PR #1683 完整报告

radixark/miles

[tml] Inkling model support

合并时间：2026-08-04 04:59

原文链接：http://prhub.com.cn/radixark/miles/pull/1683

执行摘要

- 一句话：Inkling 模型全参数 RL 支持，含多模态与启动器
- 推荐动作：值得精读。重点看三处设计：ops.py 与 sglang serving bit-identical 的 fp32 内核对齐策略、layers.py 在 CP/SP 下对 sconv 与相对位置偏置的处理、model.py 的异构 Dense/MoE 层结构与两个 provider 入口。Review 中发现的三个正确性 bug（fp32 masters 顺序、no_grad 范围、tower 同步 gate 失效）是多模态 + 权重同步组合场景的典型陷阱，可作为后续模型接入的检查清单。

功能与动机

PR body 声明目标是 “Inkling model + full-parameter RL support”，并链接 lmsys day-0 博客、sglang 支持 PR（sgl-project/sglang#31681）与 Megatron-LM 适配 PR（radixark/Megatron-LM#68），说明要同时打通 serving 与训练两端。该 PR 让 miles 用户可以直接用 Inkling / Inkling-Small 跑 GRPO 训练（全参数模式），任务覆盖 dapo_math 文本与 geo3k 视觉；同时它是 #2122 原生 LoRA 支持的基础——body 明确写 “The native LoRA layer is stacked on top in #2122”。

实现拆解

1. 模型插件层（新增）：miles_plugins/models/inkling/ 目录下，model.py 的 InklingExtra 从 HF text_config 读取 Inkling 专有配置（SWA/ 全局注意力头数、d_rel、sconv、MoE 参数等），build_inkling_config 构造 Megatron TransformerConfig；get_inkling_block_spec 利用 dense_mlp_idx 与 moe_layer_freq 构造前 Dense 后 MoE 的混合层结构，并通过 hetereogenous_dist_checkpoint 支持异构分片 checkpoint。layers.py 实现 InklingSelfAttention（qkv + 相对位置偏置一次投影、SWA 局部窗口、可选残差卷积 sconv）与 InklingRouter / InklingSharedExperts / InklingMoELayer / InklingDenseMLP。inkling_model_provider 与 inkling_mm_model_provider 两个 provider 入口分别服务纯文本与多模态。
2. 自定义算子层：ops.py 集中 CP/SP 辅助（cp_world、cp_all_gather、inkling_sp_residual_conv、seqlens_from_packed）与一组与 sglang serving bit-identical 的 fp32 精度对齐 Triton 算子（_swiglu_fwd/bwd_kernel、_sconv_fwd/bwd_kernel），以及 flex / TE / FA4 三种注意力后端。CP 要求 --allgather-cp（contiguous sharding），provider 入口显式断言，避免 zigzag 布局下 sconv 与 rel-bias 跨分片语义错误。

3. 多模态管线：mm_processor.py 的 InklingTrainProcessor 做 token 级 chat render（每个 image/audio part 插入哨兵 token）并抽取媒体；image_processing.py 实现图像 patch 化（numba/numpy 双实现，与 serving 数值一致）；audio_processing.py 实现 dmel 提取（Slaney mel 尺度 + hann 窗 STFT）；vision_encoder.py 的 HMLPPatchEncoder 做层次化 patch 折叠；mm_towers.py 的 wire_mm_towers 把冻结的 HF vision/audio tower 以 decoder_input scatter 方式接入模型 forward，并把 no_grad 限定在 tower 前向，避免 _scatter 脱离 autograd 图（review 中发现的梯度 bug，后续已修复）。
4. 权重同步与转换：megatron_to_hf/inkling.py 处理 Megatron 权重到 HF 的转换（qkvr 反分片、w13 重交织、共享专家与路由 gate 的跨调用累积组装 _PieceAccumulator）；mbridge/inkling.py 处理 HF 到 Megatron 的映射；update_weight/common.py 抽出 is_routed_expert_param 谓词，修正共享专家嵌套 ModuleList 被误判为 expert-parallel 的问题（对现有非嵌套模型是 no-op）；update_weight_from_tensor.py 增加 _mm_tower_named_tensors 把冻结 tower 权重纳入基础同步（该 gate 曾因 CLI 参数折叠失效，已修复）；model.py 中 _has_loadable_ckpt 修复 checkpoint 加载门控，只跳过“目录存在但无 iter_*/release”的情况，避免 HF 加载模式被静默跳过。
5. 启动器与测试 / 文档配套：scripts/run_inkling.py 统一启动器（模型注册表 Inkling / Inkling-Small / 4-layer 切片，_get_parallel_config 只放行已验证的 TP/PP/EP 组合；colocate 与 --fully-async 两种拓扑；配置 NVMe 优化器状态流式与 torch_memory_saver offload）。新增 tests/e2e/megatron/model_scripts/test_inkling_small_4layer_ci.py 4 层 CI 冒烟测试（全参数版），并配套 docs/models/thinkingmachines/inkling.md 文档。

