合并产出 per-datum 输出。 -

默认超参常量 PPO_DEFAULTS / CISPO_DEFAULTS / DRO_DEFAULTS, 可被 batch["loss_fn_config"] 逐 batch 覆盖; TINKER_LOSS_FUNCTIONS 是客户端字符串名到实现的注册表。- 影响: 目标函数从「进程级单例」变成「每 work unit 可换」, 这是后续协议层 / 网关 (#2844、#2845) 能按请求分派的前提。

2. 每批损失选择与归一化契约 (miles/backends/training_utils/loss.py、miles/backends/training_utils/loss_hub/losses.py) - get_loss_function(args, loss_fn=None) 在 match args.loss_type 之前短路: 只要 loss_fn 非空就返回 TINKER_LOSS_FUNCTIONS[loss_fn]。- loss_function 改为 get_loss_function(args, batch.get("loss_fn")), 其余调用点保持兼容。- 原来的 global_batch_size = 1 (multi-LoRA 特判) 被重命名为 loss_normalizer: multi-LoRA 下仍为 1, 非 multi-LoRA 回落到 global_batch_size, 语义上把“梯度缓冲权重”和“损失归一化分母”两件事拆开。- 目标函数返回的 per_datum 被 log.pop("per_datum", None) if batch.get("loss_fn") is not None else None 摘出, 并只在 Tinker 路径注入 logging dict, 避免污染原生 / 自定义 loss 的指标命名空间。
3. 训练期数据契约扩展 (miles/backends/megatron_utils/model.py) - run_forward_backward_pass(..., forward_only=False) 新增参数, 前向 -only 时用 torch.no_grad() 包裹 forward_backward_func, 并透传 forward_only 给 Megatron。- forward_step 的 get_batch key 白名单新增 loss_weights、target_tokens、sample_indices, 这是 per-datum 损失能读到权重与样本索引的唯一入口。- 同一 hunk 顺带把 rollout 前向 forward_only 里的 assert not return_schedule_plan 删掉、并简化了 docstring, 属于与本主题无关的约束放宽。- 配套: miles/ray/rollout/train_data_conversion.py 的 shard 白名单必须同步补齐, 否则 DP 切分后这些向量会静默消失, 只会在 trainer 内部才炸——test_delayed_dp_split_preserves_the_tinker_loss_vectors 正是这条守卫。
4. LoRA 执行器与 optim_step 改造 (miles/backends/megatron_utils/lora/model.py、lora/actor.py、lora/optimizer.py、miles/ray/train/group.py) - run_forward_backward(..., *, forward_only=False): forward_only 时跳过 reset_grad_metadata_keep_grads, 把 per_datum 从各 microbatch 结果里聚合出来, 返回 {"metrics": ..., "per_datum": ...} (非 PP 末段返回空 metrics + per_datum)。- MultiLoRATrainRayActor.forward_only_logprobs 被 forward_only 取代, 直接复用 run_forward_backward(..., forward_only=True); miles/ray/train/group.py 的 _execute_slots("forward_only", ...) 同步改名, RPC 名字面量变化需要网关侧对齐。- optim_step 去掉 args 与 clip_grad, per-slot 的 grad_clip_norm 改由槽位优化器在步进时应用 (PR body: "Optimizer requests also apply grad_clip_norm per slot")。
5. 测试与验证配套 - 新增 tests/fast/backends/training_utils/loss/test_tinker_loss_outputs.py (142 行): 验证 direct/recomputed 两种模式下 per-datum 输出互不串扰、不重复追加、张量 detach、零掩码 datum 对所有 5 个目标都不产生梯度, 并补充 7 组非零目标 / 梯度用例 (IS、PPO 默认与覆盖、CISPO 默认与覆盖、DRO 默认与覆盖)。- 新增 tests/fast/backends/training_utils/loss/test_tinker_explicit_targets.py: 验证 target 覆盖序列作为 gather label, 以及 shifted target 退化为 next-token 打分。- tests/fast/backends/megatron_utils/test_step_adapter_slots.py、tests/fast/backends/training_utils/loss/test_training_logprob_reuse.py 随签名调整同步

