

PR #48498 完整报告

vllm-project/vllm

[Performance] Add Triton kernel for Gemma3n sparse GELU

合并时间: 2026-09-14 15:42

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/48498>

执行摘要

- 一句话: Gemma3n 稀疏 GELU 接入 Triton 融合内核
- 推荐动作: 值得精读, 尤其是三处设计决策: (1) kernel 中 dtype roundtrip 的必要性论证——用 ablation 数据 (14/24 FP16、21/24 BF16 超差与数百至数千处 mask 不一致) 证明“为对齐 native 舍入而牺牲部分算力”是正确取舍; (2) @triton_op + wrap_triton 让 torch.compile 保持对 kernel launch 的可见性, 并用 standalone AOT 回归守护上下游依赖不被裁掉, 这是 IR 算子接 Triton 的可复用范式; (3) warp 启发式与 spill 的权衡, 以及在缺乏全局最优解时“保留现状 + 记录 TODO”的工程判断。后续建议: 补一轮 H100/L40S 等架构的 warp/spill 验证再考虑是否扩大默认启用范围; 与 #41903 收敛同一 op 的 IR 注册避免重复; 把 d=32768 的 spill TODO 转成独立 issue 跟踪。

功能与动机

PR body 明确目标是为 Gemma3n 的 `gelu_and_mul_sparse` 提供专用 Triton provider, 把逐行统计、Gaussian thresholding、tanh GELU 与 gated multiplication 融合, 避开多 kernel 的 native/Inductor 路径; 同时把该 op 注册进 vLLM IR 并通过 IR dispatch 路由。这也是 issue #32676 (用 PluggableLayer / vLLM IR 替换 CustomOp) 中激活类 CustomOp 迁移清单里 `gelu_and_mul_sparse` 的一项。作者声明本 PR 与 #41903 在 IR 注册上存在重叠, 但侧重点不同: 本 PR 是性能导向, 附带 Triton provider、CUDA 优先级配置、AOT 依赖修复与跨架构 benchmark, 并已请 #41903 排除对该 op 的重复迁移。

实现拆解

1. 把激活逻辑提升为 vLLM IR 算子

- 新增 `vllm/ir/ops/activation.py`: 用 `@register_op` 定义 `gelu_and_mul_sparse(x, std_multiplier, approximate)`, 逐行计算 `mean/std (unbiased=False)`, 做 `F.relu(gate - (mean + std * std_multiplier))` 后再 `F.gelu(...) * x[..., d:]`; `_gelu_and_mul_sparse_input_generator` 统一生成 `(x, std_multiplier, approximate)`, 供测试与 benchmark 复用。
- 原 `GeluAndMulSparse._gaussian_topk` 中“TP>1 不做 all-gather, 统计量保持分片局部”的语义说明被搬到 IR 实现中, 计算语义不变。
- `vllm/ir/ops/__init__.py` 增加导出, 使 `ir.ops.gelu_and_mul_sparse` 可被 dispatch 与优先级管理。

2. 图层退化为 dispatch 层（数据契约变化）

- vllm/model_executor/layers/activation.py: 删除 `_gaussian_topk` 与手工拆分, `forward_native` 只剩 `ir.ops.gelu_and_mul_sparse(x, self.std_multiplier, self.approximate)`, `forward_cuda` 继续转发。
- `self.std_multiplier` 从 `normal_dist.icdf(...)` 的 0 维 tensor 改为 `.item()` 的 Python float, 匹配 IR op 的标量签名, 也让 `torch.compile` / AOT 能把它当编译期常量捕获。
- ROCm 上 `approximate == "tanh"` 回退为 `"none"` 的既有逻辑保留, 配合 Triton provider 只接受 `"tanh"` 的 `supports_args`, 形成天然的平台隔离。

3. 新增 Triton provider (vllm/kernels/triton/activation.py, +110)