关键文件：

- miles_plugins/models/inkling/ops.py（模块 核心算子；类别 source；类型 core-logic；符号 cp_world, cp_all_gather, inkling_sp_residual_conv, seqlens_from_packed）：整个插件最核心的算子层：集中 CP/SP 辅助（cp_world、cp_all_gather、inkling_sp_residual_conv、seqlens_from_packed）与与 sglang serving bit-identical 的 fp32 Triton 算子（swiglu、sconv）及三种注意力后端；精度对齐策略直接决定训练 - 服务一致性。
- miles_plugins/models/inkling/layers.py（模块 模型层；类别 source；类型 core-logic；符号 _Sconv, InklingSelfAttention, _attend, _seg）：Inkling 的 Transformer 层实现：双全局 /SWA 注意力 + 相对位置偏置 + 残差卷积 sconv、sigmoid router + 共享专家联合重归一化、MoE/Dense 混合 MLP；forward 中处理 CP all-gather 与 packed 序列切分。
- miles_plugins/models/inkling/model.py（模块 模型装配；类别 source；类型 core-logic；符号 InklingExtra, build_inkling_config, get_inkling_layer_spec, get_inkling_dense_layer_spec）：模型装配入口：从 HF text_config 构建 Megatron TransformerConfig（InklingExtra 挂在 cfg.inkling），提供 layer/block spec、纯文本与多模态两个 provider 入口，并处理 embed norm、fp32 residual、全局 scale 冻结等训练细节。
- miles_plugins/models/inkling/mm_processor.py（模块 多模态；类别 source；类型 core-logic；符号 _TokenizerAdapter, render_inkling_messages_to_ids,

_append_message, InklingTrainProcessor）：多模态训练处理器：token 级 chat render（媒体哨兵）、图像 patch 化与音频 dmel 提取，供 miles 通用多模态管线使用，是 geo3k 任务的入口。

- miles/backends/megatron_utils/megatron_to_hf/inkling.py（模块 权重转换；类别 source；类型 core-logic；符号 _PieceAccumulator, _qkv_blocks_from_gathered, _reinterleave_w13, convert_inkling_to_hf）：Megatron 权重到 HF 的转换器：处理 qkvr 反分片、w13 重交织、共享专家与路由 gate 的跨调用累积（_PieceAccumulator），并设计 atomic update group 保证权重同步完整性。
- scripts/run_inkling.py（模块 启动脚本；类别 source；类型 entrypoint；符号 ScriptArgs, _get_parallel_config, _train, prepare_cp）：统一启动器：模型注册表、已验证的 TP/PP/EP 拓扑（只放行测试过的配置）、colocate / fully-async 两种拓扑与 NVMe 优化器流式，是模型可用的直接证据；CI 测试也复用它。
- miles/backends/training_utils/mm_data.py（模块 数据管线；类别 source；类型 core-logic；符号 _expand_inkling_sample, _expand_inkling_rollout_data_in_place）：修改的核心数据路径：_expand_inkling_sample 等把 Inkling 结构化消息展开为训练样本，支撑多模态 rollout 数据转换，对 geo3k 任务的正确性至关重要。
- tests/e2e/megatron/model_scripts/test_inkling_small_4layer_ci.py（模块 CI 测试；类别 test；类型 test-coverage；符号 _args, prepare, execute）：4 层 Inkling-Small 切片的 CI 冒烟测试：验证 HF→torch_dist 转换、checkpoint 加载、GRPO 训练门控（grad_norm、ppo_kl 等），是模型支持可回归的保障。