更新。 - PR body 声明 `pre-commit run --all-files` 与增量 `git diff --check` 通过, CPU/GPU CI 尚在排队 (“Fresh CPU/GPU CI is pending”), 验证与限制记录在 #2846、#3198。

关键文件:

- `miles/backends/training_utils/loss_hub/tinker_losses.py` (模块 损失函数; 类别 source; 类型 core-logic; 符号 `_target_logprobs`, `_as_tensor_like`, `_response_masks`, `_sum_loss_and_outputs`): 本 PR 的核心新增文件: 5 个 Tinker 目标函数、共享辅助 (`_target_logprobs/_response_masks/_sum_loss_and_outputs`) 与 `TINKER_LOSS_FUNCTIONS` 注册表, 定义了服务端 raw-sum、客户端归一化的损失契约。
- `miles/backends/megatron_utils/model.py` (模块 训练引擎; 类别 source; 类型 data-contract; 符号 `run_forward_backward_pass`): 训练 / 前向的公共 pipeline 入口: `run_forward_backward_pass` 新增 `forward_only` 参数与 `no_grad` 分支, `get_batch` 白名单新增 `loss_weights/target_tokens/sample_indices`, 是 per-datum 损失可用的数据契约来源。
- `miles/backends/training_utils/loss.py` (模块 损失分发; 类别 source; 类型 core-logic; 符号 `loss_function`): 损失分发与归一化的中枢: `get_loss_function` 改为按 `batch["loss_fn"]` 选择, `global_batch_size` 抽象为 `loss_normalizer`, 并只在 Tinker 路径摘取 per_datum。
- `miles/backends/megatron_utils/lora/model.py` (模块 适配器训练; 类别 source; 类型 data-contract; 符号 `run_forward_backward`, `optim_step`): multi-LoRA 执行入口: `run_forward_backward` 增加 `forward_only` 关键字参数、跳过梯度元数据重置、聚合 per_datum 并改为返回 `{metrics, per_datum}`; `optim_step` 去掉 `args/clip_grad`。
- `miles/backends/megatron_utils/lora/actor.py` (模块 适配器训练; 类别 source; 类型 core-logic; 符号 `forward_backward`, `optim_step`, `forward_only`, `load_slot`): RPC 面变更: `forward_only_logprobs` 被 `forward_only` 取代并复用 `run_forward_backward`, `optim_step` 签名与返回值类型调整, 直接影响网关调用方。
- `tests/fast/backends/training_utils/loss/test_tinker_loss_outputs.py` (模块 损失测试; 类别 test; 类型 test-coverage; 符号 `test_loss_passes_return_independent_detached_outputs`, `test_a_zero_loss_mask_removes_the_datum_from_every_objective`, `test_nonzero_objectives_and_gradients`): 本 PR 的核心测试: 覆盖重算不重复输出、detach/ 隔离、零掩码无梯度, 以及 7 组非零目标与梯度用例 (对应 reviewer 要求补齐的验证)。
- `miles/backends/training_utils/loss_hub/losses.py` (模块 损失注册; 类别 source; 类型 core-logic; 符号 `get_loss_function`): `get_loss_function` 增加 `loss_fn` 参数并在 `match args.loss_type` 之前短路到 `TINKER_LOSS_FUNCTIONS`, 是“每批损失选择”的接入口。
- `tests/fast/backends/training_utils/loss/test_tinker_explicit_targets.py` (模块 目标对齐测试; 类别 test; 类型 test-coverage; 符号 `_naive_compute_log_probs`, `_target_logprobs`, `test_targets_replace_the_sequence_as_gather_labels`, `test_shifted_targets_degenerate_to_next_token_scoring`): 锁定 `target_tokens` 作为显式 gather label 的语义 (含 shifted target 退化为 next-token 打分), 是 Tinker 客户端契约的回归护栏。