- `_gelu_and_mul_sparse_kernel`: 一个 program 处理一行, gate 归约出 mean/variance, 按 `mean + std * std_multiplier` 计算 cutoff, `tl.where` 截断负值后走 tanh 近似 GELU (用 `2 * sigmoid(2x) - 1` 实现 tanh), 最后与 up 相乘写回; 行索引用 `tl.int64` 防溢出。
- 每个算子边界都插入输入 dtype roundtrip (`to(input_dtype).to(tl.float32)`), 复现 native PyTorch 的舍入, 否则 FP16/BF16 的数值与稀疏 mask 都会偏离。
- `_supports_gelu_and_mul_sparse` 作为 fallback 闸门: 要求 CUDA、contiguous、ndim > 0、dtype ∈ {FP16, BF16, FP32}、`approximate == "tanh"`、最后一维为偶数且 `d <= 32768`, 不满足时自动回落 native。
- `_gelu_and_mul_sparse_triton_op` 用 `@triton_op("vllm::gelu_and_mul_sparse_triton")` + `wrap_triton` 包装 launch, 使 `torch.compile` 能看到真实 kernel 而不把整段当黑盒, 从而保住 standalone AOT 的依赖关系。
- `num_warps` 由固定 8 改为 `min(max(block_size // 256, 1), 8)` (review 后修改, 生产 `d >= 2048` 仍是 8 warps)。

4. provider 注册、默认优先级与平台差异

- vllm/kernels/triton/__init__.py 导入 activation, vllm/kernels/__init__.py 导入 triton 子包并写入 `__all__`; import vllm.kernels 的副作用即完成注册, 测试与 benchmark 都依赖这一点。
- `register_impl("triton", supported=HAS_TRITON and current_platform.is_cuda(), supports_args=...)`: 无 Triton 或非 CUDA 时 provider 直接不可用。
- vllm/platforms/cuda.py / rocm.py / xpu.py 与 vllm/config/kernel.py: CUDA 默认 priority 为 `["triton", "native"]`, ROCm/XPU 为 `["native"]`; commit d151c957 专门修掉 Inductor 关闭时平台默认值误指向未注册 vllm_c provider 导致的启动失败。

5. 测试、benchmark 与 AOT 配套

- tests/kernels/ir/test_activation.py (+183): 覆盖 provider registration、3 dtype × 3 token 数 × 6 个 intermediate size 的正确性与 mask 比例校验、第二个随机种子、小尺寸 (`d = 1/31/128`, 三 dtype, 要求 mask 精确一致)、ND/ 空张量、常量行、NaN/Inf gate、fallback 条件、dispatch + opcheck、CUDA Graph 回放。
- tests/compile/passes/ir/test_lowering.py: 新增 `SparseActivationModel` 与 `test_lowering_gelu_and_mul_sparse`, 断言 lowering 后 `selected_impls ==`

`{"gelu_and_mul_sparse": "triton"}.`

- `tests/compile/test_aot_compile.py`: 新增 `SparseActivationAotModel` (`dynamic_arg_dims={"x": 0}`) 与 `test_sparse_activation_standalone_aot_preserves_dependencies`, 用 native provider 手工复算 golden, 守护 AOT 产物不丢上游 projection 与 Triton write。
- `tests/test_config.py`: `test_ir_op_platform_defaults_support_sparse_gelu` 覆盖 compilation mode x backend 组合下的平台默认 provider。
- `benchmarks/kernels/benchmark_gelu_and_mul_sparse.py`: 7 个 token 数 x 6 个 intermediate size = 42 组配置, 对比 Triton / `native(torch.compile)` / `native(eager)`, 且计时前强制与 native 做正确性对照。

关键文件:

- `vllm/kernels/triton/activation.py` (模块 稀疏激活内核; 类别 source; 类型 core-logic; 符号 `_gelu_and_mul_sparse_kernel`, `_supports_gelu_and_mul_sparse`, `_gelu_and_mul_sparse_triton_op`, `gelu_and_mul_sparse_triton`): PR 的核心: 新增融合统计量、高斯阈值化、tanh GELU 与门控乘法的 Triton kernel, 并通过 `triton_op + wrap_triton` 注册为 IR 的 triton provider; warp 启发式、dtype roundtrip、`supports_args fallback` 均在此文件。
- `vllm/model_executor/layers/activation.py` (模块 激活算子; 类别 source; 类型 data-contract; 符号 `GeluAndMulSparse.init`, `GeluAndMulSparse.forward_native`): `GeluAndMulSparse` 从自带 native 实现改为只做 IR dispatch, 删除 `_gaussian_topk`; `std_multiplier` 由 0 维 tensor 变为 Python float, 是 IR op 标量签名与编译期常量捕获的关键契约变化。
- `tests/kernels/ir/test_activation.py` (模块 内核测试; 类别 test; 类型 test-coverage; 符号 `test_gelu_and_mul_sparse_registration`, `test_gelu_and_mul_sparse_triton`, `test_gelu_and_mul_sparse_triton_second_seed`, `test_gelu_and_mul_sparse_triton_small_sizes`): 新增 9 组用例, 覆盖 provider 注册、三 dtype x 多尺寸正确性与 mask 比例、第二随机种子、小尺寸精确 mask、ND/ 空张量、常量行、NaN/Inf、fallback 条件、dispatch + opcheck 与 CUDA Graph, 是本次数值正确性的主要防线。
- `vllm/ir/ops/activation.py` (模块 算子注册; 类别 source; 类型 data-contract; 符号 `gelu_and_mul_sparse`, `_gelu_and_mul_sparse_input_generator`): 定义 `gelu_and_mul_sparse` IR 算子, 承载从图层搬出的 native 参考实现与输入生成器, 是 provider 优先级、lowering 与 AOT 测试共同依赖的契约层。
- `tests/compile/test_aot_compile.py` (模块 编译测试; 类别 test; 类型 test-coverage; 符号 `SparseActivationAotModel`, `SparseActivationAotModel.forward`, `test_sparse_activation_standalone_aot_preserves_dependencies`, `native_reference`): 新增 `SparseActivationAotModel` 与 standalone AOT 依赖保持回归, 专门守护 AOT 产物不丢上游 projection 或 Triton write, 是编译链路唯一的结构性防线。
- `tests/compile/passes/ir/test_lowering.py` (模块 降级测试; 类别 test; 类型 test-coverage; 符号 `SparseActivationModel`, `SparseActivationModel.forward`, `test_lowering_gelu_and_mul_sparse`): 新增 `test_lowering_gelu_and_mul_sparse`, 断

言 VllmIRLoweringPass 在 CUDA 上为该 op 选中 triton provider，验证 IR 优先级到 lowering 的端到端链路。

- benchmarks/kernels/benchmark_gelu_and_mul_sparse.py (模块 性能基准; 类别 other; 类型 benchmark; 符号 benchmark_gelu_and_mul_sparse): 新增可复用的 42 组 shape 基准 (7 token 数 × 6 intermediate size), 对比 Triton / native(torch.compile) / native(eager), 并在计时前强制与 native 对照, 是后续调参与回归的基线工具。
- vllm/platforms/cuda.py (模块 平台适配; 类别 source; 类型 configuration): CUDA 平台默认 IR op priority 加入 triton 前置 (配合 vllm/config/kernel.py), 是该 kernel 默认生效的开关; ROCm/XPU 同名改动保持 native-only, 修复未注册 provider 的启动问题。
- tests/test_config.py (模块 配置测试; 类别 test; 类型 test-coverage; 符号 test_ir_op_platform_defaults_support_sparse_gelu): 新增 test_ir_op_platform_defaults_support_sparse_gelu, 覆盖 compilation mode × backend 组合下的平台默认 provider, 防止 ROCm/XPU 再次继承未注册的 vllm_c provider。
- vllm/kernels/__init__.py (模块 内核注册; 类别 source; 类型 entrypoint): 顶层 kernels 包导入 triton 子包 (triton/init.py 再导入 activation), 使 import vllm.kernels 的副作用即完成 provider 注册; 测试、benchmark 与 AOT 用例都依赖这条注册链。