关键符号：build_inkling_config, get_inkling_spec, inkling_model_provider, inkling_mm_model_provider, InklingSelfAttention.forward, inkling_sp_residual_conv, cp_all_gather, seqlens_from_packed, wire_mm_towers, convert_inkling_to_hf, is_routed_expert_param, _has_loadable_ckpt

关键源码片段

miles_plugins/models/inkling/ops.py

整个插件最核心的算子层：集中 CP/SP 辅助（cp_world、cp_all_gather、inkling_sp_residual_conv、seqlens_from_packed）与与 sglang serving bit-identical 的 fp32 Triton 算子（swiglu、sconv）及三种注意力后端；精度对齐策略直接决定训练 - 服务一致性。

```
# miles_plugins/models/inkling/ops.py
# Inkling 自定义算子集中地：CP/SP 辅助函数与精度对齐的 fp32 Triton 内核。
# Inkling 的残差卷积（sconv）与相对位置偏置都要求看到序列左上下文，
# 因此 CP 必须使用 contiguous sharding（--allgather-cp），而 miles 默认的
# zigzag / load-balanced 布局不满足该前提，provider 入口会 assert 掉。

def cp_world():
    """返回 (cp_size, cp_rank, cp_group)，cp_size 为 1 表示未启用上下文并行。"""
    cp = ps.get_context_parallel_world_size()
    if cp <= 1:
        return 1, 0, None
```

```
    return cp, ps.get_context_parallel_rank(), ps.get_context_parallel_group()


def cp_all_gather(x, group, world):
    """按 CP rank 顺序 all-gather [s, ...] -> [s * world, ...]。

    前提是 contiguous CP 分片：rank r 持有 token [r*s, (r+1)*s)。
    """
    xs = [torch.empty_like(x) for _ in range(world)]
    torch.distributed.all_gather(xs, x.contiguous(), group=group)
    return torch.cat(xs, dim=0)


def inkling_sp_residual_conv(config, conv, x_sbh, seqlens):
    """残差深度卷积（逐 token 因果）作用于 [s, b, h]。

    SP/CP 下序列被分片，本地 shard 的因果卷积会丢失左上下文，
    因此先 gather 全序列再做卷积，最后切回本 rank 的 slice；
    由于是残差路径，该通信开销可接受，且结果与单卡完全一致。
    seqlens 必须是全序列的 segment 长度（按 packed 序列切分）。
    """
    sp = getattr(config, "sequence_parallel", False) and ps.get_tensor_model_parallel_world_size()
    > 1
    cp, cp_rank, cp_group = cp_world()
    x = gather_from_sequence_parallel_region(x_sbh, tensor_parallel_output_grad=False) if sp
    else x_sbh
    if cp > 1:
        x = cp_all_gather(x, cp_group, cp)
    s, b, h = x.shape
    x = conv(x.reshape(s * b, h), seqlens).reshape(s, b, h)
    if cp > 1:
        sloc = s // cp
        x = x[cp_rank * sloc : (cp_rank + 1) * sloc]
    return scatter_to_sequence_parallel_region(x) if sp else x
```