- miles/ray/rollout/train_data_conversion.py (模块 数据切分; 类别 source; 类型 data-contract) : DP 切分的 shard key 白名单需与 get_batch 同步; 漏写会让 loss_weights/advantages/rollout_log_probs 静默消失, 只在 trainer 内部报错。
- miles/backends/megatron_utils/lora/optimizer.py (模块 适配器优化; 类别 source; 类型 configuration) : 配合 optim_step 瘦身, 把 grad_clip_norm 改为按 slot 应用, 是归一化职责从训练端迁移到优化器步骤的一部分。

关键符号: cross_entropy_loss_function, importance_sampling_loss_function, ppo_loss_function, cispo_loss_function, dro_loss_function, _target_logprobs, _response_masks, _sum_loss_and_outputs, get_loss_function, loss_function, run_forward_backward_pass, run_forward_backward, optim_step, forward_only

关键源码片段

miles/backends/training_utils/loss_hub/tinker_losses.py

本 PR 的核心新增文件: 5 个 Tinker 目标函数、共享辅助 (_target_logprobs/_response_masks/_sum_loss_and_outputs) 与 TINKER_LOSS_FUNCTIONS 注册表, 定义了服务端 raw-sum、客户端归一化的损失契约。

```
# miles/backends/training_utils/loss_hub/tinker_losses.py
# Tinker 损失族: 每个 work unit 由客户端指定 loss_fn, 训练端只累加 raw sum,
# 归一化由客户端在 loss_weights 里给出 (自定义损失的梯度表示为 weights = -dL/dlogprob)。
```

```
PPO_DEFAULTS = {"clip_low_threshold": 0.8, "clip_high_threshold": 1.2}
CISPO_DEFAULTS = {"clip_low_threshold": 0.0, "clip_high_threshold": 4.0}
DRO_DEFAULTS = {"beta": 0.05}
```

```
def _target_logprobs(args: Namespace, batch: RolloutBatch, logits: torch.Tensor) -> list[torch.Tensor]:
```

```
    # Tinker 的 target 是显式标签: 用它替换序列 response 区段的 token,
    # 于是 gather 的 label 不再等于输入 token 本身 (rl_loop 风格)。
    label_tokens = [
        torch.cat([tokens[: len(tokens) - len(targets)], _as_tensor_like(targets, tokens)])
        for tokens, targets in zip(batch["unconcat_tokens"], batch["target_tokens"], strict=True)
    ]
    outputs = get_log_probs_and_entropy(
        logits,
        args=args,
        unconcat_tokens=label_tokens,
        total_lengths=batch["total_lengths"],
        response_lengths=batch["response_lengths"],
        with_entropy=False,
        max_seq_lens=batch.get("max_seq_lens", None),
    )
    return outputs["log_probs"]
```

```
def _sum_loss_and_outputs(batch, logits, log_probs, per_datum_losses):
```

```

if per_datum_losses:
    # 逐 datum 求和后直接相加：服务端不做除以 batch size 的归一化
    loss = torch.stack(per_datum_losses).sum()
else:
    # 没有监督 token 的 microbatch 也要让计算图保持连通；
    # fp32 求和避免 fp16 溢出成 inf, 0 * inf 会变成 nan
    loss = logits.sum(dtype=torch.float32) * 0
per_datum = [
    # 输出一律 detach 并搬到 CPU：它只用于回传给客户端，不参与反传
    {"sample_index": index, "logprobs": log_prob.detach().cpu(), "loss": sample_loss.detach().cpu()}
    for index, log_prob, sample_loss in zip(batch["sample_indices"], log_probs, per_datum_losses, strict=True)
]
return loss, {"loss": loss.detach(), "per_datum": per_datum}

def ppo_loss_function(args, batch, logits, sum_of_sample_mean):
    # 裁剪阈值允许被客户端的 loss_fn_config 逐 batch 覆盖
    config = batch.get("loss_fn_config") or {}
    clip_low = config.get("clip_low_threshold", PPO_DEFAULTS["clip_low_threshold"])
    clip_high = config.get("clip_high_threshold", PPO_DEFAULTS["clip_high_threshold"])
    log_probs = _target_logprobs(args, batch, logits)
    per_datum_losses = []
    for log_prob, sampling_log_prob, advantage, mask in zip(
        log_probs, batch["rollout_log_probs"], batch["advantages"], _response_masks(batch, log_probs), strict=True
    ):
        ratio = torch.exp(log_prob - _as_tensor_like(sampling_log_prob, log_prob))
        advantages = _as_tensor_like(advantage, log_prob)
        objective = torch.minimum(ratio * advantages, torch.clamp(ratio, clip_low, clip_high) * advantages)
        per_datum_losses.append(-(objective * mask).sum())
    return _sum_loss_and_outputs(batch, logits, log_probs, per_datum_losses)

# 客户端 loss_fn 字符串 -> 实现；loss.py 通过 batch["loss_fn"] 查这张表
TINKER_LOSS_FUNCTIONS = {
    "cross_entropy": cross_entropy_loss_function,
    "importance_sampling": importance_sampling_loss_function,
    "ppo": ppo_loss_function,
    "cispo": cispo_loss_function,
    "dro": dro_loss_function,
}

