PR #22094 完整报告

sgl-project/sglang

[JIT Kernel] Reland JIT activation

合并时间：2026-04-25 14:00

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/22094

执行摘要

- 一句话：重新引入 JIT 激活内核，修复 num_token=0 边界问题
- 推荐动作：值得精读，尤其是 _fast_math_flags 的设计权衡、custom op 的注册方式、以及如何通过条件导入保持向后兼容。对于需要维护多平台支持的开发者，本 PR 展示了如何用 JIT 替代部分 AOT 组件。

功能与动机

关联 Issue #22078 因 CI 失败回退了初版 JIT activation。本 PR 修复了导致失败的 num_token=0 边界 case，并重新提交。JIT activation 可以避免对 sgl-kernel 特定编译版本的依赖，在更多 GPU 架构（如 Blackwell SM100+）上灵活控制编译选项，并为后续扩展提供统一入口。

实现拆解

1. 添加 CUDA kernel: python/sglang/jit_kernel/csrc/elementwise/activation.cuh 定义 ActivationKernel<T, kUsePDL> 类模板，支持 SiLU、GELU、GELU_Tanh 三种激活，使用向量化访存和 PDL 延迟加载。
2. 创建 Python 封装: python/sglang/jit_kernel/activation.py 通过 load_jit 编译 kernel，导出 silu_and_mul、gelu_and_mul、gelu_tanh_and_mul 函数，并注册 custom op _run_activation_inplace。_fast_math_flags 根据平台条件添加 --use_fast_math 标志（SM100+ 和 ROCm 禁用）。
3. 修改现有模块的导入路径: 在 gguf.py、cutlass_w4a8_moe.py、fused_moe_triton/triton_kernels_moe.py、fused_marlin_moe.py、cutlass_moe.py、srt/layers/activation.py、multimodal_gen/runtime/layers/activation.py 等文件中，将 CUDA 分支下激活函数的导入从 sgl_kernel 切换为 sglang.jit_kernel.activation，并作 _is_cuda / _is_musa 条件判断，确保多平台兼容。
4. 添加单元测试: python/sglang/jit_kernel/tests/test_activation.py 包含 test_activation_correctness 和 test_activation_out_param，在多种形状和 dtype 下验证数值正确性（与 PyTorch 参考实现对比），并测试外部 out 参数路径。
5. 添加性能基准: python/sglang/jit_kernel/benchmark/bench_activation.py 对比 AOT（sgl-kernel）、JIT（本模块）和 torch.compile 实现的延迟，基于 Triton 测试框架生成性能报告。

6. 后续精度修复：多个 commit 由合并者 ch-wan 提交，包括：修复 custom op fake 实现输出形状（out_shape="input" 应匹配实际输出形状）、禁用 Blackwell 上的 --use_fast_math、调整 GELU Tanh 中 alpha 乘法的顺序以匹配浮点结果、将 gate*up 乘法提升到 float32 精度以对齐 flashinfer。

关键文件：

- python/sglang/jit_kernel/activation.py（模块 JIT 核心；类别 source；类型 core-logic；符号 _fast_math_flags, _jit_activation_module, _run_activation_inplace, run_activation）：核心 JIT 激活函数封装，定义编译入口、fast_math 策略及公开 API
- python/sglang/jit_kernel/tests/test_activation.py（模块 测试；类别 test；类型 test-coverage；符号 _reference, _tolerances, test_activation_correctness, test_activation_out_param）：单元测试，覆盖三种激活、多种 dtype 和 shape，验证正确性及 out 参数传递
- python/sglang/srt/layers/quantization/gguf.py（模块 GGUF 集成；类别 source；类型 dependency-wiring）：修改 GGUF 量化模块导入路径，将 CUDA 上的激活函数切换为 JIT 版本
- python/sglang/jit_kernel/benchmark/bench_activation.py（模块 基准测试；类别 source；类型 core-logic；符号 benchmark, silu_and_mul, gelu_and_mul, gelu_tanh_and_mul）：添加性能基准，对比 AOT、JIT 与 torch.compile 三种实现
- python/sglang/jit_kernel/csrc/elementwise/activation.cuh（模块 CUDA 内核；类别 other；类型 core-logic；符号 ActivationKernel）：CUDA kernel 实现，包含三种激活的 fused kernel 和向量化访存
- python/sglang/srt/layers/moe/cutlass_w4a8_moe.py（模块 Cutlass MoE；类别 source；类型 dependency-wiring）：修改 Cutlass W4A8 MoE 的激活函数导入路径

关键符号：_fast_math_flags, _jit_activation_module, _run_activation_inplace, run_activation, silu_and_mul, gelu_and_mul, gelu_tanh_and_mul, benchmark, _reference, test_activation_correctness, test_activation_out_param