关键符号: _gelu_and_mul_sparse_kernel, _supports_gelu_and_mul_sparse, _gelu_and_mul_sparse_triton_op, gelu_and_mul_sparse_triton, GeluAndMulSparse.forward_native, gelu_and_mul_sparse, _gelu_and_mul_sparse_input_generator, SparseActivationAotModel.forward, native_reference, test_sparse_activation_standalone_aot_preserves_dependencies, test_lowering_gelu_and_mul_sparse, test_ir_op_platform_defaults_support_sparse_gelu, benchmark_gelu_and_mul_sparse

关键源码片段

vllm/model_executor/layers/activation.py

GeluAndMulSparse 从自带 native 实现改为只做 IR dispatch, 删除 _gaussian_topk; std_multiplier 由 0 维 tensor 变为 Python float, 是 IR op 标量签名与编译期常量捕获的关键契约变化。

```
class GeluAndMulSparse(CustomOp):
    """Gemma3n 的稀疏 GELU: 按行做高斯 top-k 稀疏后再 GELU, 并与 up 分支相乘."""

    def __init__(self, activation_sparsity: float, approximate: str = "none"):
        super().__init__()
        self.approximate = approximate
        if approximate not in ("none", "tanh"):
            raise ValueError(f"Unknown approximate mode: {approximate}")
        if current_platform.is_rocm() and approximate == "tanh":
            # ROCm 上 torch.compile + tanh GELU 不稳定, 保留原有回退:
            # 回退成 "none" 后 Triton provider 的 supports_args 会因 approximate 不匹配而拒绝,
            # 于是自动落到 native, 形成隐式但安全的平台隔离。
```

```

logger.warning_once(
    "[ROCM] Pytorch's native GELU with tanh approximation is currently "
    "unstable and produces garbage. Fallback to 'none' approximation."
)
self.approximate = "none"

if activation_sparsity == 0.0:
    raise ValueError("activation_sparsity is 0.0. Please use GeluAndMul.")
target_sparsity_tensor = torch.tensor(activation_sparsity, dtype=torch.float32)
normal_dist = torch.distributions.normal.Normal(0, 1)
# 从 0 维 tensor 改为 Python float: IR op 签名是 (x, std_multiplier: float, ...),
# 标量才能在 torch.compile / AOT 阶段被稳定捕获为编译期常量。
self.std_multiplier = normal_dist.icdf(target_sparsity_tensor).item()

def forward_native(self, x: torch.Tensor) -> torch.Tensor:
    # 具体实现搬到 vllm/ir/ops/activation.py 的 native provider:
    # 这里只做 IR dispatch, 由默认优先级 (CUDA 上 ["triton", "native"]) 决定实现,
    # 原来的 _gaussian_topk + F.gelu + 乘法被删除, 避免双份维护。
    return ir.ops.gelu_and_mul_sparse(x, self.std_multiplier, self.approximate)

def forward_cuda(self, x: torch.Tensor) -> torch.Tensor:
    return self.forward_native(x)

```

tests/compile/test_aot_compile.py

新增 SparseActivationAotModel 与 standalone AOT 依赖保持回归, 专门守护 AOT 产物不丢上游 projection 或 Triton write, 是编译链路唯一的结构性防线。

```

@support_torch_compile(dynamic_arg_dims={"x": 0}) # token 维动态, AOT 产物需覆盖多 shape
class SparseActivationAotModel(torch.nn.Module):
    def __init__(self, **kwargs):
        super().__init__()
        self.up_proj = torch.nn.Linear(32, 256, bias=False)
        # GeluAndMulSparse(0.95, "tanh") 在 CUDA 上默认解析到 triton provider
        self.activation = GeluAndMulSparse(0.95, "tanh")
        self.down_proj = torch.nn.Linear(128, 16, bias=False)

    def forward(self, x: torch.Tensor):
        return self.down_proj(self.activation(self.up_proj(x)))

@pytest.mark.skipif(
    not current_platform.is_cuda() or not is_torch_equal_or_newer("2.10.0"),
    reason="requires CUDA and torch 2.10",
)
def test_sparse_activation_standalone_aot_preserves_dependencies(
    monkeypatch: pytest.MonkeyPatch, vllm_tmp_cache: Path
):
    def native_reference(model: SparseActivationAotModel, x: torch.Tensor):
        # 用 native provider 手工复算 golden: up_proj -> native 稀疏 GELU -> down_proj