```

miles/backends/training_utils/loss.py

损失分发与归一化的中枢：get_loss_function 改为按 batch["loss_fn"] 选择，global_batch_size 抽象为 loss_normalizer，并只在 Tinker 路径摘取 per_datum。

```

# miles/backends/training_utils/loss.py —— loss_function 的核心分支
# 损失函数改为逐 batch 选择：同一个 trainer 可服务不同 loss_fn 的 work unit
func = get_loss_function(args, batch.get("loss_fn"))

if args.recompute_loss_function:
    # checkpoint 重算时只重跑损失，不会重复产出 per-datum 结果
    loss, log = checkpoint(func, args, batch, logits, sum_of_sample_mean)
else:
    loss, log = func(args, batch, logits, sum_of_sample_mean)

# 保持各 rank 的计算图连通，避免 CP/allgather 下反向不一致导致挂起
if parallel_state.cp.size > 1 and args.allgather_cp:
    loss = loss + 0 * logits.sum(dtype=torch.float32)

if num_rollouts is not None:
    global_batch_size = num_rollouts
else:
    assert args.use_dynamic_global_batch_size == ("dynamic_global_batch_size" in batch)
    global_batch_size = batch.get("dynamic_global_batch_size", args.global_batch_size)
# Tinker 损失已带客户端归一化；multi-LoRA 下样本以权重 1 进入梯度缓冲，
# per-adapter 归一化推迟到 optimizer step，所以这里不能再除一次
loss_normalizer = 1 if is_multi_lora_enabled(args) else global_batch_size
if not args.calculate_per_token_loss:
    if apply_megatron_loss_scaling:
        loss_parallel_size = (
            parallel_state.intra_dp.size
            if args.true_on_policy_mode and parallel_state.is_ulysses_cp
            else parallel_state.intra_dp_cp.size
        )
        loss = loss * num_microbatches / loss_normalizer * loss_parallel_size
    else:
        loss = loss / loss_normalizer * parallel_state.intra_dp.size
else:
    if apply_megatron_loss_scaling:
        loss = loss * parallel_state.cp.size

# per_datum 只属于 Tinker 协议；原生损失与 --custom-loss-function-path
# 的指标命名空间不预留这个键，因此只在 Tinker 路径上摘取
per_datum = log.pop("per_datum", None) if batch.get("loss_fn") is not None else None
return (
    loss,
    torch.tensor(num_tokens if args.calculate_per_token_loss else 1, device=logits.device),
    {
        **({"per_datum": per_datum} if per_datum is not None else {}),
        "keys": list(log.keys()),
        "values": torch.tensor(
            [num_samples if not args.calculate_per_token_loss else num_tokens] + list(log.values()),
            device=logits.device,
        ),
    },
)

