

PR #47187 完整报告

vllm-project/vllm

Make the Transformers modeling backend as fast as native vLLM

合并时间: 2026-07-06 23:59

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/47187>

执行摘要

- 一句话: Transformers 后端通过自动化层融合达到原生性能
- 推荐动作: 该 PR 是 Transformers 后端的关键进化, 展示了如何在不牺牲性能的前提下利用 Transformers 模型定义。强烈建议阅读:
 - `fx_utils.py` 的图追踪和 AST 工具设计
 - `base.py` 的融合器框架接口定义
 - 各融合器的 `match` 和 `update_forward` 实现, 尤其是 AST 操作技巧
 - 基准测试结果部分了解性能收益。

功能与动机

Transformers 建模后端之前因未利用 vLLM 的融合层和 MoE 优化, 性能落后于原生实现。本 PR 旨在消除差距, 降低维护成本, 并允许用户使用 Transformers 模型定义同时享受 vLLM 的所有优化。PR 标题直接表明: 'Make the Transformers modeling backend as fast as native vLLM'。

实现拆解

实现分为几步:

1. 图追踪与模式检测: 在 `fx_utils.py` 中提供 `trace` 函数对模块的 `forward` 进行 `torch.fx` 追踪, 容忍部分追踪失败。`_AllLeafTracer` 将子模块视为叶子节点, 保持调用结构。
2. 融合器框架: 在 `fusers/base.py` 定义 `BaseFuser` 和 `StackedFuser` 抽象类, 规范 `match`、`validate`、`fuse` 等接口。
3. 具体融合器实现:
 - `QKVFuser` (`qkv.py`): 匹配 $q(x), k(x), v(x)$ 模式, 通过 AST 重写 `forward` 为单一 `QKVParallelLinear` 调用并 `split`。
 - `GLUFuser` (`glu.py`): 匹配 $\text{act}(\text{gate}(x)) * \text{up}(x)$ 模式, 融合为 `MergedColumnParallelLinear` 并替换激活函数为 `ActivationAndMul` 变体。
 - `MoEBlockFuser` (`moe.py`): 识别门控网络和共享专家, 替换为 `FusedMoE`, 利用 vLLM 的路由和异步执行。
 - `RMSNormFuser` (`rms_norm.py`): 结构识别 `RMSNorm` 模式, 替换为 vLLM 的 `fused RMSNorm` 或 `GemmaRMSNorm`, 并支持张量并行下的全 `gather`。

- 集成到模型加载：在 `transformers/base.py` 中通过 `recursive_replace` 递归应用所有匹配的融合器，并记录日志。
- 测试与文档：为每个融合器编写了单元测试（`tests/models/transformers/fusers/`），验证模式匹配和正确性。更新了相关文档。