```

```

projected = torch.nn.functional.linear(x, model.up_proj.weight)
activated = ops.gelu_and_mul_sparse.impls["native"].impl_fn(
    projected, model.activation.std_multiplier, "tanh"
)
return torch.nn.functional.linear(activated, model.down_proj.weight)

```

with monkeypatch.context() as m:

```

# 打开 standalone AOT 编译与 mega artifact, 并隔离缓存目录
m.setenv("VLLM_USE_AOT_COMPILE", "1")
m.setenv("VLLM_USE_MEGA_AOT_ARTIFACT", "1")
m.setenv("VLLM_USE_STANDALONE_COMPILE", "1")
m.setenv("VLLM_CACHE_ROOT", str(vllm_tmp_cache / "vllm_cache"))

```

```

vllm_config = make_vllm_config()
torch.manual_seed(0)
# 同一编译产物要覆盖两种 token 数, 验证 dynamic shape 下的依赖保持
inputs = [
    torch.randn(4, 32, device="cuda", dtype=torch.bfloat16),
    torch.randn(7, 32, device="cuda", dtype=torch.bfloat16),
]

```

```

with (
    use_vllm_config(vllm_config),
    vllm_config.kernel_config.ir_op_priority.set_priority(),
    ir.enable_torch_wrap(True),
):
    compiled = SparseActivationAotModel(vllm_config=vllm_config).to(
        device="cuda", dtype=torch.bfloat16
    )
    compiled.requires_grad_(False)
    expected = [native_reference(compiled, x) for x in inputs]
    actual = [compiled(x) for x in inputs]

```

```

for result, reference in zip(actual, expected):
    # 若 AOT 产物丢掉上游 projection 或 Triton write, 这里会直接不 finite 或对不上
    assert torch.isfinite(result).all()
    torch.testing.assert_close(
        result,
        reference,
        **ops.gelu_and_mul_sparse.get_tolerance(result.dtype),
    )

```

评论区精华

1. 固定 8 warps 的跨架构泛化 (BadrBasowid, vllm/kernels/triton/activation.py)

“Is fixed `num_warps=8` optimal across the supported CUDA architectures and all intermediate sizes? ... If this was tuned for one specific GPU, should the optimized implementation be restricted to that hardware until broader results are available?”

结论：作者改为 `num_warps = min(max(block_size // 256, 1), 8)`；相对固定 8 warps，小维度几何均值在 A100 提升 1.022x、B200 提升 1.134x，`d >= 2048` 的生产 shape 保持 8 warps，故不加架构白名单。

2. dtype roundtrip 是否必要 (BadrBasowid, line 45)

“Are the dtype roundtrips necessary?”

结论：必要。作者给出 ablation：A100 上移除后 14/24 个 FP16、21/24 个 BF16 用例超差，并出现 281 / 2700 处稀疏 mask 不一致；B200 上 25 例失败、3292 处不一致。kernel 中已补注释说明意图。

3. 容差口径与小尺寸覆盖 (BadrBasowid → BWAAEEEEK)

“can you report tolerances separately for FP32, FP16, and BF16.” / “what about fp16 and fp32?”

结论：补充 dtype 容差表 (FP32 atol 1e-5、FP16/BF16 atol 1e-3、BF16 rtol 1.6e-2，最大绝对误差 1.43e-6 / 0 / 0)，小尺寸测试参数化到三种 dtype 并增加稀疏 mask 精确断言，activation 套件 74/74 通过。

4. spill 与延迟 up 加载 (BadrBasowid ⇌ BWAAEEEEK)

“I also say this because i am worried about spills on archs that haven't been tested” / “Perhaps you can just add a TODO ...”