```

```
    },  
)
```

miles/backends/megatron_utils/lora/model.py

multi-LoRA 执行入口：run_forward_backward 增加 forward_only 关键字参数、跳过梯度元数据重置、聚合 per_datum 并改为返回 {metrics, per_datum}; optim_step 去掉 args/clip_grad。

```
# miles/backends/megatron_utils/lora/model.py  
# Tinker 的 forward() 与 forward_backward() 共用同一条 loss pass  
def run_forward_backward(  
    args,  
    batch_id: int,  
    model: Sequence[DDP],  
    rollout_data: RolloutBatch,  
    *,  
    forward_only: bool = False,  
) -> dict:  
    data_iterator, num_microbatches = get_data_iterator(args, model, rollout_data)  
    assert len(num_microbatches) == 1, "a work unit is a single forward/backward pass"  
    # 模型切 chunk、train() 模式切换等准备步骤与训练态一致  
    setup_train_iteration_config(args, model, None, disable_optimizer=True)  
    if not forward_only:  
        # forward() 不产生梯度，因此不必保留上一轮的梯度元数据  
        reset_grad_metadata_keep_grads(model)  
    dumper_phase_util = DumperMegatronUtil(args, model, DumperPhase.FWD_BWD, rollout_id=  
batch_id)  
    losses_reduced = run_forward_backward_pass(  
        args,  
        dumper_phase_util,  
        data_iterator,  
        model,  
        num_microbatches[0],  
        num_rollouts=None,  
        forward_only=forward_only,  
    )  
    # per-datum 输出随 microbatch 结果一起回流，不再有模块级全局收集器，  
    # 因此 recompute_loss_function 重算损失时不会把同一 datum 追加两次  
    per_datum_outputs = [output for microbatch in losses_reduced for output in microbatch["per_  
datum"]]  
    dumper_phase_util.finalize(model)  
    if get_parallel_state().is_pp_last_stage:  
        return {"metrics": aggregate_train_losses(losses_reduced, None), "per_datum": per_datum_  
outputs}  
    return {"metrics": {}, "per_datum": per_datum_outputs}  
  
def optim_step(slot_optimizers, adam_params_by_slot):  
    # batch size 为 1：梯度按累积值直接步进，归一化由客户端损失权重承担，
```

```
# 每个 slot 的 grad_clip_norm 在槽位优化器内部应用
stepped = {slot: slot_optimizers[slot] for slot in adam_params_by_slot}
return step_slot_optimizers(stepped, adam_params_by_slot)
```

评论区精华

本 PR 共 5 条 review comment、7 条 review，实际技术交锋来自 nanjiangwill 的 4 条评论，yushengsu-thu 最终 APPROVED。

1) 测试有效性（已被采纳修复）。nanjiangwill 在 `test_tinker_loss_outputs.py:55` 指出早期的参数化用例“cannot validate the four new objectives because every result is multiplied by an all-zero mask; even an implementation that always returns zero passes”，要求补齐 IS/PPO/CISPO/DRO 的非零 loss 与梯度、包含默认与覆盖的 clip/beta。最终版本新增了 `test_nonzero_objectives_and_gradients` 的 7 组参数化用例，给出逐 token 期望值与梯度断言，问题闭环。

2) 指标命名空间污染（已被采纳修复）。针对 `loss.py` 的 `per_datum = log.pop("per_datum", None)`，他指出该分支“also runs for native and custom miles losses, where batch["loss_fn"] is None ... an existing metric with this name now disappears from logging”，建议只在 Tinker 路径摘取。最终实现改为 `if batch.get("loss_fn") is not None else None`，并在 `run_forward_backward` 里用 `log.pop("per_datum", None)` 做双重防护。

