PR #1929 完整报告

THUDM/slime

Feat/minimax m2.5 support

合并时间：2026-05-30 12:00

原文链接：http://prhub.com.cn/THUDM/slime/pull/1929

执行摘要

- 一句话：新增 MiniMax-M2.5 229B MoE 模型完整支持
- 推荐动作：该 PR 值得精读，特别是 MiniMaxM2SelfAttention 的 TP gather/scatter 实现和 MiniMaxM2Bridge 的配置映射，展示了如何适配自定义注意力结构到 Megatron 框架。建议合并后补充单元测试，并监控 TP 通信开销。

功能与动机

PR 标题和描述明确指出需要为 MiniMax-M2.5（256 专家，top-8 路由）提供完整集成，满足用户在大规模 MoE 模型上的强化学习训练需求。作者在评论中提供了 AIME 评估结果，证明实现正确性。

实现拆解

实现步骤

1. 自定义 SelfAttention 层（slime_plugins/models/minimax_m2.py）：继承 Megatron 的 SelfAttention，在 __init__ 中将默认的 per-head layernorm 替换为 IdentityOp，并构建全维度的 q_norm/k_norm（RMSNorm over all heads）。get_query_key_value_tensors 中通过 TP gather → norm → TP scatter 完成跨 GPU 的归一化。
2. 权重映射桥接（slime_plugins/mbridge/minimax_m2.py）：继承 Qwen2MoEBridge，覆盖 _ATTENTION_MAPPING 和 _MLP_MAPPING 以匹配 MiniMax 的 block_sparse_moe 前缀和 q_norm/k_norm 权重名。_build_config 设置 moe_router_score_function="sigmoid"、rotary_percent=0.5 等独有配置。
3. Megatron-to-HF 转换器（slime/backends/megatron_utils/megatron_to_hf/minimax_m2.py）：实现 convert_minimax_m2_to_hf 函数，将 Megatron 参数名转换为 HuggingFace 格式，处理专家权重拆分（w1/w2/w3）、QKV 融合拆分以及自定义 QK Norm 映射。
4. 枢纽文件注册（slime/backends/megatron_utils/megatron_to_hf/__init__.py 和 slime_plugins/mbridge/__init__.py）：在 _convert_to_hf_core 中增加 minimaxm2/minimax_m2 模型名分支，并在 mbridge.__init__ 中导出 MiniMaxM2Bridge。
5. 训练与转换脚本（scripts/ 目录）：新增 4 个 Shell 脚本，包含模型架构参数（scripts/models/minimax-m2.sh）、RL 训练启动（scripts/run-minimax-m2.sh）、HF→Megatron 转换、Megatron→HF 转换。训练脚本使用 TP=2, PP=2, EP=4, 16 GPUs 并配置 GRPO 优势估计。

配套说明：无新增单元测试文件，仅依赖已有的转换测试框架。配置和脚本变动已包含最佳实践示例。

关键文件：

- slime_plugins/models/minimax_m2.py（模块 模型规约；类别 source；类型 data-contract；符号 MiniMaxM2SelfAttention, init, get_query_key_value_tensors, get_minimax_m2_layer_spec）：核心模型规约文件，实现 MiniMax-M2.5 特有的全维度 QK Norm SelfAttention，是架构适配的关键。
- slime_plugins/mbridge/minimax_m2.py（模块 桥接层；类别 source；类型 core-logic；符号 MiniMaxM2Bridge, _build_config）：权重映射桥接，继承 Qwen2MoEBridge 并覆盖 Attention 和 MLP 映射，处理 MiniMax 特有的命名和配置。
- slime/backends/megatron_utils/megatron_to_hf/minimax_m2.py（模块 后处理；类别 source；类型 dependency-wiring；符号 convert_minimax_m2_to_hf）：Megatron 到 HF 的检查点转换核心逻辑，处理专家权重拆分、QKV 融合拆分以及自定义 QK Norm 映射。

关键符号：MiniMaxM2SelfAttention.init, MiniMaxM2SelfAttention.get_query_key_value_tensors, get_minimax_m2_layer_spec, MiniMaxM2Bridge._build_config, convert_minimax_m2_to_hf

