PR #47207 完整报告

vllm-project/vllm

[ROCm]Migrating Deepseek V3.2 to vllm/models/deepseek_v32/

合并时间：2026-07-30 21:37

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47207

执行摘要

- 一句话：迁移 DeepSeek V3.2 模型至独立模块并新增 AMD ROCm 支持
- 推荐动作：建议架构和模型开发者阅读此 PR，特别是目录设计策略和平台抽象模式。AMD 用户可直接使用。关注后续是否有更多模型采用类似 multi-platform 布局。

功能与动机

该 PR 旨在将 DeepSeek V3.2 的 AMD 特定需求迁移到 vllm/models/deepseek_v32/ 目录下，遵循其他模型的组织模式，将公共元素移入common，并更新NVIDIA依赖以反映文件移动。用户通过 --model-class-overrides 可选使用 AMD 实现。

实现拆解

1. 目录结构重组织：在 vllm/models/deepseek_v32/ 下创建 amd、nvidia、common 三个子包。common 存放 fused_ops.py、kernels.py 等共享代码；amd 存放 ROCm 专属实现；nvidia 保留原有代码但仅修改导入路径。
2. AMD 模型实现：在 amd/model.py 中定义 DeepseekV32DecoderLayer 和 DeepseekV32Model，复用 DeepseekV2 的 MoE/MLP 层，但使用自定义的 DeepseekV32MLAAttention。关键变化包括使用 fused_allreduce_rms_norm 融合前向、通过 topk_indices_buffer 支持稀疏注意力索引器。
3. AMD MTP 推测解码：在 amd/mtp.py 中实现 DeepseekV32MultiTokenPredictorLayer 和 DeepseekV32MultiTokenPredictor，通过 fused_eh_norm 融合 embed 与 hidden state 归一化，并在共享头部后使用 DeepseekV32DecoderLayer 进行预测。
4. ROCm 注意力后端：在 amd/rocm.py 中定义 DeepseekV32MLASparseBackend、DeepseekV32ROCmIndexerBackend 等类，继承通用 ROCM 稀疏注意力后段，并重写 ql_nope 计算以支持 FP4/FP8 BMM 权重。
5. 平台路由：修改 vllm/models/deepseek_v32/init.py，根据 current_platform 动态选择 AMD 或 NVIDIA 实现，XPU 暂不支持并显式抛出 NotImplementedError。
6. 文件移动与导入修复：将 attention.py、fused_ops.py、kernels.py 从 nvidia/ 移至 common/，统一路径。NVIDIA 侧仅调整 import 引用，无逻辑变化。
7. 单测适配：在 tests/kernels/test_fused_deepseek_v32_norm_rope.py 中更新导入路径，验证融合核正确性。

关键文件：

- vllm/models/deepseek_v32/amd/model.py（模块 模型定义；类别 source；类型 data-contract；符号 DeepseekV32DecoderLayer, init, forward, DeepseekV32Model）：AMD 模型主入口，定义了核心 DecoderLayer、Model 和 ForCausalLM 类，是迁移的核心。
- vllm/models/deepseek_v32/amd/mtp.py（模块 MTP 模块；类别 source；类型 data-contract；符号 DeepseekV32MultiTokenPredictorLayer, init, forward, DeepseekV32MultiTokenPredictor）：AMD MTP 推测解码实现，包含 MultiTokenPredictorLayer 和顶层 Predictor，是新增的推理优化模块。
- vllm/models/deepseek_v32/amd/rocm.py（模块 注意力层；类别 source；类型 data-contract；符号 DeepseekV32MLASparseBackend, get_supported_kernel_block_sizes, DeepseekV32ROCmIndexerBackend, DeepseekV32ROCmIndexerCache）：AMD ROCm 注意力后端核心，定义稀疏注意力后端、索引器后端和 MLAAttention 类，决定注意力计算路径。
- vllm/models/deepseek_v32/attention.py（模块 注意力基类；类别 source；类型 rename-or-move）：从 nvidia/ 重命名，是通用的注意力基类，被 AMD 和 NVIDIA 共同使用。
- vllm/models/deepseek_v32/__init__.py（模块 入口调度；类别 source；类型 data-contract）：平台路由入口，根据当前平台选择 AMD 或 NVIDIA 实现，是用户感知的接口。
- vllm/models/deepseek_v32/nvidia/model.py（模块 NVIDIA 模型；类别 source；类型 data-contract）：NVIDIA 侧模型，仅导入路径修改以保证兼容，无逻辑变化。
- tests/kernels/test_fused_deepseek_v32_norm_rope.py（模块 融合核测试；类别 test；类型 test-coverage）：测试文件，更新导入路径以验证融合核正确性。