3) per-datum schema 的可扩展性（非阻塞、未落代码）。在 `tinker_losses.py:59` 他提醒：“future client-side losses may also request multi-target logprobs, values, or entropy ... just avoid treating {sample_index, logprobs, loss} as a closed schema”。结论是当前实现不改，但后续客户端损失上线时要预留字段扩展。

4) 优化器步进性能（本 PR 未解决）。在 `miles/backends/megatron_utils/lora/executor.py` 上他指出：“With DP > 1, this loop turns one optimizer barrier into K full-pool synchronizations ... Stepping K of N slots therefore moves O(KN) adapter parameters—O(N²) when K=N”，建议把 per-slot clip 值合并进一次 `step_adapter_slots` 调用，只做一次 outcome merge 与参数 all-gather。该反馈对应到本 PR 的 `optim_step` 瘦身（去掉 `args/clip_grad`、per-slot `grad_clip_norm`），但 O(KN) 的同步模式在本 PR 的改动里看不到收敛。

5) yushengsu-thu 在 `tinker_losses.py` 上留了一条“Suggestion: <https://github.com/radixark/miles/pull/3145>”的跟进指向，未在本 PR 展开。

- 零掩码参数化测试无法验证 IS/PPO/CISPO/DRO 目标函数 (testing): 已采纳: 新增 `test_nonzero_objectives_and_gradients`, 7 组参数化用例给出逐 token 期望值与梯度断言 (is / ppo-default / ppo-override / cispo-default / cispo-override / dro-default / dro-override)。
- `per_datum` 摘取会删除原生 / 自定义损失的同名指标 (correctness): 已采纳: 改为 `per_datum = log.pop("per_datum", None) if batch.get("loss_fn") is not None else None`, 并在 `run_forward_backward` 中再次用 `log.pop` 兜底。

- per-datum 输出 schema 的可扩展性 (design): 非阻塞: 当前仅保留 sample_index/logprobs/loss, 后续客户端损失上线时需按该建议扩展字段而不破坏现有消费方。
- DP > 1 时逐 slot 步进造成 O(KN) 参数搬运与全池同步 (performance): 本 PR 未收敛: optim_step 已去掉 args/clip_grad 并把裁剪下沉到 slot 优化器, 但逐 slot 同步模式仍存在, 需要后续按该建议合并调用。
- 跟进建议指向 PR#3145 (other): 作为后续跟进记录, 本 PR 内无对应代码改动; 上下文未提供该 PR 标题与内容, 无法判断具体后续范围。

风险与影响

- 风险:
 - 归一化语义变化 (回归风险, 最高): loss.py 把 global_batch_size 替换为 loss_normalizer, multi-LoRA 下恒为 1, 即训练端只累加客户端给的 raw sum。如果客户端对某个 loss_fn 没有自带归一化 (例如直接传 advantage、或自定义损失只给 weights = -dL/dlogprob 而没缩放), 梯度尺度会整体偏移但不会报错, 属于静默失配; PR body 也承认 CI 尚待执行, 建议在 GPU CI 上做一次 loss 尺度对照 (与单卡小批量手算比对)。
 - per_datum 的主机内存与序列化开销: _sum_loss_and_outputs 对每个 datum 都执行 log_prob.detach().cpu() 与 sample_loss.detach().cpu(), 且在 lora/model.py 中是在 is_pp_last_stage 判断之前就做全量列表推导, 非末段同样构造 (随后被丢弃)。长序列 / 大 batch 下这部分 CPU 拷贝与后续 RPC 序列化会成为新开销, 需要关注。
 - PP 非末段路径缺乏测试覆盖: run_forward_backward 返回 {"metrics": {}, "per_datum": per_datum_outputs} 的分支里, per_datum_outputs 直接迭代 losses_reduced 取 microbatch["per_datum"]; 非末段的 losses_reduced 结构与末段并不相同, 而现有 fast 测试都是单 stage, 建议在 TP/PP>1 的 GPU CI 或 e2e 上确认该分支不会因结构差异报错。
 - target_tokens 长度契约无断言: _target_logprobs 用 tokens[: len(tokens) - len(targets)] 拼接, 隐含 len(targets) == response_lengths; 长度不匹配时不会抛错, 只会让 gather 位置整体偏移, 建议补断言或校验。
 - 顺带删除的断言: miles/backends/megatron_utils/model.py 中 rollout 前向 forward_only 的 assert not return_schedule_plan 被移除, 属于与本 PR 主题无关的约束放宽, review 中未见讨论, 建议确认是否有意为之。
 - 优化器步进性能隐患: nanjiangwill 指出的 K 次 step_adapter_slots 触发 O(KN) 参数 all-gather / 全池同步问题在本 PR 未闭环, 在 DP > 1 且多 slot 并发步进时会放大。
 - 耦合与兼容性: actor RPC 名称 forward_only_logprobs -> forward_only 是字面量级的接口变更, 任何仍按旧名调用的网关 / 脚本都会直接失败; optim_step 去掉 args 后, per-slot clip 语义依赖优化器内部实现, 需要与其他 multi-LoRA 分片 (3/7 的槽位检查点、优化器状态) 保持同步。
- 影响:
 - 对用户: Tinker 客户端现在可以按 work unit 选择 5 种目标函数并通过 forward() 拿到逐 datum 的 sample_index / logprobs / loss, 为 rl_loop 风格的显式 target、自定义