miles_plugins/models/inkling/layers.py

Inkling 的 Transformer 层实现：双全局 /SWA 注意力 + 相对位置偏置 + 残差卷积 sconv、sigmoid router + 共享专家联合重归一化、MoE/Dense 混合 MLP；forward 中处理 CP all-gather 与 packed 序列切分。

```
# miles_plugins/models/inkling/layers.py
# InklingSelfAttention：qkv 额外输出相对位置偏置 r，SWA 层用局部窗口 + 残差卷积（sconv），
# 全局层用完整注意力；CP 下先 all-gather k/v 到全序列再做 sconv 与注意力。
    def forward(self, hidden_states, attention_mask, inference_context=None,
                rotary_pos_emb=None, attention_bias=None, packed_seq_params=None, **kw):
        if hidden_states.dtype != self.config.params_dtype:
            hidden_states = hidden_states.to(self.config.params_dtype)
        sq = hidden_states.shape[0]
        x = hidden_states.reshape(sq, -1)
```

```
        qkvr, _ = self.linear_qkv(x) # 一次投影出 q、k、v、r 四部分
        T = qkvr.shape[0]
        cp, cp_rank, cp_group = _cp_world()
        q, k, v, r = qkvr.split(
            [self.nh_l * self.hd, self.nkv_l * self.hd, self.nkv_l * self.hd, self.nh_l * self.d_rel], dim=-
            1
        )
        if cp > 1:
            # CP 下先 all-gather k/v 到全序列，再做 sconv 与注意力：
            # 因果卷积需要左上下文，本地 shard 会截断
            k = _cp_all_gather(k, cp_group, cp)
            v = _cp_all_gather(v, cp_group, cp)
            T_full = k.shape[0]
            seqlens = _seqlens_from_packed(packed_seq_params, T_full)
            self.config.inkling._seqlens = seqlens
            if self.k_sconv is not None:
                k = self.k_sconv(k, seqlens)
                v = self.v_sconv(v, seqlens)
            q = self.q_norm(q.reshape(-1, self.hd)).reshape(T, self.nh_l, self.hd)
            k = self.k_norm(k.reshape(-1, self.hd)).reshape(T_full, self.nkv_l, self.hd)
            v = v.reshape(T_full, self.nkv_l, self.hd)
            r = r.reshape(T, self.nh_l, self.d_rel)
            if self.attn_backend == "flex":
                out = self._seg_cp_flex(q, k, v, r, seqlens, cp_rank * T, T_full)
            else:
                out = self._seg_cp(q, k, v, r, seqlens, cp_rank * T, T_full)
        else:
            seqlens = _seqlens_from_packed(packed_seq_params, T)
            self.config.inkling._seqlens = seqlens
            if self.k_sconv is not None:
                k = self.k_sconv(k, seqlens)
                v = self.v_sconv(v, seqlens)
            q = self.q_norm(q.reshape(-1, self.hd)).reshape(T, self.nh_l, self.hd)
            k = self.k_norm(k.reshape(-1, self.hd)).reshape(T, self.nkv_l, self.hd)
            v = v.reshape(T, self.nkv_l, self.hd)
            r = r.reshape(T, self.nh_l, self.d_rel)
            out = self._attend(q, k, v, r, seqlens)
        proj_out, proj_bias = self.linear_proj(out.unsqueeze(1))
        if self.attn_sconv is not None:
            # 残差路径上的输出卷积：SP/CP 下需要先 gather 全序列再取本 rank slice
            proj_out = _sp_residual_conv(self.config, self.attn_sconv, proj_out, seqlens)
        return proj_out, proj_bias
```

