PR #7050 完整报告

verl-project/verl

[rocm] feat: enable DeepSeek-V4-Flash GRPO on AMD GPUs

合并时间：2026-08-07 16:36

原文链接：http://prhub.com.cn/verl-project/verl/pull/7050

执行摘要

- 一句话：在 AMD GPU 上启用 DeepSeek-V4-Flash GRPO 训练。
- 推荐动作：值得精读。核心看点有两个：一是 restore_moe_expert_maps 以“布局是纯函数、就地位写回维持 CUDA graph 地址”的思路修复 RL 循环中推理引擎的隐藏状态，设计注释非常清晰；二是 refit 触发点（update_weights_from_ipc 的 step 1 与 process_quanted_weights_after_loading）的选择。阅读时建议同时核对当前 vLLM 版本中这些内部 API 是否仍然存在，并考虑补充一个 CPU 或 ROCm 冒烟测试。

功能与动机

PR 标题与 body 明确目标是让 DeepSeek-V4-Flash 的 GRPO 训练在 AMD GPU 上可跑通。实现上主要解决三类障碍：vLLM 引擎以 dummy 权重启动时在 ROCm 上会清零 expert 并行路由映射且权重 refit 不恢复；MLA 稀疏注意力缓存的 bf16 wo_a 副本不会随权重更新；MXFP4 refit 缺少 AITER 后端支持。此外作者在评论中回应了 Megatron 版本对齐问题，将镜像中的 Megatron-LM 固定到 NVIDIA 官方提交，避免版本不匹配带来的兼容性补丁。

实现拆解

1. 新增 verl/utils/vllm/rocm_vllm_moe_expert_map.py，提供 restore_moe_expert_maps(model)：遍历所有 RoutedExperts 层，利用 determine_expert_map 按并行布局重建 _expert_map 与 expert_mask，并就地写回以保持 CUDA graph 地址有效；形状不一致时抛错。该修复被接入 verl/workers/rollout/vllm_rollout/utils.py 的 update_weights_from_ipc：在进入权重加载流程的第一步，若 torch.version.hip 非空则对所有模型（含 drafter）先重建映射，保证后续 bucketed 权重 refit 的路由正确。
2. 在 verl/utils/vllm/vllm_quant_utils.py 新增 clear_rocm_attention_weight_caches，并在 process_quanted_weights_after_loading 开头调用：只对 ROCm + DeepSeek-V4 模型删除模块上的 _dsv4_wo_a_bf16 缓存，让 vLLM 在后续前向中惰性重建，避免注意力继续使用上一轮权重派生的 bf16 副本。
3. 在 verl/utils/vllm/vllm_fp4_utils.py 的 _refittable_mxfp4_backends 增加 AITER_MXFP4_BF16：ROCm 的 AITER 后端权重转换可通过 replace_parameter 回放，满足 refit 可重放条件，从而在 RL 权重更新后能重新生成 MXFP4 推理布局。
4. 新增 docker/rocm/Dockerfile.rocm.deepseek-v4 与 docker/rocm/patches/non-leaf-fix.patch：基于 vLLM ROCm 官方镜像，安装 ROCm

7.2.3 的 apex 与 transformer_engine wheel 及 mbridge、flash-linear-attention 等依赖，固定 Megatron-LM 到 fd1121b8… 并应用补丁 —— 补丁将分布式优化器的 model_param.view(-1)[...] 改为 detach().view(-1)[...]，避免 torch.optim 收到非叶子张量而报错。

5. 将 examples/grpo_trainer/run_deepseek_v4_flash_megatron.sh 的 MODEL_PATH、TRAIN_FILE、TEST_FILE 从硬编码改为 ${VAR:-默认值}，便于在 AMD 环境覆盖路径；examples/grpo_trainer/README.md 将 DeepSeek-V4-Flash 的硬件支持列从 nvidia 更新为 nvidia, amd。

测试与配套：本次没有新增测试文件；验证依赖 PR body 中的 step3 实验指标（rollout_probs 皮尔逊相关系数 0.996、log_ppl 差异 0.0014 等）以及 AMD 环境的实际运行。Dockerfile 与 patch 属于基础设施配套，未接入 CI 工作流。

