PR #26134 完整报告

sgl-project/sglang

[refactor] unify cuda-graph capture/replay across attention backends

合并时间：2026-05-23 09:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26134

执行摘要

- 一句话：统一 5 个 attention 后端的 CUDA 图捕获 / 重放逻辑，消除重复代码并修复潜在 bug
- 推荐动作：值得精读。此 PR 展示了大型重构中消除重复的经典手法：提取公共方法、capture 委托 replay、工厂方法封装。对从事推理加速、CUDA 图优化、attention 模块开发的工程师有重要参考价值。建议在下次设计新的 attention 后端时，直接参考此 PR 总结的公共接口。

功能与动机

PR 描述指出 init_forward_metadata_capture_cuda_graph 和 init_forward_metadata_replay_cuda_graph 在每个 attention 后端中共享大量逻辑但各自维护独立副本，导致实际出现了分歧（如 WaveAttnBackend 跳过 get_num_kv_splits，CutlassMLABackend 含无用断言，FlashInferAttnBackend 存在约 240 行重复的模式分派代码）。统一逻辑可消除这些隐患，降低维护成本。

实现拆解

1. 提取公共辅助函数：在 TritonBackend 中新增 _fill_kv_indptr_and_indices、_update_decode_kv_buffers、_update_target_verify_buffers、_update_draft_extend_buffers、_build_cuda_graph_forward_metadata 等方法，将分散在 capture 和 replay 中的重复缓冲填充逻辑集中到带文档字符串的独立方法中，便于后续后端复用。
2. 采用 capture 委托 replay 模式（Pattern A）：对 WaveAttnBackend 和 CutlassMLABackend，capture 在非多步推测路径下直接调用 replay 完成缓冲写入，再通过 _build_cuda_graph_forward_metadata 冻结 ForwardMetadata。此举确保 capture 与 replay 使用完全一致的缓冲设置，修复了 WaveAttnBackend 中 get_num_kv_splits 在 capture 阶段缺失的问题。
3. 重构 FlashInferAttnBackend 为 prepare+replay 模式（Pattern B）：提取 _create_decode_wrappers 和 _create_prefill_wrappers 工厂方法负责包装器的构建（参数集中管理），新增 _prepare_cuda_graph_metadata 统一处理模式分派。capture 简化为调用 _prepare_cuda_graph_metadata 后调用 replay，不再包含分发分支。同时将 replay 方法中 is_target_verify 与 is_draft_extend 两条完全相同逻辑的分支合并。

4. 合并 FlashInferMLAAttnBackend 的重复分支：将该后端 capture 和 replay 方法中 is_target_verify 与 is_draft_extend 两个完全相同的分支分别合并为一个条件，减少代码量约 38 行。
5. 适配 CutlassMLABackend 并移除无用断言：采用 Pattern A，将 capture 委托给 replay，同时移除了 replay 中从未读取的 assert seq_lens_cpu is not None，使代码更简洁。

关键文件：

- python/sglang/srt/layers/attention/triton_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 _fill_kv_indptr_and_indices, _update_decode_kv_buffers, _update_target_verify_buffers, _update_draft_extend_buffers）：最核心的重构目标，新增 5 个辅助方法将 capture/replay 中的重复缓冲逻辑提取为独立函数，为其他后端提供复用模板。
- python/sglang/srt/layers/attention/flashinfer_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 init_forward_metadata_capture_cuda_graph, _create_decode_wrappers, _create_prefill_wrappers, _prepare_cuda_graph_metadata）：采用 Pattern B 重构，提取 _create_decode_wrappers 和 _create_prefill_wrappers 工厂方法，新增 _prepare_cuda_graph_metadata 统一模式分派，capture 简化为两行。
- python/sglang/srt/layers/attention/wave_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 _build_cuda_graph_forward_metadata, init_forward_metadata_capture_cuda_graph, init_forward_metadata_replay_cuda_graph）：采用 Pattern A，新增 _build_cuda_graph_forward_metadata 方法，capture 委托 replay 后调用此方法冻结元数据，修复 get_num_kv_splits 缺失 bug。
- python/sglang/srt/layers/attention/flashinfer_mla_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 init_forward_metadata_capture_cuda_graph, init_forward_metadata_replay_cuda_graph）：合并 target_verify 和 draft_extend 分支，消除完全相同逻辑的重复代码。
- python/sglang/srt/layers/attention/cutlass_mla_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 init_forward_metadata_capture_cuda_graph, init_forward_metadata_replay_cuda_graph）：采用 Pattern A，capture 委托 replay，并移除未使用的 assert。

