PR #2122 完整报告

radixark/miles

[tml] Inkling native LoRA support

合并时间：2026-08-04 05:15

原文链接：http://prhub.com.cn/radixark/miles/pull/2122

执行摘要

- 一句话：Inkling 原生 LoRA 支持：adapter-only GRPO 全链路打通
- 推荐动作：值得精读。重点关注三类设计决策： 1) 插件式 LoRA 如何绕过 Megatron-Bridge 的 PEFT 集成，通过参数标签 + 显式梯度归约保证分布式正确性； 2) _pp_assemble_full_adapter 与 _repack_onto_fresh_storage 解决的多维并行 + CUDA IPC 工程问题； 3) SGLang 侧 lora_target_modules=["all"] 与 virtual-experts 的搭配。建议后续跟进 EP>TP 时 native shard 的加载验证、non-colocate 引擎的 LoRA 同步，以及 42/66 层真实配置的 CI 覆盖。

功能与动机

PR body 声明：Stacked on #1683（Inkling 模型 + 全参 RL），本 PR 在其上增加 LoRA 层，实现 adapter-only GRPO，复用同一套后端与并行堆栈。核心动机是训练成本：全参模式下每步都要把所有权重同步给 rollout 引擎（49.4s），而 LoRA 只需同步小型 adapter（2.5s），单步耗时降到全参的约 85%；同时遵循 Inkling 官方发布的 LoRA schema（r=32 all-linear），保证 adapter 可直接用于 serving（引擎侧原生 triton 后端 + virtual experts）。

实现拆解

实现按 5 个步骤拆解：

新增 miles_plugins/models/inkling/lora.py（759 行）。InklingLoRAAdapter 是纯参数容器，sharded_state_dict() 返回空字典，使 adapter 对 Megatron dist-checkpointing 完全不可见；每个 adapter 记录 load_meta（TP/EP rank、头维度、local intermediate 等导出元数据）。attention 因 MLA 结构拆 wq/wk/wv/wr 四组 A/B，并挂到 linear_qkv/linear_proj 的 forward 上叠加 delta，RMSNorm 已融合进 TELayerNormColumnParallelLinear，forward 里用 _rmsnorm 重算 norm 再喂 LoRA 输入；dense MLP 的 fc1/fc2 各一组 A/B，同样处理 fused layer-norm 与 sequence-parallel gather/reduce；routed experts 的 w1/w3/w2 三组参数中 w1_A/w3_A 共享输入投影，B 是三维参数 (num_local_experts, moe_intermediate, rank)，用 _grouped_linear（CUDA 走 F.grouped_mm，CPU 回退逐专家 linear）按 tokens_per_expert 做 grouped GEMM，并断言 expert_tensor_parallel_size == 1；shared experts 采用多专家共享 A、各专家独立 B 的 shared-outer 结构。参数约定：B 零初始化、A xavier，TP 复制参数标 _lora_grad_sum_group="tp"，EP 复制参数标 "ep" 且 allreduce=False。model.py 在 LoRA + Inkling 时用 wrap_model_provider_with_inkling_lora 包装 provider，initialize_model_and_optimizer 里

支持 --lora-adapter-path 热启动（load_inkling_lora_adapter +optimizer.reload_model_params() 刷新 fp32 master，避免首步 optimizer 覆盖刚写入的 adapter 值）。

lora_utils.py 新增 reduce_marked_lora_grads：在 finalize_model_grads 之前，按 _lora_grad_sum_group 标签把 "tp" 组、"ep" 组内各 rank 的 partial grad 做 all_reduce SUM（按 dtype 扁平化一次通信），随后才走常规 DDP 的 DP reduce-scatter。参数列表以 id(model[0]) 为 key 缓存，避免每步全模型扫描。该函数挂在 model.py::finalize_model_grads_with_empty_cache 中 finalize_model_grads 调用之前。