损失 (weights = -dL/dlogprob) 打开通路; 原生 miles RL/SFT 用户在 batch["loss_fn"] 为 None 时走旧路径, 理论上无行为变化 (per_datum 摘取有守卫), 但 multi-LoRA 的归一化契约变化对所有使用该链路的实验都生效。

- 对系统: loss_function 的返回契约从 {keys, values} 扩展出可选的 per_datum; Megatron 训练循环新增 forward_only 参数与 torch.no_grad() 分支; LoRA 执行器新增一条与 forward_backward 共用 loss pass 的推理路径, 减少了此前 compute_log_prob 与训练 loss 两条实现漂移的可能。
- 对团队 / 后续演进: 本 PR 是 7 篇系列第 2 篇, 3/7~7/7 (槽位检查点、快照发布、协议层、网关 runtime、示例与测试) 都依赖这里的 loss_fn 分派、loss_normalizer、per_datum 结构与 actor 方法名; review 中暴露的优化器 all-gather 复杂度问题会成为后续优化器步骤统一改造的输入。
- 风险标记: 核心训练路径变更, 归一化契约调整, 新增 no_grad 前向路径, PP 多段路径缺测试, per_datum 主机内存开销, 优化器逐 slot 步进性能隐患, CI 尚未验证

关联脉络

- PR #2845 [multi-lora v2 6/7] Tinker gateway runtime and entry point: 同一系列下一篇, 网关 runtime 依赖本 PR 定下的 per_datum 输出结构与 forward_only/optimize_step 的 actor 接口。
- PR #2844 [multi-lora v2 5/7] Tinker gateway protocol face: 协议层按 work unit 组装 loss_fn / loss_fn_config 与 sample_index, 正是本 PR 损失选择契约的消费方。
- PR #2842 [multi-lora v2 3/7] Per-slot checkpoints with optimizer state: 同为 slot 优化器线路: 本 PR 调整 optimize_step 与 per-slot grad_clip_norm, 与本系列检查点 / 优化器状态改动直接相邻。
- PR #3245 fix(megatron): respect --no-load-optim when resuming a LoRA adapter: 同样涉及 LoRA 优化器状态与恢复语义, 与本 PR 的 optimize_step 归一化 / 裁剪职责调整属于同一模块的演进。
- PR #3083 fix(megatron): respect --no-save-optim when saving checkpoints: LoRA 检查点与优化器状态保存的历史修复, 说明该线路对优化器状态语义变更敏感, 本 PR 的 grad_clip_norm 下沉需与其保持一致。