关键源码片段

python/sglang/jit_kernel/activation.py

核心 JIT 激活函数封装，定义编译入口、fast_math 策略及公开 API

```
from __future__ import annotations
from typing import TYPE_CHECKING, Optional
import torch

from sglang.jit_kernel.utils import (
    cache_once,
    get_jit_cuda_arch,
    is_arch_support_pdl,
    is_hip_runtime,
    load_jit,
    make_cpp_args,
```

```
)
from sglang.srt.utils.custom_op import register_custom_op

if TYPE_CHECKING:
    from tvm_ffi.module import Module


def _fast_math_flags() -> list[str]:
    # 平台条件判断：ROCm 和 Blackwell (SM100+) 禁用 --use_fast_math
    # 因为 Blackwell 需要精确的 expf，且 triton 在 ROCm 下使用 clang 编译会出错
    if is_hip_runtime():
        return []
    if get_jit_cuda_arch().major >= 10:
        return []
    return ["--use_fast_math"]


@cache_once
def _jit_activation_module(dtype: torch.dtype) -> Module:
    args = make_cpp_args(dtype, is_arch_support_pdl())
    return load_jit(
        "activation",
        *args,
        cuda_files=["elementwise/activation.cuh"],
        extra_cuda_cflags=_fast_math_flags(),
        cuda_wrappers=[
            ("run_activation", f"ActivationKernel<{args}>::run_activation"),
        ],
    )


SUPPORTED_ACTIVATIONS = {"silu", "gelu", "gelu_tanh"}


@register_custom_op(mutates_args=["out"])
def _run_activation_inplace(op_name: str, input: torch.Tensor, out: torch.Tensor) -> None:
    # 视图变换为 (batch, hidden*2) 和 (batch, hidden) 后调用编译后的 kernel
    hidden_size = input.shape[-1] // 2
    module = _jit_activation_module(input.dtype)
    input_2d = input.view(-1, hidden_size * 2)
    out_2d = out.view(-1, hidden_size)
    module.run_activation(input_2d, out_2d, op_name)


def run_activation(op_name: str, input: torch.Tensor, out: Optional[torch.Tensor] = None) ->
torch.Tensor:
    assert op_name in SUPPORTED_ACTIVATIONS, f"Unsupported activation: {op_name}"
    hidden_size = input.shape[-1] // 2
    if out is None:
```

```
        out = input.new_empty(*input.shape[:-1], hidden_size)
    _run_activation_inplace(op_name, input, out)
    return out


def silu_and_mul(input: torch.Tensor, out: Optional[torch.Tensor] = None) -> torch.Tensor:
    return run_activation("silu", input, out)


def gelu_and_mul(input: torch.Tensor, out: Optional[torch.Tensor] = None) -> torch.Tensor:
    return run_activation("gelu", input, out)


def gelu_tanh_and_mul(input: torch.Tensor, out: Optional[torch.Tensor] = None) -> torch.
Tensor:
    return run_activation("gelu_tanh", input, out)
```

python/sglang/jit_kernel/tests/test_activation.py

单元测试，覆盖三种激活、多种 dtype 和 shape，验证正确性及 out 参数传递

```
import sys
import pytest
import torch
import torch.nn.functional as F

from sglang.jit_kernel.activation import SUPPORTED_ACTIVATIONS, run_activation
from sglang.jit_kernel.utils import get_ci_test_range
from sglang.test.ci.ci_register import register_cuda_ci

register_cuda_ci(est_time=20, suite="stage-b-kernel-unit-1-gpu-large")
register_cuda_ci(est_time=30, suite="nightly-kernel-1-gpu", nightly=True)

OPS = SUPPORTED_ACTIVATIONS
DTYPES = [torch.float16, torch.bfloat16, torch.float32]
SHAPES = get_ci_test_range(
    full_range=[
        (7, 16),
        (83, 1024),
        (3, 5, 16),
        (2, 3, 512),
        (1, 17, 4096),
        *[(2**x, 2048) for x in range(0, 15, 2)],
        *[(2**x, 65536) for x in range(0, 5, 2)],
    ],
    ci_range=[(7, 16), (2, 3, 512)],
)


def _reference(op_name: str, x: torch.Tensor) -> torch.Tensor:
```

```
    # 参考实现：使用 PyTorch 的官方激活函数，在 float32 下计算并缩放 up
    d = x.shape[-1] // 2
    lhs = x[..., :d].float()
    rhs = x[..., d:]
    if op_name == "silu":
        act = F.silu(lhs)
    elif op_name == "gelu":
        act = F.gelu(lhs, approximate="none")
    else:
        act = F.gelu(lhs, approximate="tanh")
    return act.to(dtype=x.dtype) * rhs


def _tolerances(dtype: torch.dtype) -> tuple[float, float]:
    # float32 使用更严格容差，半精度使用 1e-2
    if dtype == torch.float32:
        return 1e-4, 1e-4
    return 1e-2, 1e-2


@pytest.mark.parametrize("op_name", OPS)
@pytest.mark.parametrize("dtype", DTYPES)
@pytest.mark.parametrize("shape", SHAPES)
def test_activation_correctness(op_name, dtype, shape):
    x = torch.randn(shape, dtype=dtype, device="cuda")
    out = run_activation(op_name, x, None)
    expected = _reference(op_name, x)
    atol, rtol = _tolerances(dtype)
    torch.testing.assert_close(out, expected, atol=atol, rtol=rtol)


@pytest.mark.parametrize("op_name", OPS)
@pytest.mark.parametrize("dtype", DTYPES)
@pytest.mark.parametrize("shape", SHAPES)
def test_activation_out_param(op_name, dtype, shape):
    # 验证传入外部 out 张量时，结果正确且返回的是同一对象
    x = torch.randn(shape, dtype=dtype, device="cuda")
    out = torch.empty(shape[:-1] + (shape[-1] // 2,), dtype=dtype, device="cuda")
    result = run_activation(op_name, x, out)
    assert result is out
    expected = _reference(op_name, x)
    atol, rtol = _tolerances(dtype)
    torch.testing.assert_close(out, expected, atol=atol, rtol=rtol)
```

