PR #39466 完整报告

vllm-project/vllm

[XPU] Enable torch.compile for XPU GDN attention

合并时间：2026-04-24 16:26

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/39466

执行摘要

- 一句话：将 XPU GDN kernel 包装为自定义 op 以支持 torch.compile
- 推荐动作：值得关注自定义 op 注册模式，这是 vllm 中处理 torch.compile 兼容性的标准做法。建议阅读 vllm/_xpu_ops.py 中的注册流程和 forward_xpu 的简化逻辑，可对比原先的内联版本理解抽象层次。

功能与动机

Dynamo 无法跟踪直接调用 torch.ops._xpu_C.gdn_attention 的代码，导致 torch.compile 时丢失该 kernel。通过注册为自定义 op，使编译器能够识别并正确处理该算子。

实现拆解

1. 注册自定义 op：在 vllm/_xpu_ops.py 中新增 _gdn_attention_core_xpu_impl 和 _gdn_attention_core_xpu_fake 函数，通过 direct_register_custom_op 注册 opgdn_attention_core_xpu。实现函数从 forward context 获取层对象和注意力元数据，调用底层 SYCL kernel；fake 函数直接返回，用于 torch.compile 图构建。
2. 简化 XPU forward 路径：修改 vllm/model_executor/layers/mamba/gdn_linear_attn.py 中的 forward_xpu 方法，将原先内联的元数据提取和 kernel 调用逻辑替换为一行 torch.ops.vllm.gdn_attention_core_xpu(...) 调用，大幅减少代码量。
3. 纳入编译系统：在 vllm/config/compilation.py 的 _attention_ops 列表中添加 "vllm::gdn_attention_core_xpu"，使编译 / 图分割逻辑能正确识别该 op。
4. 配套调整（讨论中涉及但未包含在此 PR）：XPU 上跳过 cudagraph 内存估计的修改在 PR #39977 中单独处理。

关键文件：

- vllm/_xpu_ops.py（模块 算子注册；类别 source；类型 core-logic；符号 _gdn_attention_core_xpu_impl, _gdn_attention_core_xpu_fake）：注册自定义 op 的核心文件，包含 op 实现和 fake 实现，决定了 torch.compile 如何跟踪 XPU GDN kernel。
- vllm/model_executor/layers/mamba/gdn_linear_attn.py（模块 注意力层；类别 source；类型 data-contract；符号 forward_xpu）：修改 forward_xpu 方法，用自定义 op 替换内联 kernel 调用，是实际使用新 op 的地方。
- vllm/config/compilation.py（模块 编译配置；类别 source；类型 core-logic）：在 attention ops 列表中添加新 op，使编译系统正确处理该 op。

