PR #43361 完整报告

vllm-project/vllm

[8/n] Migrate merge_attn_states, mamba, sampler to torch stable ABI (continued)

合并时间：2026-05-28 00:35

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43361

执行摘要

本 PR 是 torch stable ABI 迁移系列的第八阶段，将 merge_attn_states、Mamba selective_scan 以及 sampler（top-k、repetition penalties）等 CUDA 内核从旧扩展 csrc/ 迁移到稳定扩展 csrc/libtorch_stable/。通过使用 torch::stable::Tensor 等稳定类型替换 ATen 类型，_C.abi3.so 中的不稳定符号从 99 减少到 98。此次迁移不改变内核逻辑，但提高了构建稳定性和 torch 版本兼容性。

功能与动机

延续 PR #38841 的目标：将 vLLM 中所有 CUDA 算子迁移到 torch stable ABI，以减少不稳定符号数量，使扩展更易维护且兼容未来 torch 版本。PR body 中 torch-abi-audit 的对比直观展示了改进。由于 main 分支已有较大变动（如 topk 改用 persistent_topk），提交需要手动重写部分文件。

实现拆解

1. 文件搬迁：将 merge_attn_states.cu、sampler.cu、topk.cu、selective_scan_fwd.cu 及其头文件从 csrc/ 移动到 csrc/libtorch_stable/ 对应子目录。
2. 声明迁移：在 csrc/libtorch_stable/ops.h 中添加稳定 ABI 声明（使用 torch::stable::Tensor），在 csrc/ops.h 中删除旧声明。
3. 注册迁移：在 csrc/libtorch_stable/torch_bindings.cpp 的 STABLE_TORCH_LIBRARY_FRAGMENT 中通过 ops.def() 和 ops.impl() 注册算子，并在旧绑定文件中删除对应注册。
4. 内核适配：在各 .cu 文件中替换类型（torch::Tensor → torch::stable::Tensor）和宏（TORCH_CHECK → STD_TORCH_CHECK），调整 device_guard 并添加 const 限定。
5. 构建配置：更新 CMakeLists.txt 使新文件参与稳定扩展编译；在 pyproject.toml 中添加 sharedMemPerBlockOptin 到拼写检查白名单。

csrc/libtorch_stable/torch_bindings.cpp

核心绑定注册文件，在此添加了 merge_attn_states、sampler、mamba 的稳定 ABI 定义和实现注册，是迁移的核心入口。

```
// csrc/libtorch_stable/torch_bindings.cpp
// 在 STABLE_TORCH_LIBRARY_FRAGMENT 中添加以下注册：

// Merge attn states（合并注意力状态）
```

```
// 实现 https://www.arxiv.org/pdf/2501.01005 第 2.2 节
// 用于在 split-KV 情况下合并部分注意力结果
ops.def(
    "merge_attn_states("
    "    Tensor! output,"
    "    Tensor!? output_lse,"
    "    Tensor prefix_output,"
    "    Tensor prefix_lse,"
    "    Tensor suffix_output,"
    "    Tensor suffix_lse,"
    "    int!? prefill_tokens_with_context,"
    "    Tensor? output_scale=None) -> ()");

// 在 CUDA impl 段中绑定实现
ops.impl("merge_attn_states", TORCH_BOX(&merge_attn_states));

// Apply repetition penalties（应用重复惩罚）
ops.def(
    "apply_repetition_penalties_(Tensor! logits, Tensor prompt_mask, "
    "Tensor output_mask, Tensor repetition_penalties) -> ()");
ops.impl("apply_repetition_penalties_", TORCH_BOX(&apply_repetition_penalties_));

// Optimized top-k per row（每行 top-k 优化）
ops.def(
    "top_k_per_row_prefill(Tensor logits, Tensor rowStarts, Tensor rowEnds, "
    "Tensor! indices, int numRows, int stride0, "
    "int stride1, int topK) -> ()");
ops.impl("top_k_per_row_prefill", TORCH_BOX(&top_k_per_row_prefill));

ops.def(
    "top_k_per_row_decode(Tensor logits, int next_n, "
    "Tensor seq_lens, Tensor! indices, "
    "int numRows, int stride0, int stride1, int topK) -> ()");
ops.impl("top_k_per_row_decode", TORCH_BOX(&top_k_per_row_decode));

ops.def(
    "persistent_topk(Tensor logits, Tensor lengths, Tensor! output, "
    "Tensor workspace, int k, int max_seq_len) -> ()");
ops.impl("persistent_topk", TORCH_BOX(&persistent_topk));

// Mamba selective scan 前向内核
ops.def(
    "selective_scan_fwd(Tensor! u, Tensor! delta,"
    "Tensor! A, Tensor! B, Tensor! C,"
    "Tensor? D_, Tensor!? z_, Tensor? delta_bias_,"
    "bool delta_softplus,"
    "Tensor? query_start_loc,"
    "Tensor? cache_indices," // ... 更多参数省略
    ") -> ()");
```

```
ops.impl("selective_scan_fwd", TORCH_BOX(&selective_scan_fwd));
```

