

# PR #55272 完整报告

vllm-project/vllm

[Qwen3.8-Flash-Next] Remove torch.compile for NVIDIA implementation

合并时间: 2026-09-07 12:57

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/55272>

## 执行摘要

- 一句话: 移除 Qwen3.8-Flash-Next NVIDIA 路径的 torch.compile 依赖, 改用可中断 CUDA 图。
- 推荐动作: 此 PR 值得精读, 因为它不仅修复了一个重要的生产问题 (FP8 OOM), 还展示了如何将模型执行路径从 torch.compile 的隐式优化迁移至更显式、可控制的 CUDA 图捕获模式。关键的设计决策包括:
  - 选择性 eager break: 仅在需要运行时元数据 (如 attn\_metadata) 的函数上使用 @eager\_break\_during\_capture, 而不是在整个模型前向传播中。
  - 缓冲区地址稳定性: 通过确保 ngram\_context 和 query\_start\_loc 缓冲区在捕获和重放期间使用相同的内存地址, 避免了 CUDA 图的重新捕获。
  - 参数传递重构: 将 Q/K 投影提前到 Qwen4ExpQSAAttention.forward 中, 使 QSAIndexer.forward 接收预投影的 projected\_qk, 简化了数据流。

推荐关注以下文件: [vllm/models/qwen4\\_exp/nvidia/qsas.py](#) (QSA 注意力路径的重构)、[vllm/models/qwen4\\_exp/nvidia/model\\_state.py](#) (缓冲区稳定性逻辑) 和 [vllm/models/qwen4\\_exp/nvidia/ple\\_layer.py](#) (PLE 层的 eager break 添加)。

## 功能与动机

此 PR 解决了 Issue #54688 中提出的问题: Qwen3.8-Flash-Next 的 NVIDIA 实现 (内部称为 Qwen4Exp) 仍然对主模型和 MTP 类应用 `support_torch_compile`。CUDA 图捕获并不需要这种模型级的 AOT 编译, 且模型特定操作越来越多地以显式 vLLM 自定义算子的形式实现。移除 torch.compile 还能解决一个关键的实际部署问题: 在单个 GB300 上加载 FP8 检查点时, torch.compile 会尝试自动调优 embedding 算子, 导致一个额外的 n-gram 表张量消耗约 50GB 内存, 从而引发 OOM。

## 实现拆解

- 移除 `@support_torch_compile` 装饰器: 从 `vllm/models/qwen4_exp/nvidia/model.py` 中的 `Qwen4ExpModel` 类、`vllm/models/qwen4_exp/nvidia/mtp.py` 中的 `Qwen4ExpMultiTokenPredictor` 和 `Qwen4ExpMTP` 类上移除了 `@support_torch_compile` 装饰器及其 `dynamic_arg_dims` 参数。这标志着模型不再参与 torch.compile 的全局图捕获和优化。