update_weight_from_tensor.py 是性能与正确性的关键：_pp_assemble_full_adapter 解决 exporter 的 bridge 只 gather TP/EP、不 gather PP 的问题，PP>1 时按 PP 组 all_gather_object 元数据 + 按 dtype 扁平 broadcast 完成全量组装；skip_base_sync 条件扩展 colocate_base_persistent（colocate 且未 offload_rollout 时 base 常驻引擎内存，可跳过 base 重传）；_repack_onto_fresh_storage 解决 torch_memory_saver.region() 的 cuMem MemPool 内存无法经 legacy CUDA IPC 导出的问题（offload_train 时把 adapter 张量拷到普通 caching-allocator 内存，按 (dtype, device) 合并成少量扁平 buffer）；同步结束后显式 ipc_collect() + empty_cache() 防 IPC 引用累积。sglang_engine.py 侧 load_lora_adapter_from_tensors 扩展 serialized_tensors 整包传输（与原 per-TP-rank 的 serialized_named_tensors 互斥二选一），Inkling checkpoint 时 lora_target_modules = ["all"] 交给 SGLang 自动识别模块名，dummy base load 时跳过启动 lora_paths 依赖 weight-sync。

scripts/run_inkling.py 新增 train_mode: "full" | "lora"，lora 模式拼装 --lora-rank 32 --lora-alpha 32 --target-modules all-linear --experts-shared-outer-loras --sglang-lora-backend triton --sglang-lora-use-virtual-experts --sglang-max-loras-per-batch 1 等参数，并设置 --sglang-ep-size rollout_num_gpus_per_engine；fully_async 断言仅支持 full。默认 lr 按模式区分（lora 5e-6，文档强调实际训练用 2e-4 才能让零初始化的 B 快速积累可见 delta-W）。utils/lora.py 新增 lora_rollout_enabled = is_lora_enabled and not lora_train_only，统一门控引擎侧 enable_lora、per-request lora_path 与 adapter 权重同步；actor.py 的 sleep/wake（torch_memory_saver tag）与更新器 is_lora 都改用它。arguments.py 新增 --lora-train-only，并修复 experts_shared_outer_loras 与 sglang_experts_shared_outer_loras 一致性检查在属性缺失时的崩溃。

新增 tests/e2e/megatron/model_scripts/test_inkling_small_4layer_lora_ci.py：在 stage-c-4-gpu-h200 套件（约 30 分钟）跑 4 层 Inkling-Small LoRA 冒烟，开启 --check-lora-weight-equal（engine 侧 sha256 校验 adapter 一致性），skip list 处理 frozen towers 与 engine 派生 buffer，并注册 train/grad_norm、train/ppo_kl、rollout/raw_reward 等 CI gate；文档同步补充 train modes 与 lr 建议。

关键文件：

- miles_plugins/models/inkling/lora.py（模块 适配层；类别 source；类型 data-contract；符号 InklingLoRAAdapter, _rmsnorm, _new_param, _register_param）：新增 759 行核心插件：定义 InklingLoRAAdapter 与 attention/dense MLP/routed experts/shared experts 的 LoRA 参数注册与 forward monkey-patch，全部 LoRA 参数对 Megatron

