PR #42663 完整报告

vllm-project/vllm

[6/n] Migrate activation kernels, gptq, gguf, non cutlass w8a8 to libtorch stable ABI (continued)

合并时间：2026-05-20 15:18

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42663

执行摘要

本 PR 是 vLLM 向 PyTorch libtorch stable ABI 迁移的第六阶段，将激活内核、W8A8 INT8/FP8 量化、GPTQ、GGML/GGUF 从传统 _C 扩展迁移至 _C_stable_libtorch 扩展，同时启用 ROCm 平台支持。新增了独立的错误检查头文件 csrc/torch_utils.h，解除了对 CUTLASS 的依赖。该 PR 合并后，构建 libtorch-ABI-stable CUDA wheels 的进度向前迈进一大步。

功能与动机

PyTorch 团队正在推动 libtorch ABI 稳定（参见 Issue #26946），使自定义算子扩展可以跨 PyTorch 版本运行。vLLM 是 PyTorch 生态中的关键下游，希望通过迁移内核到 stable ABI，使得构建的 wheel 不再绑定到特定 PyTorch 版本，从而简化构建系统、降低维护成本并改善开发者体验。本 PR 是系列迁移中的第 6 阶段，处理了剩余的非 CUTLASS 内核。

实现拆解

1. 统一错误检查宏：新建 csrc/torch_utils.h，根据是否定义 TORCH_TARGET_VERSION 选择使用 STD_TORCH_CHECK（stable 路径）或 TORCH_CHECK（传统路径），且不依赖 CUTLASS/CUTE，使得 dtype_fp8.cuh 等头文件可以安全包含。
2. 移动源文件：将 csrc/activation_kernels.cu、csrc/quantization/w8a8/int8/scaled_quant.cu、csrc/quantization/w8a8/fp8/common.cu、csrc/quantization/gptq/q_gemm.cu、csrc/quantization/gguf/gguf_kernel.cu 等文件移至 csrc/libtorch_stable/ 对应子目录，并更新 CMakeLists.txt 编译目标。
3. 迁移算子声明与注册：在 csrc/libtorch_stable/ops.h 新增 stable API 声明，在 csrc/libtorch_stable/torch_bindings.cpp 中使用 STABLE_TORCH_LIBRARY_FRAGMENT 定义 schema 并用 STABLE_TORCH_LIBRARY_IMPL + TORCH_BOX 注册 CUDA 实现。
4. 清理旧扩展注册：从 csrc/torch_bindings.cpp 和 csrc/ops.h 中删除已迁移算子的注册与声明，保留尚未迁移的 silu_and_mul_per_block_quant（因其依赖不稳定）和 CPU 构建需要的声明。
5. 启用 ROCm 支持：在 CMake 中调整编译守卫，使 _C_stable_libtorch 同时支持 CUDA 和 HIP，将 CUDA 独占源文件（CUTLASS、FP4、AWQ、permute_cols）放入 #ifndef USE_ROCM 块内。
6. 测试验证：在 PR body 中提供了 test_activation.py、test_ggml.py、test_fp8_quant.py、test_int8_quant.py 的通过截图。