miles_plugins/models/inkling/model.py

模型装配入口：从 HF text_config 构建 Megatron TransformerConfig（InklingExtra 挂在 cfg.inkling），提供 layer/block spec、纯文本与多模态两个 provider 入口，并处理 embed norm、fp32 residual、全局 scale 冻结等训练细节。

```
# miles_plugins/models/inkling/model.py
```

```
# Inkling 专有配置：从 HF text_config 读出并挂到 cfg.inkling。
# 调试开关默认走生产配方，MILES_INKLING_* 环境变量可覆盖（单卡内核调试用）。
class InklingExtra:
    def __init__(self, t: dict):
        self.num_attention_heads = t["num_attention_heads"]
        self.num_key_value_heads = t["num_key_value_heads"]
        self.head_dim = t["head_dim"]
        # SWA 层与全局注意力层使用不同的头数 / 头维
        self.swa_num_attention_heads = t["swa_num_attention_heads"]
        self.swa_num_key_value_heads = t["swa_num_key_value_heads"]
        self.swa_head_dim = t["swa_head_dim"]
        self.sliding_window_size = t["sliding_window_size"]
        self.d_rel = t["d_rel"] # 相对位置偏置的维度
        self.rel_extent = t.get("rel_extent", 1024)
        self.local_layer_ids = set(t["local_layer_ids"]) # 哪些层用 SWA
        self.sconv_kernel_size = t["sconv_kernel_size"]
        self.use_sconv = t["use_sconv"]
        self.dense_mlp_idx = int(t.get("dense_mlp_idx", 0)) # 前几层用 Dense MLP
        self.n_routed_experts = t["n_routed_experts"]
        self.num_experts_per_tok = t["num_experts_per_tok"]
        self.n_shared_experts = t["n_shared_experts"]
        self.attn_backend = os.environ.get("MILES_INKLING_ATTN_BACKEND", "flex") # flex | te |
        fa4
        self.sconv_impl = os.environ.get("MILES_INKLING_SCONV_IMPL", "triton") # triton | torch
        self.sconv_packed = os.environ.get("MILES_INKLING_SCONV_PACKED", "0") == "1"
        self.freeze_global_scale = os.environ.get("MILES_INKLING_FREEZE_GLOBAL_SCALE", "all")


def build_inkling_config(text_cfg: dict, tp=1, ep=1, pp=1, bf16=True, sp=False,
                         etp=1, cp=1, varlen=True, permute_fusion=False,
                         fp32_residual=False, pp_first_stage_layers=None,
                         pp_last_stage_layers=None) -> TransformerConfig:
    inter = text_cfg["intermediate_size"]
    ns = text_cfg["n_shared_experts"]
    denseI = int(text_cfg.get("dense_intermediate_size", inter))
    cfg = TransformerConfig(
        hidden_dropout=0.0, # 曾有 bug：dropout 未透传，默认 0.1 导致 embedding 10%
        通道被置零
        attention_dropout=0.0,
        num_layers=text_cfg["num_hidden_layers"],
        hidden_size=text_cfg["hidden_size"],
        num_attention_heads=text_cfg["num_attention_heads"],
        num_query_groups=text_cfg["num_key_value_heads"],
        kv_channels=text_cfg["head_dim"],
        ffn_hidden_size=denseI,
        tensor_model_parallel_size=tp,
        pipeline_model_parallel_size=pp,
        expert_model_parallel_size=ep,
        sequence_parallel=sp,
```

```
        context_parallel_size=cp,
        variable_seq_lengths=varlen,
        fp32_residual_connection=fp32_residual,
        num_moe_experts=text_cfg["n_routed_experts"],
        moe_router_topk=text_cfg["num_experts_per_tok"],
        moe_ffn_hidden_size=inter,
        moe_shared_expert_intermediate_size=inter if ns > 0 else None,
        moe_router_score_function="sigmoid", # 与 serving 对齐的 sigmoid 路由
        moe_router_pre_softmax=True,
        moe_router_load_balancing_type="seq_aux_loss",
        moe_aux_loss_coeff=0.0, # 训练侧关闭 aux loss
        moe_router_bias_update_rate=0.0,
        moe_router_dtype="fp32",
        moe_grouped_gemm=True,
        moe_token_dispatcher_type="alltoall",
        add_bias_linear=False,
        normalization="RMSNorm",
        layernorm_epsilon=text_cfg["rms_norm_eps"],
        qk_layernorm=True, # q/k norm + 相对位置偏置
        bf16=bf16,
        params_dtype=torch.bfloat16 if bf16 else torch.float32,
        gated_linear_unit=True,
        pipeline_dtype=torch.bfloat16 if bf16 else torch.float32,
    )
    cfg.inkling = InklingExtra(text_cfg)
    cfg.moe_activation_in_fp32 = True # 与 sglang serving 保持 fp32 语义对齐
    cfg.moe_combine_in_fp32 = True
    _didx = cfg.inkling.dense_mlp_idx
    # 前 dense_mlp_idx 层 Dense、其余 MoE 的混合结构,
    # heterogeneus dist checkpoint 处理两种层的分片差异
    cfg.moe_layer_freq = [0] * _didx + [1] * (cfg.num_layers - _didx)
    cfg.hetereogenous_dist_checkpoint = True
    return cfg
```