dist-checkpoint 透明（sharded_state_dict 返回空），并定义 _lora_grad_sum_group 标签契约。

- miles/backends/megatron_utils/lora_utils.py（模块 适配工具；类别 source；类型 dependency-wiring；符号 sglang_lora_target_all_sentinel, reduce_marked_lora_grads, save_lora_checkpoint, load_lora_adapter）：新增 sglang_lora_target_all_sentinel（Inkling 时让 SGLang 自动检测模块名）与 reduce_marked_lora_grads（TP/EP 复制参数梯度显式归约）；同时把 native checkpoint 分片从 tp/pp 命名改为 global_rank 命名并加 legacy fallback。
- miles/backends/megatron_utils/update_weight/update_weight_from_tensor.py（模块 权重同步；类别 source；类型 dependency-wiring；符号 _pp_assemble_full_adapter, _repack_onto_fresh_storage, _send_to_colocated_engine, _send_lora_params）：权重同步核心：新增 _pp_assemble_full_adapter 补全 PP 维度、_repack_onto_fresh_storage 解决 cuMem IPC 导出失败、扩展 skip_base_sync 条件和同步后显存回收，是 weight update 49.4s→2.5s 的关键。
- miles/backends/sglang_utils/sglang_engine.py（模块 引擎适配；类别 source；类型 dependency-wiring；符号 load_lora_adapter_from_tensors, _compute_server_args）：引擎侧适配：load_lora_adapter_from_tensors 支持整包 serialized_tensors 传输；Inkling checkpoint 时 lora_target_modules=['all'] 交给 SGLang 自动识别；dummy load 时跳过启动 lora_paths 依赖 weight-sync。
- miles/backends/megatron_utils/model.py（模块 模型构建；类别 source；类型 data-contract；符号 setup_model_and_optimizer, finalize_model_grads_with_empty_cache, initialize_model_and_optimizer）：模型构建与训练钩子：LoRA+Inkling 时用 wrap_model_provider_with_inkling_lora 包装 provider；finalize_model_grads 前插入 reduce_marked_lora_grads；--lora-adapter-path 热启动后刷新 fp32 master。
- scripts/run_inkling.py（模块 启动脚本；类别 source；类型 core-logic；符号 ScriptArgs, _train）：launcher 核心：新增 --train-mode lora 及 LoRA 相关参数拼装，lora 模式使用 triton backend + virtual experts + ep-size；fully-async 断言仅支持 full，文档同步补 lr 建议。
- tests/e2e/megatron/model_scripts/test_inkling_small_4layer_lora_ci.py（模块 CI 测试；类别 test；类型 test-coverage；符号 _args, prepare, execute）：4 层 LoRA 冒烟 CI：验证 checkpoint 转换、LoRA 训练、--check-lora-weight-equal 的 sha256 校验，注册 grad_norm/ppo_kl 等 CI gate；是功能而非精度的兜底。
- miles/utils/arguments.py（模块 参数解析；类别 source；类型 core-logic；符号 add_lora_arguments, miles_validate_args）：新增 --lora-train-only 门控；修复 experts_shared_outer_loras 与 sglang_experts_shared_outer_loras 一致性检查在属性缺失时的崩溃。
- miles/utils/lora.py（模块 开关门控；类别 source；类型 core-logic；符号 lora_rollout_enabled）：新增 lora_rollout_enabled 统一门控 rollout 侧 LoRA（enable_lora、lora_path、adapter weight sync），--lora-train-only 下关闭。

- miles/backends/megatron_utils/actor.py（模块 训练执行；类别 source；类型 dependency-wiring；符号 init, sleep, wake_up）：训练 actor 的 sleep/wake 与权重同步改用 lora_rollout_enabled，决定 torch_memory_saver 的 tag 与是否走 adapter 同步。

关键符号：InklingLoRAAdapter, _rmsnorm, _new_param, _register_param, _dropout, _grouped_linear, _apply_attention_lora, _apply_dense_mlp_lora, _apply_expert_lora, _apply_shared_experts_lora, reduce_marked_lora_grads, sglang_lora_target_all_sentinel, _pp_assemble_full_adapter, _repack_onto_fresh_storage, lora_rollout_enabled, wrap_model_provider_with_inkling_lora, load_inkling_lora_adapter