python/sglang/srt/layers/quantization/gguf.py

修改 GGUF 量化模块导入路径，将 CUDA 上的激活函数切换为 JIT 版本

```
# 在 gguf.py 中，根据平台条件导入激活函数
if _is_cuda:
```

```
    from sgl_kernel import moe_align_block_size, moe_sum
    from sgl_kernel.quantization import (
        ggml_dequantize,
        ggml_moe_a8,
        ggml_moe_a8_vec,
        ggml_moe_get_block_size,
        ggml_mul_mat_a8,
        ggml_mul_mat_vec_a8,
    )
    # CUDA 平台使用 JIT 编译的激活函数，替代 sgl-kernel 的 AOT 版本
    from sglang.jit_kernel.activation import gelu_and_mul, silu_and_mul
elif _is_musa:
    from sgl_kernel import gelu_and_mul, moe_align_block_size, moe_sum, silu_and_mul
    from sgl_kernel.quantization import (
        ggml_dequantize,
        ggml_moe_a8,
        ggml_moe_a8_vec,
        ggml_moe_get_block_size,
        ggml_mul_mat_a8,
        ggml_mul_mat_vec_a8,
    )
else:
    if not _is_hip:
        warnings.warn(f"Only CUDA and MUSA support GGUF quantization currently.")

# fused_moe_gguf 内部的 act 辅助函数也相应简化
def act(x: torch.Tensor):
    # 直接调用简化后的接口，无需手动管理临时张量
    if activation == "silu":
        return silu_and_mul(x)
    elif activation == "gelu":
        return gelu_and_mul(x)
    raise ValueError(f"Unsupported activation: {activation}")
```

评论区精华

> Review 讨论 (ch-wan): 在 activation.py 中，--use_fast_math 标志是否应该全局启用？DarkSharpness 后续在 commit 中实现条件禁用：SM100+ (Blackwell) 和 ROCm 上关闭，其余开启。结论：在 _fast_math_flags 函数中依据平台和架构版本动态决定，平衡性能与精度。 PR 普通评论 (ch-wan): "I fixed some numerical issues. Could you check my changes?" — 合并者发现了多个数值不一致问题，包括 custom op fake shape、GELU multiply 顺序、gate*up 精度等，并逐个提交修复，确保了 JIT 版本与 sgl-kernel 的输出在测试容忍度内一致。

- 关于 --use_fast_math 的使用策略 (performance): 采纳条件禁用方案，在 _fast_math_flags 中实现。
- 数值精度不一致修复 (correctness): 所有修复已合并，JIT 版本与 sgl-kernel 在测试容忍度内一致。

风险与影响

- 风险：
  1. 回归风险：替换多个模块的激活函数导入，如果条件判断错误（如 MUSA 平台意外使用了 JIT 版本），可能导致编译失败或行为异常。
  2. 精度风险：尽管有测试保障，但 --use_fast_math 在非 Blackwell 上的开启可能引入微小数值差异，尤其在混合精度场景下。
  3. 性能风险：JIT 编译首次调用有额外开销；benchmark 显示 JIT 版本与 AOT 版本性能接近，但在某些 shape 上可能略逊。
  4. 兼容性风险：非 CUDA 平台（MUSA, XPU, HIP）仍需 sgl_kernel 路径，修改后 gguf.py 等文件的导入区分了 _is_cuda 和 _is_musa，需确保所有分支均正确。 - 影响：用户影响：模型输出应保持一致，首次调用可能增加几秒 JIT 编译时间。系统影响：统一了激活函数实现，便于后续添加新激活函数。团队影响：减少对 sgl-kernel 二进制包的依赖，使新 GPU 架构（如 Blackwell）无需等待 sgl-kernel 预编译即可运行。 - 风险标记：跨模块依赖变更，精度敏感，多平台条件分支，JIT 编译首次性能开销

关联脉络

- PR #22078 Revert "[Feature] JIT activation and update skills (by codex)": 本 PR 重新引入了被 #22078 回退的 JIT activation 功能，并修复了导致回退的 num_token=0 问题。