关键符号：_gdn_attention_core_xpu_impl, _gdn_attention_core_xpu_fake, forward_xpu

关键源码片段

vllm/_xpu_ops.py

注册自定义 op 的核心文件，包含 op 实现和 fake 实现，决定了 torch.compile 如何跟踪 XPU GDN kernel。

```
# 自定义 op 实现：从 forward context 获取层和注意力元数据，调用 SYCL kernel
# 注意导入在函数内部以避免循环依赖
def _gdn_attention_core_xpu_impl(
    core_attn_out: torch.Tensor,
    z: torch.Tensor,
    projected_states_qkvz: torch.Tensor,
    projected_states_ba: torch.Tensor,
    layer_name: str,
) -> None:
    from vllm.forward_context import get_forward_context
    from vllm.v1.attention.backends.gdn_attn import GDNAttentionMetadata

    forward_context = get_forward_context()
    self = forward_context.no_compile_layers[layer_name]
    attn_metadata_raw = forward_context.attn_metadata

    if attn_metadata_raw is None:
        return # profiling 时无元数据，跳过 kernel，z 保持 empty

    assert isinstance(attn_metadata_raw, dict)
    attn_metadata = attn_metadata_raw[self.prefix]
    assert isinstance(attn_metadata, GDNAttentionMetadata)
    assert attn_metadata.spec_sequence_masks is None # XPU 暂不支持推测解码

    conv_weights = self.conv1d.weight.view(
        self.conv1d.weight.size(0), self.conv1d.weight.size(2)
    )

    torch.ops._xpu_C.gdn_attention(
        core_attn_out, z, projected_states_qkvz, projected_states_ba,
        self.num_k_heads, self.num_v_heads, self.head_k_dim, self.head_v_dim,
        conv_state=self.kv_cache[0], ssm_state=self.kv_cache[1],
        conv_weights=conv_weights, conv_bias=self.conv1d.bias,
        activation=self.activation, A_log=self.A_log, dt_bias=self.dt_bias,
        num_prefills=attn_metadata.num_prefills,
        num_decodes=attn_metadata.num_decodes,
        has_initial_state=attn_metadata.has_initial_state,
        non_spec_query_start_loc=attn_metadata.non_spec_query_start_loc,
        non_spec_state_indices_tensor=attn_metadata.non_spec_state_indices_tensor,
        num_actual_tokens=attn_metadata.num_actual_tokens,
        tp_size=self.tp_size,
```

```
        reorder_input=not self.gqa_interleaved_layout,
    )


def _gdn_attention_core_xpu_fake(
    core_attn_out: torch.Tensor,
    z: torch.Tensor,
    projected_states_qkvz: torch.Tensor,
    projected_states_ba: torch.Tensor,
    layer_name: str,
) -> None:
    return # fake impl: no-op
```

评论区精华

核心讨论：为什么需要自定义 op

> jikunshang: "what's the relationship with torch.compile?" 作者：之前的 _xpu_C.gdn_attention 未注册为 vllm 自定义 op，编译器无法处理。

导入顺序问题

> jikunshang: 建议在函数内部导入 GDNAttentionMetadata 以避免循环依赖。 作者采纳，将导入移至 _gdn_attention_core_xpu_impl 函数内部。

z tensor 初始化风险

> Copilot: 当 attn_metadata 为 None 时，op 直接返回，z 保持 torch.empty，后续 norm 使用未初始化数据产生不确定输出。建议在返回前 z.zero_()。 未在最终代码中采纳（原逻辑已有相似问题），但可能在实际使用中因 profiling 路径短而未被触发。

- 为什么需要自定义 op (design): 确认需要通过注册自定义 op 来实现 torch.compile 的 traceability。
- z tensor 未初始化风险 (correctness): 未采纳，认为原逻辑已有相似问题且 profile 阶段后续不会实际使用 z（可能因重新初始化）。风险仍存在。
- 导入顺序和循环依赖 (design): 作者采纳，将 GDNAttentionMetadata 导入移到函数内部。

风险与影响

- 风险：
  1. z tensor 未初始化：forward_xpu 中 z 初始化为 torch.empty_like，若 op 在 attn_metadata 为 None 时直接返回（profile 阶段），后续输出投影使用未初始化的 z 可能导致 NaN 或错误。原逻辑已有此问题，但可能因 profile 后重新初始化而逃逸。
  2. 导入顺序风险（已缓解）：通过将 GDNAttentionMetadata 的导入移到函数内部，避免了模块级循环依赖。
  3. 平台专用风险：变更仅影响 XPU 平台，对其他平台无影响，但若 XPU 上 op 注册失败，forward_xpu 会直接报错。

4. 缺少测试覆盖：未看到针对新 op 的单元测试或集成测试。 - 影响：启用 torch.compile 后，XPU 上使用 GDN attention 的模型（如 Qwen3.5 等 Mamba 架构）可获得编译优化带来的性能提升。影响范围限于 XPU 平台，且仅影响注意力计算路径。代码量减少，可维护性提升。 - 风险标记：缺少测试覆盖，z tensor 未初始化风险，XPU 平台专用

关联脉络

- PR #39977 skip cudagraph memory profiling for XPU when cudagraph_mode is NONE: 讨论中引用了此 PR 来处理 XPU 上 cudagraph 内存估计的跳过逻辑，属于功能依赖。