关键源码片段

slime_plugins/models/minimax_m2.py

核心模型规约文件，实现 MiniMax-M2.5 特有的全维度 QK Norm SelfAttention，是架构适配的关键。

```
from megatron.core import parallel_state
from megatron.core.models.gpt.gpt_layer_specs import get_gpt_decoder_block_spec
from megatron.core.tensor_parallel import (
    gather_from_tensor_model_parallel_region,
    scatter_to_tensor_model_parallel_region,
)
from megatron.core.transformer.attention import SelfAttention
from megatron.core.transformer.identity_op import IdentityOp
from megatron.core.transformer.spec_utils import build_module


class MiniMaxM2SelfAttention(SelfAttention):
    """Custom SelfAttention for MiniMax-M2.5 with full-dimension QK Norm.

    MiniMax-M2.5 对 Q 和 K 分别做全维度 RMSNorm（所有 head 拼接），
    而不是 Megatron Core 默认的 per-head norm。
    使用 TP gather → norm → TP scatter 以保证跨 GPU 归一化正确性。
    """

    def __init__(self, config, submodules, *args, **kwargs):
        # 保存原始 layernorm 规格，暂时替换为 IdentityOp 阻止父类创建 per-head norms
        q_layernorm = submodules.q_layernorm
```

```
        k_layernorm = submodules.k_layernorm
        submodules.q_layernorm = IdentityOp
        submodules.k_layernorm = IdentityOp

        super().__init__(config, submodules, *args, **kwargs)

        # 恢复 submodules，避免影响其他地方复用
        submodules.q_layernorm = q_layernorm
        submodules.k_layernorm = k_layernorm

        # 创建全维度 norms: hidden_size = num_heads * head_dim
        self.q_norm = build_module(
            q_layernorm,
            hidden_size=self.hidden_size_per_attention_head * config.num_attention_heads,
            config=self.config,
            eps=self.config.layernorm_epsilon,
        )
        self.k_norm = build_module(
            k_layernorm,
            hidden_size=self.hidden_size_per_attention_head * config.num_query_groups,
            config=self.config,
            eps=self.config.layernorm_epsilon,
        )

    def get_query_key_value_tensors(self, hidden_states, key_value_states=None, *args,
    **kwargs):
        query, key, value = super().get_query_key_value_tensors(hidden_states, key_value_states,
        *args, **kwargs)
        # query: [sq, b, num_heads_local, head_dim]
        # key: [sq, b, num_kv_heads_local, head_dim]

        # 合并 head 维度：[sq, b, num_heads_local * head_dim]
        query = query.reshape(*query.shape[:-2], -1)
        key = key.reshape(*key.shape[:-2], -1)

        # TP gather -> 全维度 norm -> TP scatter
        tp_size = parallel_state.get_tensor_model_parallel_world_size()
        if tp_size > 1:
            query = gather_from_tensor_model_parallel_region(query)
            key = gather_from_tensor_model_parallel_region(key)

        query = self.q_norm(query)
        key = self.k_norm(key)

        if tp_size > 1:
            query = scatter_to_tensor_model_parallel_region(query)
            key = scatter_to_tensor_model_parallel_region(key)

        # 恢复 head 维度：[sq, b, num_heads_local, head_dim]
```

```
        query = query.view(*query.shape[:2], -1, self.hidden_size_per_attention_head)
        key = key.view(*key.shape[:2], -1, self.hidden_size_per_attention_head)

        return query, key, value


def get_minimax_m2_layer_spec(args, config, vp_stage=None):
    """Build layer spec for MiniMax-M2.5, 替换默认 SelfAttention 为自定义版本。"""
    kwargs = {"use_transformer_engine": args.transformer_impl == "transformer_engine"}
    if vp_stage is not None:
        kwargs["vp_stage"] = vp_stage
    spec = get_gpt_decoder_block_spec(config, **kwargs)

    for layer_spec in spec.layer_specs:
        layer_spec.submodules.self_attention.module = MiniMaxM2SelfAttention

    return spec
```