csrc/torch_bindings.cpp

旧绑定注册文件，移除了已迁移的 merge_attn_states、sampler 和 mamba 注册，只保留尚未迁移的量化等操作。

```
// csrc/torch_bindings.cpp（删除的片段）
// 以下注册被整体移除，迁移至稳定库：

/* 删除前：
ops.def(
    "merge_attn_states("
    " Tensor! output,"
    ...
    " Tensor? output_scale=None) -> ()");
ops.impl("merge_attn_states", torch::kCUDA, &merge_attn_states);

ops.def("apply_repetition_penalties_(...)");
ops.impl("apply_repetition_penalties_", torch::kCUDA, &apply_repetition_penalties_);

ops.def("top_k_per_row_prefill(...)");
ops.impl("top_k_per_row_prefill", torch::kCUDA, &top_k_per_row_prefill);

ops.def("top_k_per_row_decode(...)");
ops.impl("top_k_per_row_decode", torch::kCUDA, &top_k_per_row_decode);

ops.def("persistent_topk(...)");
ops.impl("persistent_topk", torch::kCUDA, &persistent_topk);

ops.def("selective_scan_fwd(...)");
ops.impl("selective_scan_fwd", torch::kCUDA, &selective_scan_fwd);
*/
```

csrc/libtorch_stable/ops.h

稳定 ABI 头文件，新增了迁移内核的函数声明，使用 torch::stable::Tensor 等类型。

```
// csrc/libtorch_stable/ops.h（新增声明）

// Attention 内核（共享 CUDA/ROCm）
void merge_attn_states(
    torch::stable::Tensor& output,
    std::optional<torch::stable::Tensor> output_lse,
    const torch::stable::Tensor& prefix_output,
    const torch::stable::Tensor& prefix_lse,
    const torch::stable::Tensor& suffix_output,
    const torch::stable::Tensor& suffix_lse,
    const std::optional<int64_t> prefill_tokens_with_context,
    const std::optional<torch::stable::Tensor>& output_scale = std::nullopt);

// Sampler 内核（共享 CUDA/ROCm）
```

```
void apply_repetition_penalties_(
    torch::stable::Tensor& logits, const torch::stable::Tensor& prompt_mask,
    const torch::stable::Tensor& output_mask,
    const torch::stable::Tensor& repetition_penalties);

void top_k_per_row_prefill(const torch::stable::Tensor& logits,
    const torch::stable::Tensor& rowStarts,
    const torch::stable::Tensor& rowEnds,
    torch::stable::Tensor& indices, int64_t numRows,
    int64_t stride0, int64_t stride1, int64_t topK);

void top_k_per_row_decode(const torch::stable::Tensor& logits, int64_t next_n,
    const torch::stable::Tensor& seqLens,
    torch::stable::Tensor& indices, int64_t numRows,
    int64_t stride0, int64_t stride1, int64_t topK);

void persistent_topk(const torch::stable::Tensor& logits,
    const torch::stable::Tensor& lengths,
    torch::stable::Tensor& output,
    torch::stable::Tensor& workspace, int64_t k,
    int64_t max_seq_len);

void selective_scan_fwd(
    const torch::stable::Tensor& u, const torch::stable::Tensor& delta,
    const torch::stable::Tensor& A, const torch::stable::Tensor& B,
    const torch::stable::Tensor& C,
    const std::optional<torch::stable::Tensor>& D_,
    // ... 更多参数
);
```

评论区精华

- 常量正确性：janeyx99 与 cleonard530 讨论 lengths 参数应为 const，最终确认正确。
- 与原始 PR 的差异：cleonard530 说明 topk.cu 因主分支改用 persistent_topk 而手动重写，janeyx99 认可。
- DeviceGuard 使用：janeyx99 建议保持模块化，但未要求立即修改。

风险与影响

- 回归风险：内核逻辑无变化，但类型替换可能带来编译问题；已有测试覆盖。
- 兼容性：使用官方稳定类型，无额外风险。
- 构建风险：CMake 配置必须正确包含新位置文件，本 PR 已验证通过。
- 合并冲突：因多次手动重写，后续阶段需继续谨慎合并。

关联脉络

本 PR 是 torch stable ABI 迁移系列的第八阶段，直接继承 #38841 并基于 #43209。后续仍有 98 个不稳定符号需要迁移，预计会继续以类似模式进行。该系列体现了 vLLM 对构建系统

现代化和长期可维护性的投入。