2. 引入可中断 CUDA 图捕获 (`@eager_break_during_capture`): 在关键的、依赖运行时元数据的函数上添加了 `@eager_break_during_capture` 装饰器, 确保它们在 CUDA 图捕获期间被中断并以 eager 模式执行。具体包括:
  - `vllm/models/qwen4_exp/nvidia/ple_layer.py` 中的 `Qwen4ExpPLELayer._short_conv` 方法 (PLE 短卷积)。
  - `vllm/models/qwen4_exp/nvidia/qa.py` 中的 `Qwen4ExpQSAAttention._run_qsa` 方法 (QSA 缓存更新、索引和注意力计算)。
  - `vllm/model_executor/layers/mamba/gdn/qwen_gdn_linear_attn.py` 中的 `qwen_gdn_attention_core` 和 `qwen_gdn_attention_core_fused_norm_packed` 函数 (GDN 注意力核心)。
3. 移除 NVIDIA 路径的自定义算子: 删除了专门为 `torch.compile` 设计的自定义算子注册, 这些算子原本用于绕过编译器检查。具体删除了:
  - `vllm/models/qwen4_exp/nvidia/ple_layer.py` 中的 `qwen4_exp_compute_ple_ngram_ids` 和 `qwen4_exp_ple_short_conv` 函数及其 `direct_register_custom_op` 调用。
  - `vllm/models/qwen4_exp/nvidia/qa.py` 中的 `qwen4_exp_qsa_with_output` 函数及其 `direct_register_custom_op` 调用。
4. 重构函数调用和参数传递: 为了适应移除自定义算子后的直接调用模式, 调整了多个函数的参数:
  - `Qwen4ExpPLELayer.forward` 方法现在直接调用 `self.compute_ngram_ids` 和 `self._short_conv`, 而不是通过 `torch.ops.vllm.*`。
  - `Qwen4ExpQSAAttention.forward` 方法现在直接计算 `projected_qk` 并调用 `self._run_qsa`, 移除了通过 `torch.ops.vllm.qwen4_exp_qsa_with_output` 或本地函数 `qwen4_exp_qsa_with_output` 的间接调用。
  - `QSAIndexer.forward` 方法的输入从 `hidden_states` 改为 `projected_qk`, 将 Q/K 投影提前到调用方完成。
5. 调整模型状态缓冲区以保持稳定地址: 修改了 `vllm/models/qwen4_exp/nvidia/model_state.py` 中的 `Qwen4ExpModelState` 类, 确保其 `ngram_context` 和 `ple_query_start_loc` 缓冲区在 CUDA 图捕获 (通过 `prepare_dummy_inputs`) 和后续重放 (通过 `prepare_inputs`) 期间保持固定的形状和内存地址。具体修改包括:
  - 缓冲区始终使用 `max_num_reqs` 作为第一维, 而不是根据当前 `num_reqs` 或 `num_reqs_padded` 进行切片。
  - 在 `prepare_inputs` 中, 将未使用的容量表示为尾部零长度请求, 填充 `query_start_loc` 的剩余部分。
6. 更新平台配置和测试:
  - 在 `vllm/config/vllm.py` 和 `vllm/config/compilation.py` 中, 将 Qwen4Exp 系列模型 (包括 `Qwen4ExpForCausalLM`、`Qwen4ExpForConditionalGeneration` 和 `Qwen4ExpMTP`) 添加到可中断 CUDA 图的平台默认架构列表中。
  - 在 `tests/test_config.py` 中, 更新了测试以验证这些新模型在非 ROCm 平台上默认启用可中断 CUDA 图。

- 在 `tests/models/qwen4_exp/test_qsa_reference.py` 中，新增了 `test_qsa_unfused_cache_update_ignores_padded_qk` 测试，验证填充的 Q/K 行不会影响缓存更新。
- 在 `tests/models/qwen4_exp/test_ple.py` 中，增加了测试以验证 PLE 模块在移除编译后的正确性。
- 在 `tests/models/qwen4_exp/test_config.py` 中，扩展了测试以验证 `Qwen4ExpModelState` 在请求布局变化时，其准备的输入缓冲区（`query_start_loc` 和 `ngram_context`）地址保持不变。

关键文件：

- `vllm/models/qwen4_exp/nvidia/qsa.py`（模块 QSA 注意力；类别 source；类型 core-logic；符号 `qwen4_exp_qsa_with_output`, `Qwen4ExpQSAAttention._run_qsa`, `Qwen4ExpQSAAttention.forward`）：QSA 注意力路径的核心重构文件，移除了自定义算子并引入了 `eager break`，是变更最集中的源码文件。
- `vllm/models/qwen4_exp/nvidia/model_state.py`（模块 模型状态；类别 source；类型 data-contract；符号 `Qwen4ExpModelState._prepare_ngram_context`, `Qwen4ExpModelState.prepare_inputs`, `Qwen4ExpModelState.prepare_dummy_inputs`）：负责为 CUDA 图捕获和重放准备稳定的模型输入缓冲区，其修改是确保新执行模式正确性的关键。
- `vllm/models/qwen4_exp/nvidia/ple_layer.py`（模块 PLE 层；类别 source；类型 core-logic；符号 `qwen4_exp_compute_ple_ngram_ids`, `qwen4_exp_ple_short_conv`, `Qwen4ExpPLELayer._short_conv`, `Qwen4ExpPLELayer.forward`）：PLE 层的核心逻辑文件，移除了两个自定义算子并添加了 `eager break`，是变更量最大的源码文件之一。
- `tests/test_config.py`（模块 配置测试；类别 test；类型 test-coverage；符号 `test_breakable_cudagraph_platform_default`）：测试文件，验证了新增的 `Qwen4Exp` 模型默认使用可中断 CUDA 图的配置逻辑。
- `tests/models/qwen4_exp/test_qsa_reference.py`（模块 QSA 测试；类别 test；类型 test-coverage；符号 `test_qsa_unfused_cache_update_ignores_padded_qk`）：测试文件，新增了验证填充 Q/K 输入不影响缓存的测试，是重构后正确性保障的重要部分。