结论：作者在 B200 全 `d` 范围尝试多种 `up` 加载位置：紧随 mean 之后加载无回退但未降低 spill；延迟到 variance 之后可把 SM80 `d=32768` 的 stack frame 从 1312B 降到 592B，却在部分 B200 shape 回退 2-3%。最终保留现实现并留下 TODO，属未完全解决、显式延后的项。

5. 并发场景端到端表现 (issue 评论, tjtanana)

“how does it fair with various concurrencies e.g. 1, 64, 128? Is this always faster even when the number of tokens are large?”

结论：A100 在 C1/C64/C128 均落在统计噪声内；B200 C64 快 3.100% ($p \approx 1.1e-10$)、C128 快 2.089% ($p = 0.00357$)，C1 中性；配合 CUDA Graph 下 GSM8K 准确率 0.5307 (native 0.5262)，无质量回退。

6. ROCm/XPU 默认 provider 回归 (commit d151c957 + tests/test_config.py) CI 暴露出：Inductor 关闭时平台默认优先级会继承到未注册的 `vllm_c` provider 导致启动失败；修复为只有 CUDA 默认 `["triton", "native"]`，并新增 `test_ir_op_platform_defaults_support_sparse_gelu` 回归。审查方 tjtanana 最终给定 “LGTM now.”。

- 固定 `num_warps=8` 的跨架构适配性与是否需要硬件白名单 (performance): 改为 `num_warps = min(max(block_size // 256, 1), 8)`，生产 `d >= 2048` 仍为 8 warps；小维度几何均值 A100 +1.022x、B200 +1.134x；`d=32768` 在 A100 有 48B spill 但仍是实测最快配置，故不加架构限制。
- dtype roundtrip 是否必要 (数值与稀疏 mask 的对齐) (correctness): roundtrip 必需，用于复现 native PyTorch 在每个算子边界的输入 dtype 舍入；作者在 kernel 内补充了说明

注释, reviewer 接受。

- 分 dtype 容差报告与小尺寸测试的 dtype 覆盖 (testing): 作者给出容差表 (FP32 atol 1e-5、FP16/BF16 atol 1e-3、BF16 rtol 1.6e-2, 最大绝对误差 1.43e-6 / 0 / 0), 小尺寸测试参数化到三种 dtype 并增加稀疏 mask 精确相等断言, activation 套件在两个架构上各 74/74 通过。
- 大 reduction 下的寄存器 spill 与延迟 up 加载 (performance): 延迟到 variance 之后可把 SM80 d=32768 的 stack frame 从 1312B 降到 592B, 但会让部分 B200 shape 回退 2-3%, 没有普遍更优的放置方式, 决定保留现实现并在 kernel 中留下 TODO 作为后续跟进。
- 不同并发度下的端到端收益与长 token 场景表现 (performance): A100 三档并发均在统计噪声内; B200 C64 快 3.100% (p 约 1.1e-10)、C128 快 2.089% (p=0.00357), C1 中性; 长输入场景中位延迟 150.14 ms → 147.81 ms (1.58%), greedy token 与 logprobs 完全一致。
- ROCm/XPU 默认 provider 误指向未注册 vllm_c 的 CI 回归 (correctness): 最终 CUDA 默认 ["triton", "native"]、ROCm/XPU 默认 ["native"], 并新增 test_ir_op_platform_defaults_support_sparse_gelu 覆盖 compilation mode × backend 组合; tjtanana 最终批准 (LGTM now)。