csrc/libtorch_stable/torch_bindings.cpp

该文件是 stable ABI 扩展的算子注册中心，本 PR 为其新增了激活、W8A8 量化、GPTQ、GGML 等所有迁移算子的 schema 定义和 CUDA 实现注册，是迁移的核心目标。

```
// 在 STABLE_TORCH_LIBRARY_FRAGMENT 中定义算子 schema（稳定 ABI 版本）
STABLE_TORCH_LIBRARY_FRAGMENT(_C, ops) {
  // ... 已有定义 ...
  // 激活算子：SwiGLU、GeGLU、GELU 变体等
  ops.def("silu_and_mul(Tensor! result, Tensor input) -> ()");
  ops.def("mul_and_silu(Tensor! out, Tensor input) -> ()");
  ops.def("gelu_and_mul(Tensor! out, Tensor input) -> ()");
  // ...

  // W8A8 INT8 量化
  ops.def(
      "static_scaled_int8_quant(Tensor! result, Tensor input, Tensor scale,"
      "Tensor? azp) -> ()");
  ops.def(
      "dynamic_scaled_int8_quant(Tensor! result, Tensor input, Tensor! scale, "
      "Tensor!? azp) -> ()"); // 注意：schema 中 Tensor!? 应改为 Tensor?

  // W8A8 FP8 量化（支持 per-tensor / per-channel / per-token）
  ops.def(
      "static_scaled_fp8_quant(Tensor! result, Tensor input, Tensor scale, "
      "int[]? group_shape=None) -> ()");
  ops.def(
      "dynamic_scaled_fp8_quant(Tensor! result, Tensor input, Tensor! scale) "
      "-> ()");
  // GPTQ
  ops.def(
      "gptq_gemm(Tensor a, Tensor b_q_weight, Tensor b_gptq_qzeros, "
      "Tensor b_gptq_scales, Tensor b_g_idx, bool use_exllama, bool "
      "use_v2_format, int bit) -> Tensor");
  ops.def("gptq_shuffle(Tensor! q_weight, Tensor q_perm, int bit) -> ()");
  // GGML
  ops.def(
      "ggml_dequantize(Tensor W, int type, SymInt m, SymInt n, ScalarType? "
      "dtype) -> Tensor");
  // ...
}

// 注册 CUDA 实现（共享 CUDA/ROCm）
STABLE_TORCH_LIBRARY_IMPL(_C, CUDA, ops) {
#ifndef USE_ROCM
  // CUTLASS 相关实现仍由 CUDA 独占
  // ...
#endif
  // 激活内核（共享 CUDA/ROCm）
```

```
  ops.impl("silu_and_mul", TORCH_BOX(&silu_and_mul));
  ops.impl("silu_and_mul_with_clamp", TORCH_BOX(&silu_and_mul_clamp));
  ops.impl("mul_and_silu", TORCH_BOX(&mul_and_silu));
  // 量化内核
  ops.impl("static_scaled_int8_quant", TORCH_BOX(&static_scaled_int8_quant));
  ops.impl("dynamic_scaled_int8_quant", TORCH_BOX(&dynamic_scaled_int8_quant));
  // GPTQ
  ops.impl("gptq_gemm", TORCH_BOX(&gptq_gemm));
  ops.impl("gptq_shuffle", TORCH_BOX(&gptq_shuffle));
  // GGML
  ops.impl("ggml_dequantize", TORCH_BOX(&ggml_dequantize));
  // ...
}
```

csrc/torch_utils.h

新建的独立头文件，将 TORCH_UTILS_CHECK 宏从 cutlass_extensions 中剥离出来，使 attention/quant 等模块可以安全使用而不引入 CUTLASS 依赖，是本次迁移的基础设施改进。

```
#pragma once

// Shared TORCH_UTILS_CHECK across both libtorch stable and unstable source
// files. Keep this header free of CUTLASS/CUTE so attention/quant headers can
// use it.
//
// If TORCH_TARGET_VERSION is defined, we are building _C_stable_libtorch.so so
// use STD_TORCH_CHECK via header-only.
// Otherwise, use TORCH_CHECK via torch/all.h.

#ifdef TORCH_TARGET_VERSION
  #include <torch/headeronly/util/Exception.h>
  #define TORCH_UTILS_CHECK STD_TORCH_CHECK
#else
  #include <torch/all.h>
  #define TORCH_UTILS_CHECK TORCH_CHECK
#endif
```

评论区精华

- silu_and_mul_clamp 声明缺失：gemini-code-assist[bot] 发现 ops.h 中缺少 silu_and_mul_clamp 的声明，作者确认已补充。
- CMakeLists.txt 中 fused 内核文件归属错误：机器人指出 fused_silu_mul_block_quant.cu 等文件仍属于 _C 扩展，作者承认并最终将其移回非稳定扩展以避免依赖冲突。
- STD_TORCH_CHECK 宏定义争议：janeyx99 质疑在 dtype_fp8.cuh 中直接检查 STD_TORCH_CHECK 定义的做法，讨论后作者创建了独立的 csrc/torch_utils.h 统一处理。
- Schema Tensor!? 语法错误：机器人指出 dynamic_scaled_int8_quant 的 schema 中 Tensor!? 是不合法序列，应改为 Tensor?。

风险与影响

- 功能回归风险：内核迁移等价变换，但 stable API 的替代实现（如 DeviceGuard、get_current_cuda_stream）可能存在细微差异，需依赖测试覆盖。
- 构建失败风险：CMakeLists.txt 中的目标文件列表必须精确，否则链接错误。已有 AMD CI 修复记录。
- 性能风险：torch::stable::Tensor 在热路径上可能有微小开销，但计算密集型内核应无影响。
- 兼容性风险：未来编译需要 PyTorch 版本包含 stable API 头文件。
- 影响范围：直接影响 vLLM 构建系统，为 ABI 稳定 wheel 分发铺路；对最终用户无功能可见差异。

关联脉络

- #38757：本 PR 是该 PR 的延续，处理了剩余内核和 ROCm 适配。
- #26946：PyTorch 团队发起的 RFC，目标是构建 libtorch-ABI-stable vLLM wheels。
- 跨 PR 演进：从早期的迁移阶段（如 #38671）开始，逐步将 attention、CUTLASS 等模块向 stable ABI 迁移。本 PR 完成后，_C_stable_libtorch 已覆盖除 fused 量化内核外的主要算子。