关键符号：`qwen4_exp_compute_ple_ngram_ids`, `qwen4_exp_ple_short_conv`, `qwen4_exp_qsa_with_output`, `Qwen4ExpPLELayer._short_conv`, `Qwen4ExpQSAAttention._run_qsa`, `Qwen4ExpQSAAttention.forward`, `Qwen4ExpNGramEmbedding.forward`, `Qwen4ExpModelState._prepare_ngram_context`, `Qwen4ExpModelState.prepare_inputs`, `Qwen4ExpModelState.prepare_dummy_inputs`, `QSAIndexer.forward`

## 关键源码片段

### `vllm/models/qwen4_exp/nvidia/qsa.py`

QSA 注意力路径的核心重构文件，移除了自定义算子并引入了 `eager break`，是变更最集中的源码文件。

# `vllm/models/qwen4_exp/nvidia/qsa.py`（部分重构后的代码）

```

import torch
from vllm.compilation.breakable_cudagraph import eager_break_during_capture
# ... 其他导入 ...

class Qwen4ExpQSAAttention(nn.Module):
    # ... __init__ 等方法 ...

    @eager_break_during_capture # 新增：在 CUDA 图捕获期间中断，以 eager 模式执行
    def _run_qsa(
        self,
        projected_qk: torch.Tensor, # 参数从 hidden_states 改为 projected_qk
        positions: torch.Tensor,
        query: torch.Tensor,
        key: torch.Tensor,
        value: torch.Tensor,
        output: torch.Tensor,
    ) -> None:
        # ... 内部逻辑保持不变，但接收预投影的 Q/K ...

    def forward(
        self,
        positions: torch.Tensor,
        hidden_states: torch.Tensor,
    ) -> torch.Tensor:
        qkv, _ = self.qkv_proj(hidden_states)
        q, k, v, gate = self._project_qkv_gate(qkv, positions)
        num_tokens = hidden_states.shape[0]
        query = q.view(num_tokens, self.num_heads, self.head_dim)
        key = k.view(num_tokens, self.num_kv_heads, self.head_dim)
        value = v.view(num_tokens, self.num_kv_heads, self.head_dim)
        attn_output = torch.empty_like(query)
        # 新增：将索引投影移出 eager break，提前计算 projected_qk
        projected_qk, _ = self.indexer.index_qk_proj(hidden_states)
        # 直接调用 _run_qsa，不再通过 torch.ops 或本地函数间接调用
        self._run_qsa(
            projected_qk,
            positions,
            query,
            key,
            value,
            attn_output,
        )
        flat_output = attn_output.view(num_tokens, -1)
        if gate is not None:
            flat_output = flat_output * torch.sigmoid(gate)
        output, _ = self.o_proj(flat_output)
        return output

# 移除了 qwen4_exp_qsa_with_output 函数和 direct_register_custom_op 调用