关键文件：

- verl/utils/vllm/rocm_vllm_moe_expert_map.py（模块 权重同步；类别 source；类型 core-logic；符号 restore_moe_expert_maps）：新增文件，核心修复：ROCm 上 dummy 权重初始化会清零 expert 并行路由映射，refit 权重流不恢复，导致 token 全部路由到本地 expert 0。该函数按并行布局重建映射并就地写回。
- verl/utils/vllm/vllm_quant_utils.py（模块 量化设施；类别 source；类型 core-logic；符号 clear_rocm_attention_weight_caches, process_quanted_weights_after_loading）：核心路径修改：新增 clear_rocm_attention_weight_caches，并在 process_quanted_weights_after_loading 中调用，保证 refit 后注意力不使用上一轮权重派生的 bf16 wo_a 副本。
- verl/workers/rollout/vllm_rollout/utils.py（模块 采样引擎；类别 source；类型 dependency-wiring；符号 update_weights_from_ipc）：权重 refit 的入口点：在 update_weights_from_ipc 的 step 1 中按平台插入 restore_moe_expert_maps 调用，是所有 rollout 权重更新的必经路径。
- docker/rocm/Dockerfile.rocm.deepseek-v4（模块 镜像构建；类别 infra；类型 infrastructure）：新增 ROCm 7.2.3 运行时镜像，固定 Megatron-LM 与 Megatron-Bridge 版本，并启用 AITER 相关环境变量，是 AMD 平台可复现运行的基石。
- docker/rocm/patches/non-leaf-fix.patch（模块 镜像补丁；类别 infra；类型 infrastructure）：Megatron-LM 分布式优化器补丁：将模型参数分片改为 detach 后的叶子张量，避免 torch.optim 拒绝非叶子参数，属于镜像内依赖的必要修复。
- verl/utils/vllm/vllm_fp4_utils.py（模块 量化设施；类别 source；类型 core-logic；符号 _refittable_mxfp4_backends）：将 AITER_MXFP4_BF16 加入可回灌 MXFP4 后端，ROCm 上权重 refit 后能正确重放 MXFP4 推理布局。
- examples/grpo_trainer/run_deepseek_v4_flash_megatron.sh（模块 示例脚本；类别 other；类型 configuration）：将模型路径与训练 / 测试数据集路径参数化，便于 AMD 环境复用同一脚本。
- examples/grpo_trainer/README.md（模块 使用文档；类别 docs；类型 documentation）：文档同步声明 DeepSeek-V4-Flash 在 Megatron 引擎下支持 nvidia 与 amd，告知用户能力边界。

关键符号：restore_moe_expert_maps, clear_rocm_attention_weight_caches, process_quanted_weights_after_loading, update_weights_from_ipc, _refittable_mxfp4_backends

关键源码片段

verl/utils/vllm/rocm_vllm_moe_expert_map.py

新增文件，核心修复：ROCm 上 dummy 权重初始化会清零 expert 并行路由映射，refit 权重流不恢复，导致 token 全部路由到本地 expert 0。该函数按并行布局重建映射并就地写回。

```
# verl/utils/vllm/rocm_vllm_moe_expert_map.py

import logging
import os

logger = logging.getLogger(__name__)
logger.setLevel(os.getenv("VERL_LOGGING_LEVEL", "WARN"))


def restore_moe_expert_maps(model):
    """重建每个 ``RoutedExperts`` 层的 EP 路由映射（就地写回）。

    rollout 引擎以 dummy 权重启动，在 ROCm 上 int32 的 ``_expert_map`` 与
    ``expert_mask`` 缓冲会被初始化为 0，而 refit 权重流并不会恢复它们 ——
    这些映射只由并行布局决定，不来自 checkpoint。清零的映射会把所有 token
    路由到本地 expert 0，而不是真正的归属 rank。

    这里选择就地写回（in-place），保证此前捕获的 CUDA graph 地址仍然有效；
    该函数可安全地在每次 refit 时调用：映射是布局的纯函数，重建要么是
    无操作、要么是修复。没有启用 expert 并行的层会被跳过。
    """
    from vllm.model_executor.layers.fused_moe import RoutedExperts
    from vllm.model_executor.layers.fused_moe.expert_map_manager import determine_expert_
    map

    repaired = 0
    for module in model.modules():
        if not isinstance(module, RoutedExperts):
            continue
        manager = getattr(module, "expert_map_manager", None)
        if manager is None or not module.moe_config.moe_parallel_config.use_ep:
            continue

        # 不直接调用 ExpertMapManager._calculate_expert_maps：它每次都会把
        # fused shared experts 折进 local_num_experts，重放会虚增专家数。
        _, expert_map, expert_mask = determine_expert_map(
            ep_size=manager.ep_size,
            ep_rank=manager.ep_rank,
```

```
            global_num_experts=manager.global_num_experts,
            expert_placement_strategy=manager.placement_strategy,
            num_fused_shared_experts=manager.num_fused_shared_experts,
            return_expert_mask=manager.rocm_aiter_enabled,
        )

        for name, rebuilt in (("_expert_map", expert_map), ("expert_mask", expert_mask)):
            live = getattr(module, name, None)
            if live is None or rebuilt is None:
                continue
            if live.shape != rebuilt.shape:
                # 布局形状对不上属于硬错误：说明 rollout 引擎底下的 expert
                # 布局发生了变更，不能静默覆盖。
                raise RuntimeError(
                    f"Rebuilt {name} is {tuple(rebuilt.shape)} but the live buffer is "
                    f"{tuple(live.shape)}; the expert layout changed under the rollout engine."
                )
            live.copy_(rebuilt.to(device=live.device))
            repaired += 1

    if repaired:
        logger.info("Rebuilt %d expert-parallel routing buffers on the rollout model", repaired)
    return repaired
```