评论区精华

> “Optimizer fp32 mains still hold init values here, so the first step() wipes the loaded adapter. Call optimizer.reload_model_params() after this.” —— yushengsu-thu 在 miles/backends/megatron_utils/model.py 指出 LoRA adapter 加载与分布式优化器 fp32 master 初始化顺序错位；对应提交 “Reload optimizer masters” 在加载后刷新 masters。该问题出现在拆分前的中间版本，LoRA 代码随后整体迁往 #2122。

> “no_grad covers _scatter too, so embeddings get zero grad on mm batches. Wrap the towers only.” —— yushengsu-thu 在 mm_towers.py 抓到隐蔽梯度 bug：no_grad 把 embedding 克隆也包进去，多模态 batch 的 embedding 梯度恒为零。提交 “Keep the MM scatter in the autograd graph” 将上下文限定在 tower 前向。

> “This bypasses inkling_fixed.jinja and drops tool_calls / thinking-effort / end-sampling tokens, so tool and thinking flows train off-template.” —— yushengsu-thu 指出手写 renderer 缺工具调用、推理努力与 end-sampling token，训练会 off-template；最终改为路由到 inkling_fixed.jinja 并退役手写 renderer。
>
> “This arg is never registered, so the gate is always False: geo3k rollouts get garbage towers after the first offload.” —— yushengsu-thu 指出 args.inkling_mm_towers 在 CLI 折叠为 provider 后已不存在，tower 权重重同步门控失效；修复改为按 provider 路径判定。
>
> yueming-yuan 建议 “suggest not using .sh launch script, and merging the 4 Inkling scripts into 1 script (or 2 scripts if needed) and using args to select the config”，Zhichenzzz 回复 “agreed”，最终合并为 run_inkling.py 单入口并折叠 per-slice .sh 包装。
>
> yueming-yuan 在 issue 中提出 “have some thoughts on lora checkpoint; will discuss offline”，涉及 LoRA checkpoint 的后续设计，随 LoRA 拆分转入 #2122 的线下讨论。
>
> Zhichenzzz 通过三轮 CI 复跑把 4 个 h200 shard 失败定位为 main 侧回归而非本 PR：症状均为 dp-attention e2e 中 abort_request 无法触达引擎，指向 #2012（dp-aware routing），并论证 “These tests never ran on post-#2012 main”，排除了本 PR 引入回归的可能。

- LoRA adapter 加载后 optimizer fp32 masters 过期 (correctness): 提交 “Reload optimizer masters after loading a LoRA adapter” 在加载后刷新 fp32 masters；该问题出现在 LoRA 拆分前的中间版本，随后 LoRA 代码整体迁往 #2122。
- no_grad 覆盖 _scatter 导致 embedding 零梯度 (correctness): 提交 “Keep the MM scatter in the autograd graph” 将 no_grad 限定在 tower 前向，_scatter 保持在 autograd 图中。
- MM tower 权重同步 gate 读到未注册参数 (correctness): 提交 “Key the MM tower re-sync off the provider path” 改为按 provider 路径判定。
- 手写 chat renderer 绕过 inkling_fixed.jinja (design): 提交 “Render Inkling chats through the fixed template” 将 apply_chat_template 路由到 inkling_fixed.jinja 并退役手写 renderer。
- Inkling 启动脚本合并建议 (design): 最终合并为 scripts/run_inkling.py 单入口，fully-async 也以 --fully-async 标志折叠进同一脚本，并删除 per-slice 的 .sh 包装。
- h200 CI 失败归因：#2012 的 main 侧回归 (question): 该结论被接受；本 PR 的 GPU 套件成为 main 侧 dp-aware routing 回归的探测器。
- LoRA checkpoint 设计线下讨论 (question): 随 LoRA 代码拆分转入 #2122 的线下讨论，本 PR 不阻塞。