```

## vllm/models/qwen4\_exp/nvidia/model\_state.py

负责为 CUDA 图捕获和重放准备稳定的模型输入缓冲区，其修改是确保新执行模式正确性的关键。

```
# vllm/models/qwen4_exp/nvidia/model_state.py (部分重构后的代码)
```

```
import torch
```

```
# ... 其他导入 ...
```

```
class Qwen4ExpModelState(MambaHybridModelState):
```

```
    # ... __init__ 等方法 ...
```

```
    def _prepare_ngram_context(
```

```
        self,
```

```
        input_batch: InputBatch,
```

```
        req_states: RequestState,
```

```
) -> torch.Tensor:
```

```
    num_reqs = input_batch.num_reqs
```

```
    # 修改: 始终使用完整的 ngram_context 缓冲区, 而不是切片
```

```
    context = self.ngram_context
```

```
    context.fill_(self.ngram_eos_token_id)
```

```
    if num_reqs == 0:
```

```
        return context
```

```
    # ... 后续填充逻辑不变 ...
```

```
    def prepare_inputs(
```

```
        self,
```

```
        input_batch: InputBatch,
```

```
        req_states: RequestState,
```

```
) -> dict[str, Any]:
```

```
    model_inputs = super().prepare_inputs(input_batch, req_states)
```

```
    if not self.uses_ngram_embedding:
```

```
        return model_inputs
```

```
    num_reqs_padded = input_batch.num_reqs_after_padding
```

```
    query_start_loc = self.ple_query_start_loc # 使用完整的缓冲区
```

```
    query_start_loc[: num_reqs_padded + 1].copy_(input_batch.query_start_loc)
```

```
    # 新增: 将未使用的容量表示为尾部零长度请求
```

```
    query_start_loc[num_reqs_padded + 1:].copy_(input_batch.query_start_loc[-1])
```

```
    model_inputs.update(
```

```
        query_start_loc=query_start_loc,
```

```
        ngram_context=self._prepare_ngram_context(input_batch, req_states),
```

```
)
```

```
    return model_inputs
```

```
    def prepare_dummy_inputs(
```

```
        self,
```

```
        num_reqs: int,
```

```
        num_tokens: int,
```

```
) -> dict[str, Any]:  
    # ... 类似调整, 确保缓冲区在捕获期间地址固定 ...
```

## vllm/models/qwen4\_exp/nvidia/ple\_layer.py

PLE 层的核心逻辑文件, 移除了两个自定义算子并添加了 eager break, 是变更量最大的源码文件之一。

```
# vllm/models/qwen4_exp/nvidia/ple_layer.py ( 部分重构后的代码 )  
import torch  
from vllm.compilation.breakable_cudagraph import eager_break_during_capture  
# ... 其他导入 ...  
  
class Qwen4ExpPLELayer(nn.Module):  
    # ... __init__ 等方法 ...  
  
    # 新增: 在 CUDA 图捕获期间中断, 以 eager 模式执行短卷积  
    @eager_break_during_capture  
    def _short_conv(self, inputs: torch.Tensor, residual: torch.Tensor) -> None:  
        # ... 内部逻辑保持不变 ...  
  
    def forward(  
        self,  
        inputs: torch.Tensor,  
        positions: torch.Tensor,  
        conv_state: torch.Tensor,  
        # ... 其他参数 ...  
    ) -> torch.Tensor:  
        # ... 前部分逻辑 ...  
        # 修改: 直接调用 compute_ngram_ids 和 _short_conv, 移除了 torch.ops 调用  
        ngram_ids = self.compute_ngram_ids(input_ids, query_start_loc, ngram_context)  
        # ... 中间逻辑 ...  
        self._short_conv(conv_input, gated_output)  
        return gated_output  
  
# 移除了 qwen4_exp_compute_ple_ngram_ids, qwen4_exp_ple_short_conv 函数和 direct_register_  
custom_op 调用
```