关键符号：DeepseekV32DecoderLayer.init, DeepseekV32DecoderLayer.forward, DeepseekV32Model.init, DeepseekV32ForCausalLM, DeepseekV32MultiTokenPredictorLayer.init, DeepseekV32MultiTokenPredictorLayer.forward, DeepseekV32MultiTokenPredictor.init, DeepseekV32MLAAttention.init, DeepseekV32MLAAttention._compute_ql_nope, DeepseekV32MLAAttention._run_indexer

关键源码片段

vllm/models/deepseek_v32/amd/model.py

AMD 模型主入口，定义了核心 DecoderLayer、Model 和 ForCausalLM 类，是迁移的核心。

```
# vllm/models/deepseek_v32/amd/model.py ( 片段 )
# DeepseekV32DecoderLayer 使用 fused_allreduce_rms_norm 融合前向，
# 并在初始化时根据层索引决定使用 MoE 还是 dense MLP。

class DeepseekV32DecoderLayer(torch.nn.Module):
    def __init__(
        self,
        vllm_config: VllmConfig,
        prefix: str,
        config=None,
```

```
    topk_indices_buffer: torch.Tensor | None = None,
) -> None:
    super().__init__()
    if config is None:
        config = vllm_config.model_config.hf_config
    quant_config = vllm_config.quant_config
    parallel_config = vllm_config.parallel_config

    self.hidden_size = config.hidden_size
    moe_layer_freq = getattr(config, "moe_layer_freq", 1)
    layer_idx = int(prefix.split(sep=".")[-1])
    self.layer_idx = layer_idx
    self.use_mha = False

    # 使用 AMD 专用的 MLAAttention（在 rocm.py 中定义）
    self.self_attn = DeepseekV32MLAAttention(
        vllm_config=vllm_config,
        config=config,
        prefix=f"{prefix}.self_attn",
        topk_indices_buffer=topk_indices_buffer,
    )

    # 根据层索引决定 MoE 还是 dense MLP
    if (
        config.n_routed_experts is not None
        and layer_idx >= config.first_k_dense_replace
        and layer_idx % moe_layer_freq == 0
    ):
        self.mlp = DeepseekV2MoE(
            config=config,
            parallel_config=parallel_config,
            quant_config=quant_config,
            prefix=f"{prefix}.mlp",
        )
        self.mlp.experts.moe_config.skip_final_all_reduce = True
    else:
        self.mlp = DeepseekV2MLP(
            hidden_size=config.hidden_size,
            intermediate_size=config.intermediate_size,
            hidden_act=config.hidden_act,
            quant_config=quant_config,
            prefix=f"{prefix}.mlp",
            reduce_results=False,
        )
    self.input_layernorm = RMSNorm(config.hidden_size, eps=config.rms_norm_eps)
    self.post_attention_layernorm = RMSNorm(
        config.hidden_size, eps=config.rms_norm_eps
    )
    self.routed_scaling_factor = getattr(config, "routed_scaling_factor", 1.0)
```

```
    def forward(
        self,
        positions: torch.Tensor,
        hidden_states: torch.Tensor,
        residual: torch.Tensor | None,
    ) -> tuple[torch.Tensor, torch.Tensor]:
        # 无残差时使用 input_layernorm；有残差时使用 fused_allreduce_rms_norm 融合 all-reduce
        与归一化
        if residual is None:
            residual = hidden_states
            hidden_states = self.input_layernorm(hidden_states)
        else:
            hidden_states, residual = fused_allreduce_rms_norm(
                hidden_states, residual, self.input_layernorm
            )
        hidden_states = self.self_attn(positions=positions, hidden_states=hidden_states)
        # 注意力后同样融合
        hidden_states, residual = fused_allreduce_rms_norm(
            hidden_states, residual, self.post_attention_layernorm
        )
        hidden_states = self.mlp(hidden_states)
        return hidden_states, residual
```