slime/backends/megatron_utils/megatron_to_hf/minimax_m2.py

Megatron 到 HF 的检查点转换核心逻辑，处理专家权重拆分、QKV 融合拆分以及自定义 QK Norm 映射。

```
import re
import torch

def convert_minimax_m2_to_hf(args, name, param):
    """将 Megatron 参数名/张量转换为 HuggingFace 格式（MiniMax-M2.5）。

    HF 使用 `block_sparse_moe` 前缀，专家权重名为 w1(gate)/w2(down)/w3(up)。
    自定义 SelfAttention 使用 `q_norm`/`k_norm` 而非 `q_layernorm`/`k_layernorm`。
    """
    # 顶层参数直接映射
    if name == "module.module.embedding.word_embeddings.weight":
        return [("model.embed_tokens.weight", param)]
    if name == "module.module.output_layer.weight":
        return [("lm_head.weight", param)]
    if name == "module.module.decoder.final_layernorm.weight":
        return [("model.norm.weight", param)]

    try:
        head_dim = args.kv_channels if args.kv_channels is not None else args.hidden_size //
        args.num_attention_heads
    except AttributeError:
        head_dim = args.hidden_size // args.num_attention_heads
    value_num_per_group = args.num_attention_heads // args.num_query_groups

    decoder_layers_pattern = r"module\.module\.decoder\.layers\.(\d+)\.(.+)"
    match = re.match(decoder_layers_pattern, name)
    if match:
```

```
layer_idx, rest = match.groups()

# MoE 专家层：linear_fc1 → w1 (gate) + w3 (up), linear_fc2 → w2 (down)
expert_pattern = r"mlp.experts\.(.+)\.weight(\d+)"
match = re.match(expert_pattern, rest)
if match:
    rest, expert_idx = match.groups()
    if rest == "linear_fc1":
        gate_weight, up_weight = param.chunk(2, dim=0)
        return [
            (f"model.layers.{layer_idx}.block_sparse_moe.experts.{expert_idx}.w1.weight",
            gate_weight),
            (f"model.layers.{layer_idx}.block_sparse_moe.experts.{expert_idx}.w3.weight", up_
            weight),
        ]
    elif rest == "linear_fc2":
        return [
            (f"model.layers.{layer_idx}.block_sparse_moe.experts.{expert_idx}.w2.weight",
            param),
        ]
    else:
        raise ValueError(f"Unknown expert parameter name: {name}")

# Attention 输出投影
if rest == "self_attention.linear_proj.weight":
    return [(f"model.layers.{layer_idx}.self_attn.o_proj.weight", param)]

# 融合 QKV 拆分为 Q/K/V（GQA: 48 heads, 8 kv heads）
elif rest == "self_attention.linear_qkv.weight":
    param = param.view(args.num_query_groups, -1, head_dim, args.hidden_size)
    q_param, k_param, v_param = torch.split(param, split_size_or_sections=[value_num_
    per_group, 1, 1], dim=1)
    q_param = q_param.reshape(-1, args.hidden_size)
    k_param = k_param.reshape(-1, args.hidden_size)
    v_param = v_param.reshape(-1, args.hidden_size)
    return [
        (f"model.layers.{layer_idx}.self_attn.q_proj.weight", q_param),
        (f"model.layers.{layer_idx}.self_attn.k_proj.weight", k_param),
        (f"model.layers.{layer_idx}.self_attn.v_proj.weight", v_param),
    ]

# 输入 layernorm
elif rest == "self_attention.linear_qkv.layer_norm_weight":
    return [(f"model.layers.{layer_idx}.input_layernorm.weight", param)]

# QK Norm（自定义注意力使用 q_norm/k_norm，而非 q_layernorm/k_layernorm）
elif rest == "self_attention.q_norm.weight":
    return [(f"model.layers.{layer_idx}.self_attn.q_norm.weight", param)]
elif rest == "self_attention.k_norm.weight":
```

```
            return [(f"model.layers.{layer_idx}.self_attn.k_norm.weight", param)]

        # 后注意力 layernorm
        elif rest == "pre_mlp_layernorm.weight":
            return [(f"model.layers.{layer_idx}.post_attention_layernorm.weight", param)]

        # 路由器
        elif rest == "mlp.router.weight":
            return [(f"model.layers.{layer_idx}.block_sparse_moe.gate.weight", param)]
        elif rest == "mlp.router.expert_bias":
            return [(f"model.layers.{layer_idx}.block_sparse_moe.e_score_correction_bias", param)]

    raise ValueError(f"Unknown parameter name: {name}")
```

评论区精华

主要讨论发生在评论线程中：合并者 zhuzilin 要求提供实现正确性的证据（W&B 截图、日志或验证输出）。作者 xs1997zju 回复了 AIME 评估结果截图和训练参数，证明模型在 DAPO-MATH-17k 数据集上的性能。随后 PR 被批准合并，未出现架构争议或重写。

- 实现正确性验证 (question): 作者 xs1997zju 提供了 AIME 评估结果截图和训练参数，证明模型在 DAPO-MATH-17k 数据集上的性能。

风险与影响

- 风险：
  1. 缺少测试覆盖：核心文件（slime_plugins/models/minimax_m2.py、slime_plugins/mbridge/minimax_m2.py 等）未对应新增单元测试，仅依赖作者提供的离线评估结果。后续修改可能引入回归。
  2. TP gather/scatter 性能开销：MiniMaxM2SelfAttention 在全维度 QK Norm 时引入 TP gather 和 scatter 操作，增加通信量，对大规模 MoE 训练可能成为瓶颈。
  3. 桥接层兼容性：MiniMaxM2Bridge 继承 Qwen2MoEBridge，如果父类内部接口变化，可能断裂。需关注父类演进。
  4. 转换器未覆盖所有参数：convert_minimax_m2_to_hf 未处理 bias 或其他非线性层，若未来模型变体增加参数则需扩展。 - 影响：对用户：新增 MiniMax-M2.5 模型支持，用户可直接使用提供的脚本进行 RL 训练和权重转换。对系统：引入新的注意力机制实现和桥接类，需维护；无破坏性变更。对团队：需维护该模型专有代码，但架构设计基于已有模式，学习成本低。 - 风险标记：缺少单元测试，TP gather/scatter 通信开销，桥接层继承脆弱性，转换器参数未全覆盖

关联脉络

- PR #2118 sync from internal: 该 PR 同步了内部代码，新增了 Megatron Teacher Server 等基础设施，为 MiniMax-M2.5 等模型提供底层支持。
- PR #2102 Support top_p mask: 该 PR 新增了 top_p 掩码支持，与 MiniMax-M2.5 的 RL 训练可能结合使用，共享部分架构改动。