关键源码片段

miles_plugins/models/inkling/lora.py

新增 759 行核心插件：定义 InklingLoRAAdapter 与 attention/dense MLP/routed experts/shared experts 的 LoRA 参数注册与 forward monkey-patch，全部 LoRA 参数对 Megatron dist-checkpoint 透明（sharded_state_dict 返回空），并定义 _lora_grad_sum_group 标签契约。

```
# miles_plugins/models/inkling/lora.py（节选）
# 核心参数工厂：所有 Inkling LoRA 参数都从这里创建。
# 关键约定：
# - tensor_model_parallel=False / partition_dim=-1 让 Megatron 把这个参数当作
# 非 TP 切分参数（对 replicated 参数非常重要）。
# - expert=True 时 allreduce=False，EP 下每个 rank 持有本地专家副本，由
# reduce_marked_lora_grads 在 DP reduce 之前显式做 ep 组求和。
# - _lora_grad_sum_group 标签是梯度归约契约：被标记的参数必须进入
# reduce_marked_lora_grads 的统计，否则复制的 LoRA 参数会梯度不一致。

def _new_param(ref_weight, shape, *, init, grad_sum_group=None, expert=False):
    tensor = torch.empty(*shape, dtype=ref_weight.dtype, device=ref_weight.device)
    if init == "zero":
        tensor.zero_() # B 矩阵零初始化，保证训练初期 delta 为 0
    elif tensor.ndim == 2:
        nn.init.xavier_uniform_(tensor) # A 矩阵 xavier
    else:
        for expert_tensor in tensor: # 3D 参数：逐专家 xavier
            nn.init.xavier_uniform_(expert_tensor)
    param = nn.Parameter(tensor)
    param.tensor_model_parallel = False
    param.partition_dim = -1
    param.partition_stride = 1
    if expert:
        param.allreduce = False
    if grad_sum_group is not None:
        # 被 reduce_marked_lora_grads 消费：复制的 adapter 梯度需要显式求和
        param._lora_grad_sum_group = grad_sum_group
    return param
```

```
def _grouped_linear(inputs, weights, tokens_per_expert):
    """对 permuted 后的 token 缓冲做逐本地专家 matmul，一次 grouped GEMM 完成。"""
    if inputs.is_cuda:
        offsets = torch.as_tensor(list(tokens_per_expert), device=inputs.device, dtype=torch.int32).
        cumsum(
            0, dtype=torch.int32
        )
        return F.grouped_mm(inputs, weights.transpose(1, 2), offs=offsets)
    # CPU 回退：逐 segment 做普通 linear 再拼接
    segments = torch.split(inputs, list(tokens_per_expert), dim=0)
    return torch.cat([F.linear(segment, weights[idx]) for idx, segment in enumerate(segments)],
    dim=0)


def _apply_expert_lora(moe, args, hf_prefix, *, scale, dropout, a_init):
    # Inkling 的专家 LoRA 按官方 schema 拆成 w1 / w3（gate+up）与 w2（down），
    # 且与外层 routed experts 共用一份 A 矩阵。
    config = moe.config
    assert (getattr(config, "expert_tensor_parallel_size", 1) or 1) == 1, "Inkling LoRA assumes
    ETP=1"
    rank = int(args.lora_rank)
    hidden_size = config.hidden_size
    moe_intermediate = config.moe_ffn_hidden_size
    experts = moe.experts
    num_local_experts = experts.num_local_experts
    is_ep = parallel_state.get_expert_model_parallel_world_size() > 1
    ep_group = "ep" if is_ep else None

    adapter = InklingLoRAAdapter("experts", hf_prefix + "mlp.experts.")
    fc1_ref, fc2_ref = experts.linear_fc1.weight0, experts.linear_fc2.weight0
    _register_param(adapter, "w1_A", fc1_ref, (rank, hidden_size), init=a_init, grad_sum_group=
    ep_group, expert=is_ep)
    _register_param(adapter, "w3_A", fc1_ref, (rank, hidden_size), init=a_init, grad_sum_group=
    ep_group, expert=is_ep)
    _register_param(adapter, "w1_B", fc1_ref, (num_local_experts, moe_intermediate, rank),
    expert=is_ep)
    _register_param(adapter, "w3_B", fc1_ref, (num_local_experts, moe_intermediate, rank),
    expert=is_ep)
    _register_param(adapter, "w2_A", fc2_ref, (num_local_experts, rank, moe_intermediate), init=
    a_init, expert=is_ep)
    _register_param(adapter, "w2_B", fc2_ref, (hidden_size, rank), grad_sum_group=ep_group,
    expert=is_ep)
    experts.lora_adapter = adapter

    fc1 = experts.linear_fc1
    original_fc1 = fc1.forward

    def expert_fc1_forward(inputs, tokens_per_expert, *forward_args, **forward_kwargs):
        output, bias = original_fc1(inputs, tokens_per_expert, *forward_args, **forward_kwargs)
```

```
        dropped = _dropout(inputs, dropout, fc1.training)
        # 共享的 A 矩阵拼起来走一次线性，再由 grouped GEMM 落到各自专家
        joint = F.linear(dropped, torch.cat([adapter.w1_A, adapter.w3_A], dim=0))
        gate = _grouped_linear(joint[..., :rank].contiguous(), adapter.w1_B, tokens_per_expert)
        up = _grouped_linear(joint[..., rank:].contiguous(), adapter.w3_B, tokens_per_expert)
        delta = torch.cat([gate, up], dim=-1)
        return torch.add(output, delta, alpha=scale), bias

    fc1.forward = expert_fc1_forward
    # fc2 同理：w2_A 按专家 grouped，w2_B 直接线性，略
```