verl/utils/vllm/vllm_quant_utils.py

核心路径修改：新增 clear_rocm_attention_weight_caches，并在 process_quanted_weights_after_loading 中调用，保证 refit 后注意力不使用上一轮权重派生的 bf16 wo_a 副本。

```
# verl/utils/vllm/vllm_quant_utils.py（节选）

def clear_rocm_attention_weight_caches(model):
    """丢弃 vLLM 从已加载权重派生的 bf16 ``wo_a`` 副本。

    ``rocm_aiter_mla_sparse._get_cached_wo_a_bf16`` 会把反量化后的 ``wo_a``
    缓存在模块上，并假设权重是静态的。refit 更新了权重但不会更新缓存，
    导致注意力继续使用上一轮的副本。重建是惰性的，所以这里只需要删除缓存。

    只有 ROCm 上的 DeepSeek-V4 会构建该缓存，因此下方先做平台与模型守卫。
    """
    if torch.version.hip is None or not is_deepseek_v4_model(model):
        return

    for module in model.modules():
        if hasattr(module, "_dsv4_wo_a_bf16"):
            del module._dsv4_wo_a_bf16


def process_quanted_weights_after_loading(model, reload_state):
```

```
    """重新应用 prepare_quanted_weights_for_loading 撤消的推理布局。"""
    clear_rocm_attention_weight_caches(model) # ROCm DSV4：先失效注意力缓存
    apply_mxfp8_transformation_after_loading(model)
    reload_state = reload_state or {}
    process_fp8_weights_after_loading(reload_state.get("fp8_layers") or [])
    process_mxfp4_moe_weights_after_loading(reload_state.get("mxfp4_moe_modules") or [])
```

verl/workers/rollout/vllm_rollout/utils.py

权重 refit 的入口点：在 update_weights_from_ipc 的 step 1 中按平台插入 restore_moe_expert_maps 调用，是所有 rollout 权重更新的必经路径。

```
# verl/workers/rollout/vllm_rollout/utils.py（update_weights_from_ipc 节选）

def update_weights_from_ipc(self, peft_config: dict = None, base_sync_done=False, use_shm:
bool = False):
    """更新 rollout 模型的权重。"""
    from verl.workers.rollout.vllm_rollout.bucketed_weight_transfer import
    BucketedWeightReceiver

    if self.device is None:
        # vLLM worker 在非 CUDA 平台（如 NPU）上可能不设置 self.device，
        # 回退到当前加速器上的 local rank。
        self.device = torch.device(f"{get_device_name()}:{self.local_rank}")

    # ============================ step 1: prepare for weight loading ================
    ==========
    quant_reload_states = None

    # 引擎以 dummy 权重启动，ROCm 上其初始化会把整型缓冲清零 —— 包括 expert
    # 并行路由映射，而权重流不会恢复它们。在 refit 之前修复，保证 rollout 路由正确。
    if torch.version.hip is not None:
        from verl.utils.vllm.rocm_vllm_moe_expert_map import restore_moe_expert_maps

        for model in self._iter_all_models():
            restore_moe_expert_maps(model)

    if self._is_qat_model:
        # QAT（compressed-tensors）：必须在接收任何 bucket 之前准备好权重加载
        from verl.utils.qat import prepare_qat_for_load_weights

        for model in self._iter_all_models():
            prepare_qat_for_load_weights(model, device=self.device)
        logger.info("QAT: prepare_qat_for_load_weights completed")
    elif self._is_modelopt_qat:
        from verl.utils.modelopt.vllm_modelopt_patch import prepare_modelopt_for_weight_reload

        prepare_modelopt_for_weight_reload(self.model_runner.model, device=self.device)
        logger.info("ModelOpt: prepare_modelopt_for_weight_reload completed")
    elif is_fp8_model(self.model_runner.vllm_config):
```

```
            from verl.utils.vllm.vllm_quant_utils import prepare_quanted_weights_for_loading

            quant_reload_states = [
                (model, prepare_quanted_weights_for_loading(model)) for model in self._iter_all_
                models()
            ]
        else:
            # TODO(wuxibin): 更新的 vllm 版本不再需要。
            for model in self._iter_all_models():
                patch_vllm_moe_model_weight_loader(model)
```