vllm/models/deepseek_v32/amd/mtp.py

AMD MTP 推测解码实现，包含 MultiTokenPredictorLayer 和顶层 Predictor，是新增的推理优化模块。

```
# vllm/models/deepseek_v32/amd/mtp.py ( 片段 )
# DeepseekV32MultiTokenPredictorLayer 实现单层 MTP 预测，
# 接收 input_ids, previous_hidden_states, inputs_embeds,
# 使用 fused_eh_norm 融合归一化后通过 DecoderLayer 和共享头部。

class DeepseekV32MultiTokenPredictorLayer(nn.Module):
    def __init__(self, vllm_config: VllmConfig, prefix: str) -> None:
        super().__init__()
        # 必须在 speculative 配置下运行
        assert vllm_config.speculative_config is not None
        config = vllm_config.speculative_config.draft_model_config.hf_config
        self.config = config
        quant_config = vllm_config.quant_config

        # embed norm 和 hidden norm，用于融合前
        self.enorm = RMSNorm(config.hidden_size, eps=config.rms_norm_eps)
        self.hnorm = RMSNorm(config.hidden_size, eps=config.rms_norm_eps)
        self.eh_proj = nn.Linear(config.hidden_size * 2, config.hidden_size, bias=False)

        # 准备 topk_indices_buffer 供稀疏注意力索引器使用
        topk_indices_buffer = torch.empty(
            vllm_config.scheduler_config.max_num_batched_tokens,
```

```
            config.index_topk,
            dtype=torch.int32,
            device=current_platform.device_type,
        )
        # 共享头部（在 deepseek_mtp 中定义）
        self.shared_head = SharedHead(
            config=config, prefix=prefix, quant_config=quant_config
        )
        # 核心解码块，复用 AMD 的 DecoderLayer
        self.mtp_block = DeepseekV32DecoderLayer(
            vllm_config,
            prefix,
            config=config,
            topk_indices_buffer=topk_indices_buffer,
        )

    def forward(
        self,
        input_ids: torch.Tensor,
        positions: torch.Tensor,
        previous_hidden_states: torch.Tensor,
        inputs_embeds: torch.Tensor | None = None,
        spec_step_index: int = 0,
    ) -> torch.Tensor:
        assert inputs_embeds is not None
        # fused_eh_norm 一次性完成 embed norm、hidden norm 和 RoPE
        eh_input = fused_eh_norm(
            positions,
            inputs_embeds,
            previous_hidden_states,
            self.enorm.weight,
            self.hnorm.weight,
            self.enorm.variance_epsilon,
        )
        hidden_states = self.eh_proj(eh_input)
        # 通过解码块
        hidden_states, residual = self.mtp_block(
            positions=positions, hidden_states=hidden_states, residual=None
        )
        hidden_states = tensor_model_parallel_all_reduce(hidden_states)
        hidden_states = residual + hidden_states
        hidden_states = self.shared_head.norm(hidden_states)
        return hidden_states, hidden_states

class DeepseekV32MultiTokenPredictor(nn.Module):
    # ... 管理多个 MTP 层、embedding 和 logits 处理器
```