miles/backends/megatron_utils/lora_utils.py

新增 sglang_lora_target_all_sentinel（Inkling 时让 SGLang 自动检测模块名）与 reduce_marked_lora_grads（TP/EP 复制参数梯度显式归约）；同时把 native checkpoint 分片从 tp/pp 命名改为 global_rank 命名并加 legacy fallback。

```
# miles/backends/megatron_utils/lora_utils.py（节选）
# Inkling LoRA 与标准 LoRA 的最大差异：标准路径靠 Megatron-Bridge 的 PEFT
# 集成自动处理梯度归约，而插件直接注册的 replicated 参数（TP 复制、EP 复制）
# 不会被 Megatron 自动 reduce。这里在 finalize_model_grads 之前按参数上的
# _lora_grad_sum_group 标签显式做 all_reduce SUM，随后再走 DDP 的 DP reduce-scatter。

_marked_lora_grad_params_cache: dict[int, list] = {}


def reduce_marked_lora_grads(model) -> None:
    from megatron.core import parallel_state as ps

    key = id(model[0]) if model else 0
    marked = _marked_lora_grad_params_cache.get(key)
    if marked is None:
        marked = []
        for chunk in model:
            for param in chunk.parameters():
                group_name = getattr(param, "_lora_grad_sum_group", None)
                if group_name is not None and param.requires_grad:
                    marked.append((param, group_name))
        _marked_lora_grad_params_cache[key] = marked
    if not marked:
        return
    groups = {
        "tp": (ps.get_tensor_model_parallel_group(), ps.get_tensor_model_parallel_world_size()),
        "ep": (ps.get_expert_model_parallel_group(), ps.get_expert_model_parallel_world_size()),
    }
    for group_name in ("tp", "ep"):
        group, size = groups[group_name]
        if size <= 1:
            continue # 单 rank 组无需通信
        grads = []
```

```
        for param, g_name in marked:
            if g_name != group_name:
                continue
            grad = getattr(param, "main_grad", None) # Megatron 优化器路径优先
            if grad is None:
                grad = param.grad
            if grad is not None:
                grads.append(grad)
        # 按 dtype 分组后扁平化，一次 all_reduce 处理整组梯度，减少通信次数
        for dt in {g.dtype for g in grads}:
            gs = [g for g in grads if g.dtype == dt]
            if len(gs) == 1:
                dist.all_reduce(gs[0], op=dist.ReduceOp.SUM, group=group)
                continue
            flat = torch._utils._flatten_dense_tensors(gs)
            dist.all_reduce(flat, op=dist.ReduceOp.SUM, group=group)
            for g, red in zip(gs, torch._utils._unflatten_dense_tensors(flat, gs), strict=False):
                g.copy_(red)
```

miles/backends/megatron_utils/update_weight/update_weight_from_tensor.py

权重同步核心：新增 _pp_assemble_full_adapter 补全 PP 维度、_repack_onto_fresh_storage 解决 cuMem IPC 导出失败、扩展 skip_base_sync 条件和同步后显存回收，是 weight update 49.4s→2.5s 的关键。

```
# miles/backends/megatron_utils/update_weight/update_weight_from_tensor.py（节选）
# LoRA adapter 参数在 torch_memory_saver.region()（torch.cuda.use_mem_pool）里创建，
# 存储来自 cuMem MemPool。cuMem 分配无法通过 legacy CUDA IPC 导出，
# 直接交给 SGLang 引擎会在第一次同步时报 "CUDA error: invalid argument"。
# 这里在 region 之外重新分配普通 caching-allocator 内存，把张量拷贝过去，
# 让 MultiprocessingSerializer 能拿到可导出的 IPC handle。
# 按 (dtype, device) 合并成一份平铺 buffer，pickler 会 memoize storage，
# 引擎收到的是一把 IPC handle 而不是每个 adapter 张量一个 handle。

def _repack_onto_fresh_storage(named_tensors):
    groups: dict[tuple[torch.dtype, torch.device], list[tuple[str, torch.Tensor]]] = {}
    for name, tensor in named_tensors:
        if tensor.is_cuda:
            groups.setdefault((tensor.dtype, tensor.device), []).append((name, tensor))

    views: dict[str, torch.Tensor] = {}
    for (dtype, device), items in groups.items():
        flat = torch.empty(sum(t.numel() for _, t in items), dtype=dtype, device=device)
        offset = 0
        for name, tensor in items:
            view = flat[offset : offset + tensor.numel()].view(tensor.shape)
            view.copy_(tensor)
            views[name] = view
            offset += tensor.numel()
```

```
return {name: views.get(name, tensor) for name, tensor in named_tensors}
```