关键文件：

- `vllm/model_executor/models/transformers/fx_utils.py`（模块 图工具；类别 `source`；类型 `core-logic`；符号 `_infer_len`, `_rank`, `_SizedProxy`, `len`）：融合器核心基础设施，提供 `torch.fx` 追踪、AST 重写工具（`compile_forward`, `recover_forward`, `replace_expr` 等）
- `vllm/model_executor/models/transformers/fusers/base.py`（模块 融合器；类别 `source`；类型 `core-logic`；符号 `BaseFuser`, `info`, `match`, `validate`）：定义 `BaseFuser` 和 `StackedFuser` 抽象基类，规定融合器接口
- `vllm/model_executor/models/transformers/fusers/qkv.py`（模块 融合器；类别 `source`；类型 `core-logic`；符号 `QKVFuser`, `shards`, `_get_qkv_nodes`, `match`）：QKV 融合器，将 `q`, `k`, `v` 三个线性层合并为 `QKVParallelLinear`，并重写 `forward`
- `vllm/model_executor/models/transformers/fusers/moe.py`（模块 融合器；类别 `source`；类型 `core-logic`；符号 `named_state`, `_own_returns`, `_returns_tuple`, `yields_tuple`）：MoE 融合器，将 HF MoE 块替换为 vLLM 的 `FusedMoE`，支持异步共享专家和序列并行
- `vllm/model_executor/models/transformers/fusers/rms_norm.py`（模块 融合器；类别 `source`；类型 `core-logic`；符号 `_is_squared`, `_variance_eps`, `_is_one_plus`, `_has_trailing_compute`）：RMSNorm 融合器，结构检测并替换为 vLLM 的 `fused RMSNorm`，支持张量并行
- `vllm/model_executor/models/transformers/fusers/glu.py`（模块 融合器；类别 `source`；类型 `core-logic`；符号 `GLUFuser`, `shards`, `_is_act_of_gate`, `_get_glu_nodes`）：GLU 融合器，将 `act(gate(x)) * up(x)` 融合为单个 `MergedColumnParallelLinear`
- `vllm/model_executor/models/transformers/fuser.py`（模块 融合器；类别 `source`；类型 `core-logic`；符号 `get_fuser`, `Fusers`, `init`, `getitem`）：融合器注册与分派入口，负责收集所有可用融合器并暴露给模型加载
- `vllm/model_executor/models/transformers/base.py`（模块 模型后端；类别 `source`；类型 `core-logic`；符号 `vllm_flash_attention_forward`, `_get_decoder_cls`, `register_fusion`）：Transformers 建模后端的主入口，修改以支持融合自动检测和递归替换
- `tests/models/transformers/fusers/test_linear.py`（模块 测试；类别 `test`；类型 `test-coverage`；符号 `SiluAndMulStub`, `forward`, `NoDownGLU`, `init`）：QKV 和 GLU 融合器的测试，覆盖多种模式（GQA、偏置条件等）
- `tests/models/transformers/fusers/test_moe.py`（模块 测试；类别 `test`；类型 `test-coverage`；符号 `TopKRouter`, `init`, `forward`, `CorrectionRouter`）：MoE 融合器的测试，包括多种路由器和专家配置
- `tests/models/transformers/fusers/test_rms_norm.py`（模块 测试；类别 `test`；类型 `test-coverage`；符号 `RMSNorm`, `init`, `_rms`, `forward`）：RMSNorm 融合器的测试，包括普通 RMSNorm、Gemma 风格、非 RMSNorm 等

关键符号: QKVFuser._get_qkv_nodes, QKVFuser.match, QKVFuser.update_forward, GLUFuser._get_glu_nodes, GLUFuser.match, GLUFuser.update_forward, MoEBlockFuser.match, MoEBlockFuser.fuse, RMSNormFuser.match, RMSNormFuser.fuse, trace, recover_forward, compile_forward, replace_expr, recursive_replace, get_fuser

关键源码片段

vllm/model_executor/models/transformers/fx_utils.py

融合器核心基础设施, 提供 torch.fx 追踪、AST 重写工具 (compile_forward, recover_forward, replace_expr 等)

```
# vllm/model_executor/models/transformers/fx_utils.py
```

```
import ast
import inspect
import textwrap
from collections.abc import Callable
from torch import fx, nn
```

```
class _SizedProxy(fx.Proxy):
    """Proxy whose len is inferred from the graph."""
    def __len__(self) -> int:
        length = _infer_len(self.node)
        if length is None:
            return super().__len__()
        return length
```

```
class _AllLeafTracer(fx.Tracer):
    """Tracer that treats every submodule as a leaf."""
    def is_leaf_module(self, m: nn.Module, module_qualified_name: str) -> bool:
        return True
    def proxy(self, node: fx.Node) -> fx.Proxy:
        return _SizedProxy(node, self)
```

```
def trace(module: nn.Module) -> fx.Graph | None:
    """Trace module.forward, returning the partial graph on failure."""
    try:
        return _AllLeafTracer().trace(module)
    except Exception:
        return None
```

```
def recover_forward(cls: type[nn.Module]) -> tuple[ast.FunctionDef, str]:
    """Return the AST function def and original source of cls.forward."""
    source = textwrap.dedent(inspect.getsource(inspect.unwrap(cls.forward)))
    return ast.parse(source).body[0], source
```

```
def compile_forward(funcdef: ast.FunctionDef) -> Callable:
```

```

"""Compile funcdef into a callable function."""
code = compile(ast.fix_missing_locations(funcdef), '<fuser>', 'exec')
ns = {}
exec(code, ns)
return ns[funcdef.name]