评论区精华

HollowMan6 在 review 中直接提问：是否在跑 DSv4 时使用了 NVIDIA 侧指定的 Megatron 提交，并认为改动更像是版本不匹配的兼容性修复，使用这些提交时并不需要。作者回复其实际使用的是 radixark 的 Megatron-Bridge fork 与较早的 Megatron-LM 提交，随后在评论中表示“Aligned with NVIDIA”，最终镜像参数与 NVIDIA 提交对齐。结论是版本差异导致的问题通过升级依赖解决，而不是保留适配层。

gemini-code-assist 对早期 megatron_hc_head_contraction.patch 提出性能建议：hyper-connection 残差流的加权和在 transformer 前向热路径上使用 Python 循环，应改为广播加 sum(dim=2) 的向量化写法以避免多次中间张量分配。该建议针对的 patch 未出现在合并后的文件清单中（合并落地的是 non-leaf-fix.patch），因此这条意见的采纳情况不明，但体现了热路径避免 Python 循环的优化原则。

最终由 wuxibin89 approve，无未解决的阻塞性讨论。

- Megatron 版本对齐与兼容性改动是否必要 (question): 最终 Dockerfile 将 Megatron-LM 固定为 fd1121b8、Megatron-Bridge 固定为 v0.5.0，与 NVIDIA 提交对齐，版本差异问题通过升级依赖解决。
- hyper-connection 收缩的 Python 循环应向量化 (performance): 该建议针对的 patch 未出现在合并后的文件清单，实际合入的是 non-leaf-fix.patch；建议未被直接采纳，但体现了热路径避免 Python 循环的优化原则。

风险与影响

- 风险：上游 API 耦合：restore_moe_expert_maps 直接依赖 vLLM 内部符号（RoutedExperts、expert_map_manager、determine_expert_map），clear_rocm_attention_weight_caches 依赖 _dsv4_wo_a_bf16 属性名；vLLM 任一版本升级都可能使这些假设失效，而该 PR 没有配套测试拦截。

测试缺失：本次改动无新增单测或集成测试，ROCm 路径在 CI 中不可达；update_weights_from_ipc 与 process_quanted_weights_after_loading 是 rollout 权重同步的核心路径，回归面较大。

镜像可维护性：Dockerfile.rocm.deepseek-v4 强绑定 ROCm 7.2.3 wheel、固定 commits 与 non-leaf-fix.patch；Megatron-LM 优化器内部逻辑演进后 patch 可能冲突或不再必要。

影响面控制良好：两处运行时修复都有 torch.version.hip 与 is_deepseek_v4_model 守卫，非 ROCm 平台无行为变化；AITER_MXFP4_BF16 只在 AITER 后端被选择时进入回灌路径。

- 影响：用户侧：AMD GPU 用户可基于新镜像与 run_deepseek_v4_flash_megatron.sh 直接拉起 DeepSeek-V4-Flash GRPO，脚本支持通过环境变量覆盖模型与数据集路径；README 同步声明 amd 支持。 系统侧：新增一套 ROCm 运行时依赖（AITER、flash-linear-attention、mbridge 等）与镜像构建路径，需要维护。 团队侧：建立“权重 refit 后必须修复推理引擎内部派生状态”的模式（路由映射 + 量化缓存 + 注意力缓存），对后续新模型与新硬件支持有参考价值。
- 风险标记：缺少测试覆盖，依赖 vLLM 内部 API，平台限定逻辑（ROCm），新镜像未接入 CI，外部依赖版本锁定

关联脉络

- PR #7242 [veomni] feat: add DeepSeek V4 support: 同一功能线：DeepSeek-V4 支持与 MXFP4 多后端回灌，改动同样涉及 vllm_fp4_utils 与引擎配置。
- PR #7224 [vllm] feat: enhance DeepSeek V4 fp8/fp4 linear and moe weight refit: 直接相关：修改了 vllm_fp4_utils.py 与 vllm_quant_utils.py，与本次 refit 缓存修复相邻且共享文件。
- PR #7241 [megatron, hardware] fix: pure-torch fast_hadamard_transform fallback for DSA on ROCm: 同属 AMD/ROCm 硬件支持线，涉及 Megatron 与 hardware 模块，是 ROCm 平台能力补齐的连续演进。