评论区精华

本 PR 无实质 review 讨论：仅 yueming-yuan 的 APPROVED（空正文），Issue 中只有 gemini-code-assist 的自动下线公告。设计决策主要沉淀在代码注释里，尤其是 _repack_onto_fresh_storage 对 torch_memory_saver MemPool 与 CUDA IPC 冲突的说明，以及 reduce_marked_lora_grads 对复制参数梯度归约契约的注释。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 同步链路影响面：update_weight_from_tensor.py 的 skip_base_sync 条件与 _send_to_colocated_engine 的 LoRA 分支是全仓库 LoRA 共用路径，本 PR 改动影响既有 LoRA 场景（如 #2075 session 侧 adapter 同步），需要回归。
  2. checkpoint 格式变化：native 分片从 adapter_megatron_tp{tp}_pp{pp}.pt 改为 adapter_megatron_rank{rank}.pt，旧格式靠 legacy fallback 读取且注释警告 only valid when EP<=TP；新格式在 EP>TP 等复杂并行下的加载一致性需额外验证。
  3. 梯度归约正确性：reduce_marked_lora_grads 依赖每个 replicated 参数都带 _lora_grad_sum_group 标签，漏标会静默产生错误梯度；缓存以 id(model[0]) 为键，模型重建且 id 复用时会命中陈旧缓存。
  4. 覆盖风险：CI 仅覆盖 4 层切片且定位是功能冒烟（非精度），42/66 层真实配置（TP4 SP PP8 EP4 等）未进 CI；non-colocate（distributed engine）的 LoRA 同步仍是 NotImplementedError。
  5. 性能：每步同步后新增 ipc_collect/empty_cache，对大模型步频的影响需结合 wandb 数据持续观察。 - 影响：对用户：Inkling / Inkling-Small 用户可按官方 LoRA schema 低成本训练，显著降低权重同步带宽与显存压力（weight update 49.4s→2.5s，step 约全参 85%）。对系统：改动横跨模型插件、Megatron 后端、SGLang 引擎、启动脚本与 CI，LoRA 共用链路的 save/load 命名与同步逻辑发生变化，其他 LoRA 后端需回归。对团队：确立了「每模块 plugin adapter + 显式 TP/EP 梯度归约 + PP 级全量组装 + CUDA IPC 直传引擎」模式，可为后续模型的原生 LoRA 提供标准范式。 - 风险标记：核心权重同步链路改动，新增 TP/EP 梯度归约协议，CI 仅覆盖 4 层切片，distributed engine LoRA 未支持，旧格式 checkpoint 兼容依赖 legacy fallback

关联脉络

- PR #1683 Inkling model + full-parameter RL: PR body 声明 stacked on 该 PR，是本次 LoRA 支持的基座；全参 GRPO 提供了后端与并行堆栈
- PR #2013 [tito] Add the Inkling TITO family (Inkling / Inkling-Small): 同一 Inkling 模型线的会话与 tokenizer 支持，后续 LoRA 会话训练可复用
- PR #2009 [docs] Add Inkling-Small model page: Inkling-Small 文档页面，本 PR 同步更新 inkling-small.md 的 LoRA 训练说明

- PR #1794 feat(multi-lora): enable and validate MoE expert adapters: MoE 专家层级 LoRA 的先例，涵盖 (tp,pp,ep) 分片与校验，本 PR 的专家 LoRA 是其补充
- PR #2075 session: apply the trained LoRA adapter to session-server rollouts: LoRA 训练与 rollout 链路共用 lora_path/adapter 同步逻辑，本 PR 修改同一链路需回归验证