```

```

def replace_expr(tree: ast.AST, old: ast.AST, new: ast.AST) -> bool:
    """Replace old sub-expression with new in tree, return True if replaced."""
    for node in ast.walk(tree):
        for field, value in ast.iter_fields(node):
            if isinstance(value, list):
                for i, item in enumerate(value):
                    if item is old:
                        value[i] = new
                        return True
            elif value is old:
                setattr(node, field, new)
                return True
    return False

```

... 更多工具函数如 is_linear, find_node, peel, single_self_call 等

vllm/model_executor/models/transformers/fusers/base.py

定义 BaseFuser 和 StackedFuser 抽象基类，规定融合器接口

```
# vllm/model_executor/models/transformers/fusers/base.py
```

```

from abc import ABC, abstractmethod
from dataclasses import dataclass
from typing import TYPE_CHECKING, ClassVar
from torch import fx, nn

```

```
@dataclass
```

```
class BaseFuser(ABC):
```

```
    source_cls: str # HF 模块的类名，用于日志
```

```
    @abstractmethod
```

```
    def info(self, name: str) -> str:
```

```
        """返回融合的描述信息，如 'Fused: q + k + v (MyAttention) -> qkv_proj (QKVParallelLinear)
        """
```

```
    @classmethod
```

```
    @abstractmethod
```

```
    def match(cls, graph: fx.Graph, module: nn.Module) -> 'BaseFuser | None':
```

```
        """在图和模块中搜索模式，返回匹配的融合器实例或 None。"""
```

```
    @abstractmethod
```

```
    def validate(self, module: nn.Module, model_config: 'ModelConfig') -> bool:
```

```
        """检查模块实例是否满足融合条件（如输入输出维度、偏置等）。"""
```

```

@abstractmethod
def fuse(self, module: nn.Module, prefix: str, model_config: 'ModelConfig',
        quant_config: 'QuantizationConfig') -> nn.Module:
    """对验证通过的模块执行融合，返回替换后的模块。"""

@dataclass
class StackedFuser(BaseFuser):
    """融合多个并列线性层为一个堆叠的线性层（如 QKV、GLU）的基类。"""
    merged_name: ClassVar[str] # 融合后模块的属性名
    merged_cls: ClassVar[str] # 目标 vLLM 类名
    fused_forward: Callable = field(init=False, repr=False)

    def fuse(self, module, prefix, model_config, quant_config):
        # 1. 更新前向函数：用 AST 重写 forward 源码
        self.update_forward(type(module))
        # 2. 替换原有线性层为单个 MergedColumnParallelLinear / QKVParallelLinear
        # 3. 绑定编译后的 forward 到模块实例
        # 详见各个子类的 update_forward 和 update_attrs 方法

```

vllm/model_executor/models/transformers/fusers/qkv.py

QKV 融合器，将 q, k, v 三个线性层合并为 QKVParallelLinear，并重写 forward

```
# vllm/model_executor/models/transformers/fusers/qkv.py
```

```

@dataclass
class QKVFuser(StackedFuser):
    q_name: str
    k_name: str
    v_name: str
    o_name: str | None # 可选的 o_proj
    merged_name: ClassVar[str] = "qkv_proj"
    merged_cls: ClassVar[str] = "QKVParallelLinear"

    @property
    def shards(self) -> list[tuple[str, ShardId]]:
        return [(self.q_name, "q"), (self.k_name, "k"), (self.v_name, "v")]

    @classmethod
    def _get_qkv_nodes(cls, graph: fx.Graph, module: nn.Module) -> tuple[fx.Node, fx.Node, fx.
Node] | None:
        """搜索图，找到三个共享同一 placeholder 输入的线性调用，即为 q, k, v。"""
        by_input: dict[fx.Node, list[fx.Node]] = {}
        for node in graph.nodes:
            if (is_linear(node, module) and len(node.args) == 1
                and not node.kwargs and isinstance(node.args[0], fx.Node)
                and node.args[0].op == "placeholder"):
                by_input.setdefault(node.args[0], []).append(node)
        triples = [nodes for nodes in by_input.values() if len(nodes) == 3]
        if len(triples) != 1:

```

```

        return None
    q_node, k_node, v_node = triples[0]
    # 处理 GQA 情况: q 的 out_features 与其他两个不同
    outs = [module.get_submodule(n.target).out_features for n in (q_node, k_node, v_node)]
    if len(set(outs)) == 2:
        # 找到唯一的那个作为 q
        q_node = next(n for n, o in zip(triples[0], outs) if outs.count(o) == 1)
        k_node, v_node = (n for n, o in zip(triples[0], outs) if outs.count(o) == 2)
        if module.get_submodule(q_node.target).out_features != max(outs):
            return None
    elif len(set(outs)) != 1:
        return None
    return q_node, k_node, v_node

@classmethod
def match(cls, graph, module):
    if (nodes := cls._get_qkv_nodes(graph, module)) is None:
        return None
    q, k, v = nodes
    names = dict(q_name=q.target, k_name=k.target, v_name=v.target)
    # 尝试识别 o_proj
    attn_width = module.get_submodule(q.target).out_features
    candidates = [name for name, child in module.named_children()
                   if isinstance(child, nn.Linear)
                   and name not in names.values()
                   and child.in_features == attn_width]
    names['o_name'] = candidates[0] if len(candidates) == 1 else None
    return cls(source_cls=type(module).__name__, **names)

def update_forward(self, module: nn.Module) -> None:
    """通过 AST 重写 forward: 将 q(x), k(x), v(x) 替换为 qkv(x).split(sizes, -1)"""
    # 1. 获取 forward 的 AST
    funcdef, fn = recover_forward(type(module))
    # 2. 验证三个调用的参数相同
    # 3. 用 innermost_block 找到每个调用的所在块
    # 4. 在第一个调用处替换为 qkv = self.qkv_proj(x); q, k, v = qkv.split(...)
    # 5. 删除后两个调用
    # 6. 编译并绑定到实例

```

评论区精华

主要讨论围绕融合器的扩展性。Isotr0py 建议支持 Deepseek MLA 的 `qkv_a_proj` 和 Qwen3.5 GDN 的 `qkvz_proj` 融合，但作者 hmellor 认为这些需要特殊处理，且扫描其他模型后影响低，故留作未来工作。此外，Reviewer 建议将测试组织到 `tests/models/transformers/` 目录下，该建议已被采纳（PR 中测试已放置在该目录）。

- 扩展融合到 Deepseek MLA 和 Qwen3.5 GDN (design): 当前 PR 不包含 MLA/GDN 融合，未来考虑。

- 测试目录结构和文件拆分 (testing): hmellor 已采纳, PR 中测试已按建议组织。

风险与影响

- 风险: 主要风险包括:
 - 图匹配误报: QKVFuser 和 GLUFuser 的图模式匹配可能错误匹配非目标结构, 导致运行时错误。代码通过 validate 和 AST 检查进行双重验证, 但仍有遗漏风险。
 - AST 重写稳定性: 对 forward 源码的 AST 操作依赖于 Python 语法树的具体结构, 不同模型或 Transformers 版本可能语法不同, 导致重写失败或生成错误代码。
 - RMSNorm TP aware 广播: TPAwareNormMixin 在所有张量并行 ranks 上收集完整隐藏状态再归一化, 可能增加通信开销, 影响大模型扩展性。
 - MoE 融合兼容性: MoEBlockFuser 假设了标准的路由模式 (如 topk + softmax), 非标准路由 (如容量因子、带偏置) 可能不适用。
- 影响: 影响范围广泛但依赖模型选择:
 - 性能: 基准测试显示在密集 / 滑动注意力和 MoE 模型上达到或超越原生 vLLM 性能, 尤其在高并发场景下。
 - 可用性: 用户可继续使用 Transformers 模型定义, 无需为 vLLM 编写专门的模型代码, 降低接入门槛。tp_plan 和 pp_plan 可选简化了配置。
 - 维护负担: 融合器的引入增加了新的代码路径和复杂度, 但长期可能减少因模型差异导致的 bug 修复工作。
 - 测试覆盖: 新增大量单元测试, 确保核心模式匹配的正确性。
 - 风险标记: 核心路径变更, 复杂图分析, AST 重写依赖, TP 通信开销, 非标准模型兼容

关联脉络

- PR #47329 [Refactor] Remove multiple dead code: 同为 Transformers 建模后端相关, 进行了代码清理, 为融合器提供了更干净的基座