vllm/models/deepseek_v32/amd/rocm.py

AMD ROCm 注意力后端核心，定义稀疏注意力后端、索引器后端和 MLAAttention 类，决定注意力计算路径。

```
# vllm/models/deepseek_v32/amd/rocm.py ( 片段 )
# 定义 ROCm 平台下的注意力后端类，包括稀疏后端、索引器后端和最终 MLAAttention。

class DeepseekV32MLASparseBackend(ROCMAiterMLASparseBackend):
    @staticmethod
    def get_supported_kernel_block_sizes() -> list:
        return [16, 32] # 仅支持 16/32 块大小

class DeepseekV32ROCmIndexerBackend(DeepseekV32IndexerBackend):
    @staticmethod
    def get_supported_kernel_block_sizes() -> list:
        return [16, 32]

class DeepseekV32ROCmIndexerCache(DeepseekV32IndexerCache):
    def get_attn_backend(self):
        return DeepseekV32ROCmIndexerBackend

class DeepseekV32MLAAttention(DeepseekV32Attention):
    require_fp8_kv_cache: bool = False
    indexer_cls = DeepseekV32ROCmIndexer # 使用 ROCm 索引器

    def __init__(self, vllm_config, config, prefix, topk_indices_buffer=None):
        super().__init__(
            vllm_config,
            config,
            prefix,
            topk_indices_buffer,
            attn_backend=DeepseekV32MLASparseBackend,
        )
        # 根据索引器创建 SparseAttnIndexer 操作对象
        self.indexer_op: SparseAttnIndexer | None = None
        if self.indexer is not None:
            self.indexer_op = SparseAttnIndexer(
                self.indexer.k_cache,
                self.indexer.quant_block_size,
                self.indexer.scale_fmt,
                self.indexer.topk_tokens,
                self.indexer.head_dim,
                self.indexer.max_model_len,
                self.indexer.max_total_seq_len,
                topk_indices_buffer,
                skip_k_cache_insert=True,
            )
        self._fp8_kv = is_quantized_kv_cache(self.kv_cache_dtype)
        self._fp8_kv_needs_view = self._fp8_kv and self.kv_cache_dtype != "fp8_ds_mla"

    def _compute_ql_nope(self, q_c: torch.Tensor) -> tuple[torch.Tensor, torch.Tensor]:
```

```
# 计算 ql_nope，支持 FP4/FP8 定制内核或 fallback 到 torch.bmm
q = self.q_b_proj(q_c)[0].view(-1, self.num_local_heads, self.qk_head_dim)
q_nope, q_pe = q.split([self.qk_nope_head_dim, self.qk_rope_head_dim], dim=-1)
q_nope = q_nope.transpose(0, 1)

if self.is_aiter_triton_fp4_bmm_enabled:
    from aiter.ops.triton.batched_gemm_a16wfp4 import batched_gemm_a16wfp4
    ql_nope = batched_gemm_a16wfp4(
        q_nope, self.W_K, self.W_K_scale, transpose_bm=True, prequant=True
    )
elif self.is_aiter_triton_fp8_bmm_enabled:
    from vllm._aiter_ops import rocm_aiter_ops
    ql_nope = rocm_aiter_ops.triton_fp8_bmm(
        q_nope, self.W_K, self.W_K_scale, group_size=128, transpose_bm=True
    )
else:
    ql_nope = torch.bmm(q_nope, self.W_UK_T).transpose(0, 1)
return ql_nope, q_pe
```

评论区精华

Review 中仅有一条实质性讨论：@dllehr-amd 在 amd/model.py 上要求移除多余注释（"can you remove the extraneous comments etc?"）。@stacyroberts 随后回应已移除，并调整了结构。整体讨论较少，PR 由 AMD 团队内部协作完成。

- 移除 amd/model.py 中的多余注释 (style): @stacyroberts 回应已移除并调整结构。

风险与影响

- 风险：
  1. 导入路径变更风险：common/*.py 从 nvidia/ 移至 common/，若其他模块存在直接 import 旧路径，会导致导入失败。需配合全局搜索确认无残留引用。
  2. AMD 代码未充分测试：虽然提供了 GSM8k 精度测试和性能 benchmark，但未覆盖所有 token 长度、批处理场景，可能存在边界条件（如 MTP 步数较大时的 shape 不匹配）。
  3. MTP 模块依赖：MTP 模块依赖 speculative_config，若配置不当可能触发断言失败（assert vllm_config.speculative_config is not None）。需用户正确设置 --speculative_config。
  4. XPU 阻塞：init.py 中 XPU 直接抛出 NotImplementedError，若有 XPU 用户尝试使用将无法降级，需后续补充实现。 - 影响：对 AMD GPU 用户：可以正常使用 DeepSeek V3.2 进行推理和 MTP 推测解码，通过 model-class-overrides 选配。对 NVIDIA 用户：无影响，导入路径自动走 nvidia 子模块。对系统级：重构了模型代码组织，为未来多平台模型管理奠定基础。团队需维护两套注意力后端（NVIDIA vs. ROCm），但公共算子已共享。 - 风险标记：导入路径变更可能影响其他模块，AMD 代码测试覆盖有限，MTP 模块依赖 speculative_config，XPU 暂时不支持

关联脉络

- 暂无明显关联 PR