## 评论区精华

讨论主要集中在代码清理和遗留依赖上。评审者 ZJY0516 指出 `vllm/model_executor/layers/mamba/gdn/qwen_gdn_linear_attn.py` 中的一些导入可以移除, 但作者 gau-nerst 解释说这些导入“仍然为现有 Qwen 模型所需”, 表明在清理特定于此 PR 的变更时, 需要考虑对其他模型的兼容性。另一个由 coderabbitai[bot] 提出的重要技术点是, 在 `vllm/models/qwen4_exp/nvidia/indexer_qsa.py` 的未融合路径中, `projected_qk` 的重塑操作可能包含 K 列, 导致 `positions` 的长度不匹配。虽然此 PR 未直接修复该问题, 但它突出了在移除编译依赖后, 手动管理数据形状和流动路径的重要性。整体上, 讨论反映了从自动编译优化转向显式、手动控制执行路径时, 需要仔细审视数据流和边界条件。

- GDN 线性注意力层中导入的清理 (style): 作者 gau-nernst 解释说这些导入“仍然为现有 Qwen 模型所需”，因此未在本次 PR 中移除。
- QSA 索引器未融合路径中 projected\_qk 的重塑 (correctness): 此问题在 PR 评论中提出，但未在本次提交中修复，可能作为后续优化或修复处理。

## 风险与影响

- 风险:

1. 性能回归风险: 移除 torch.compile 可能导致自动内核融合和调度优化的丢失。PR 中提供的性能数据显示，在低并发 (1-4) 下，吞吐量有轻微下降 (-2% 至 -5%)，而在高并发 (16-64) 下有轻微提升。作者认为结果“看起来有噪音”，并指出“未来的手动融合将进一步提高性能”。风险在于，如果未来没有手动优化，低并发场景的性能可能会持续受影响。
2. 兼容性风险: 此变更主要影响 NVIDIA 平台的 Qwen4Exp 模型。移除自定义算子后，必须确保新的直接调用路径与所有现有检查点和部署配置兼容。测试覆盖了 BF16 和 FP8 检查点，但未明确覆盖所有可能的量化配置 (如 ModelOpt)。
3. CUDA 图捕获和重放的正确性风险: 引入 @eager\_break\_during\_capture 和调整缓冲区逻辑增加了 CUDA 图管理的复杂性。虽然测试 (如 test\_qsa\_unfused\_cache\_update\_ignores\_padded\_qk 和 test\_config.py 中的缓冲区地址测试) 旨在验证正确性，但复杂的动态批处理和投机解码场景仍可能存在边界情况。
4. 安全风险: 无直接安全风险。此变更主要是内部执行模型的重构。

- 影响:

1. 用户影响: 对于使用 NVIDIA GPU 部署 Qwen3.8-Flash-Next 模型的用户，此 PR 解决了 FP8 检查点加载时的 OOM 问题，使得在单张 GB300 上运行成为可能。推理性能在大多数情况下保持稳定，但用户可能会观察到低并发场景下吞吐量的微小变化。
2. 系统影响: 此变更影响了 vLLM 的模型执行引擎，特别是 CUDA 图捕获和重放机制。它引入了可中断 CUDA 图作为一种新模式，用于处理依赖运行时元数据的操作。这可能为其他类似模型 (如当前在 GDN 中保留的导入) 的未来优化奠定基础。
3. 团队影响: 此 PR 为团队提供了一个从自动编译依赖迁移到更显式、可预测执行路径的案例研究。它强调了手动管理内存布局 (如固定缓冲区地址) 和执行流程 (如显式 eager break) 的重要性。相关的性能权衡和测试策略可以为未来类似重构提供参考。 - 风险标记: 核心路径变更，性能权衡，CUDA 图兼容性，缺少全面量化配置测试

## 关联脉络

- PR #54688 [RFC] Remove torch.compile dependency from Qwen3.8-Flash-Next NVIDIA path: 本 PR 是对此 RFC Issue 的实现，直接解决了其提出的目标：移除 torch.compile 依赖。
- PR #53896 [Qwen3.8-Flash-Next] Initial NVIDIA implementation: 本 PR 修改的 Qwen4Exp 模型文件 (如 model.py, qsa.py, ple\_layer.py) 最初由该 PR 引入。本 PR 移除了该实现中添加的 @support\_torch\_compile 装饰器。