风险与影响

- 风险：
  1. checkpoint 加载门控回归：合并期 _has_loadable_ckpt 曾两次出错——先是对未设置 --load 的情况误判跳过，再是对 HF 目录（Bridge 模式）静默跳过 load_checkpoint，

导致训练在 provider 初始化权重上进行。最终修复依赖 load_checkpoint 自身 dispatch，但与 _has_loadable_ckpt 存在重复判定，后续改动易再次踩坑。

1. kernel 精度对齐风险：flex 训练后端与 fa4/ 服务端并非 bit-identical，CI 只能靠 metric-history gate（--ci-disable-kl-checker + ppo_kl 门控）而非零 KL 精确断言；swiglu、sconv、注意力三处 fp32 语义必须与 sglang 保持一致，任一算子换算顺序变化都会造成训练 - 服务不一致。
2. CP 布局约束：cp_all_gather 假定 contiguous CP sharding，provider 仅通过 assert 强制 --allgather-cp；若未来引入 zigzag CP 的其他模型复用这些 ops，会静默算错或崩溃。
3. 权重转换器假设：megatron_to_hf/inkling.py 的 qkvr 反分片明确不支持 nkv < tp 的 kv 头复制场景（注释说明训练侧保持 nkv >= tp），非标准 parallel 配置下会断言失败；专家权重假定 ETP = 1。
4. CI 覆盖有限：e2e 只覆盖 4 层 Inkling-Small 切片（约 33 GB），无法覆盖 66 层 975B 的 EP16、fully-async、多模态 tower 同步等生产路径；分布式（非 colocate）引擎的 tower 同步仍未完全支持。 - 影响：用户 / 任务：可直接启动 Inkling 与 Inkling-Small 的全参数 GRPO 训练（dapo_math 文本、geo3k 视觉），支持 colocate 与 fully-async 两种拓扑、NVMe 优化器状态流式与 torch_memory_saver offload。

系统：新增完整的模型插件体系（kimi_k3 风格目录）、Megatron↔HF 转换器与多模态数据扩展 _expand_inkling_sample；权重同步核心路径（update_weight/common.py、update_weight_from_tensor.py）的改动对所有模型生效，其中共享专家排除逻辑对非嵌套模型为 no-op。Provider 路径下的 checkpoint 加载行为也被收紧。

团队：本 PR 是 Inkling 家族支持的第一块基石，后续 #2122（LoRA）、#2013（TITO 会话）直接建立在它之上；其 h200 GPU 套件还暴露了 main 侧 dp-aware routing 回归（#2012），成为重要的回归探测器。

- 风险标记：checkpoint 加载门控回归，与 sglang kernel 精度对齐要求高，CP 强制 allgather-cp，CI 仅覆盖 4 层切片，权重同步核心路径改动

关联脉络

- PR #2122 [tml] Inkling native LoRA support: 本 PR body 明确声明原生 LoRA 叠加在其上，且最终提交 “Split out the native LoRA support to a stacked PR” 把 adapter 插件、launcher 模式与 LoRA CI 测试整体迁往 #2122。
- PR #1716 Move fully-async rollout from examples into miles/rollout: 本 PR 的 fully-async 启动器在合并期依赖该重构（FullyAsyncRolloutFn），提交 “Repair the fully-async launcher” 适配了 #1716 的新接口。
- PR #2012 router: enable dp-aware routing under dp-attention: 本 PR 的 h200 CI 失败被归因于 #2012 引入的 main 侧回归（dp-attention e2e abort_request 不可达），Zhichenzzz 在 issue 评论中给出了完整证据链。
- PR #1793 feat(optimizer): NVMe optimizer-state streaming as a miles plugin: 本 PR 启动器配置了 NVMe 优化器状态流式（--offload-train-target disk / --stream-optimizer-state-to-disk），与 #1793 的插件能力对接。

- PR #2013 [tito] Add the Inkling TITO family (Inkling / Inkling-Small): Inkling 会话模板支持，复用本 PR 建立的模型与模板设施，同属 Inkling 家族接入脉络。