风险与影响

- 风险:
 - 核心路径默认行为变更: CUDA 上 Gemma3n 的 GeluAndMulSparse 从 native 默认切换到 Triton, 波及所有 CUDA 用户; 验证只覆盖 SM80 (A100) 与 SM100 (B200), H100、L40S、消费级卡与未来架构未测, reviewer 对未测架构 spill 的担忧最终只以 TODO 收尾。
 - 已知未解决项: d=32768 在 A100 存在 48 字节 spill, 虽然实测仍是最快配置, 但属于记录在案的性能债; TODO 注释未绑定 issue。
 - 数值容差偏宽: BF16 rtol 放宽到 1.6e-2; 主正确性用例只要求 mask 不一致比例 < 1e-3 (仅小尺寸要求精确相等), 长行真实输入下 mask 抖动会向 down_proj 传播, 端到端有统计兜底 (GSM8K 无退化) 但缺逐 token 的严格保证。
 - 与 native 实现的隐式耦合: kernel 明确“复现输入 dtype 在每个 native 算子边界的舍入”, 若 IR native provider 将来调整算子顺序 (如先取负再算 cutoff), Triton 侧必须同步, 否则数值与 mask 会再次漂移。
 - 编译链路依赖: triton_op + wrap_triton 的可追踪性依赖 PyTorch 版本行为, standalone AOT 若再次丢失上游 projection 或 Triton write, 只会静默产生错误结果; 现有回归只覆盖 CUDA 且 torch >= 2.10。
 - 平台默认值维护面: cuda/rocm/xpu 三处默认 priority 各自维护, ROCm 因 __init__ 强制 approximate="none" 而天然被 supports_args 挡住, 行为安全但属于隐性耦合; std_multiplier 改为 float 后, 任何把它当 tensor 使用的外部代码会失效 (仓库内已全部对齐)。
- 影响:

- 用户影响: Gemma3n 系列 (如 E2B/E4B) 在 CUDA 上默认获得融合内核, kernel 级 A100 几何均值 2.59x (相对 compiled native)、range 1.15x–4.06x; 端到端 A100 持平, B200 长输入 (2048 token) 中位延迟从 150.14 ms 降到 147.81 ms (1.58%), 高并发 C64/C128 吞吐提升 2–3%, GSM8K 准确率与非法率均无回退。
- 系统影响: 新增 vllm/kernels/triton/ 命名空间并确立“Triton provider = triton_op + wrap_triton + register_impl + supports_args + AOT 测试”的接入模板, 直接服务 issue #32676 / PR #41903 的激活算子 IR 化路线, 后续 fatrelu_and_mul、swigluoai_and_mul、relu2 等可复用同一模式。
- 团队影响: 跨平台默认 priority 多了一个需要同步维护的点 (CUDA 走 triton、ROCm/XPU 走 native), kernel 内留下一个 spill 优化 TODO, 需要转成独立跟进项; 新增 benchmark 脚本为后续调参与回归提供统一基线。
- 风险标记: 核心激活路径默认变更, 跨架构验证不足 (仅 SM80/SM100), 已知寄存器 spill 未解决, BF16 容差偏宽 (rtol 1.6e-2), IR 注册与 #41903 重叠

关联脉络

- PR #41903 [vLLM IR] port activation ops (fatrelu_and_mul, relu2, gelu_and_mul_sparse, swigluoai_and_mul): 同一 op 的 IR 注册来源, 本 PR body 明确记录与其重叠, 并请求其在保留其余激活迁移的同时排除 gelu_and_mul_sparse, 是近期最需要收敛的关联项。
- PR #56676 [Bugfix] Skip Triton autotune inspection without Triton: 同属 Triton 可用性边界问题: 本 PR 依赖 HAS_TRITON 门控 provider 注册与 vllm/kernels/triton 导入链, 与无 Triton 环境下的导入 / 回退处理是同一类风险面。
- PR #56707 [CI] Initialize warmup registry in V2 QSA runner fixture: 同属 kernels/JIT 注册副作用引发的下游 CI 问题: 本 PR 也要求测试显式 import vllm.kernels 才能完成 provider 注册, 仓库后续为同类注册缺失补过夹具。
- PR #55768 [Warmup] Gemma 4 de-JITification: 同属把模型侧 kernel 收敛到集中式内核与 JIT 基础设施的演进路线, 均涉及 vllm/kernels 与 attention 目录的内核组织方式。
- PR #56688 [Perf] Avoid redundant conversions in FP8 dummy initialization: 同类性能优化但取舍相反: 该 PR 消除冗余 dtype 拷贝, 本 PR 则刻意保留 dtype roundtrip 以对齐 native 数值, 可对比理解“必要的转换”与“冗余转换”的判定标准。