关键符号：_fill_kv_indptr_and_indices, _update_decode_kv_buffers, _update_target_verify_buffers, _update_draft_extend_buffers, _build_cuda_graph_forward_metadata, update_sliding_window_buffer_cuda_graph, init_forward_metadata_capture_cuda_graph, _create_decode_wrappers, _create_prefill_wrappers, _prepare_cuda_graph_metadata

关键源码片段

python/sglang/srt/layers/attention/triton_backend.py

最核心的重构目标，新增 5 个辅助方法将 capture/replay 中的重复缓冲逻辑提取为独立函数，为其他后端提供复用模板。

```
# triton_backend.py - 新提取的缓冲填充辅助方法

def _fill_kv_indptr_and_indices(
    self,
    bs: int,
    seq_lens: torch.Tensor,
    req_pool_indices: torch.Tensor,
    kv_indices: torch.Tensor,
) -> torch.Tensor:
    """填充 kv_indptr 和 kv_indices 的通用方法，被 capture/replay 共用"""
    kv_indptr = self.kv_indptr[: bs + 1]
    kv_indptr[1:] = torch.cumsum(seq_lens, dim=0)
    # 使用 Triton 内核从 req_pool_indices 构建 flat kv_indices
    create_flashinfer_kv_indices_triton[(bs,)](
        self.req_to_token,
        req_pool_indices,
        seq_lens,
        kv_indptr,
        None,
        kv_indices,
        self.req_to_token.stride(0),
    )
    return kv_indptr

def _update_decode_kv_buffers(
    self,
    bs: int,
    seq_lens: torch.Tensor,
    req_pool_indices: torch.Tensor,
):
    """填充 decode/idle 模式下 CUDA 图所需的 KV 缓冲（含滑动窗口）"""
    seq_lens = seq_lens[:bs]
    req_pool_indices = req_pool_indices[:bs]
    kv_indptr = self._fill_kv_indptr_and_indices(
        bs, seq_lens, req_pool_indices, self.cuda_graph_kv_indices
    )
    window_kv_indptr = self.window_kv_indptr
    window_kv_lens = None
    if self.sliding_window_size is not None and self.sliding_window_size > 0:
        window_kv_indptr, _, window_kv_lens, _ = update_sliding_window_buffer(
            self.window_kv_indptr,
            self.req_to_token,
            self.sliding_window_size,
            seq_lens,
            req_pool_indices,
            bs,
            token_to_kv_pool_allocator=self.token_to_kv_pool_allocator,
            window_kv_indices=self.cuda_graph_window_kv_indices,
        )
```

```
    return kv_indptr, window_kv_indptr, window_kv_lens
```

python/sglang/srt/layers/attention/wave_backend.py

采用 Pattern A，新增 _build_cuda_graph_forward_metadata 方法，capture 委托 replay 后调用此方法冻结元数据，修复 get_num_kv_splits 缺失 bug。

```
# wave_backend.py - 从预分配缓冲构造 ForwardMetadata，capture 委托 replay

def _build_cuda_graph_forward_metadata(
    self,
    bs: int,
    forward_mode: ForwardMode,
    spec_info: Optional[SpecInput],
) -> ForwardMetadata:
    """从 CUDA 图的预分配缓冲中组装 ForwardMetadata（不执行任何核心计算）"""
    if forward_mode.is_decode_or_idle():
        return ForwardMetadata(
            attn_logits=self.cuda_graph_attn_logits,
            attn_lse=self.cuda_graph_attn_lse,
            max_extend_len=None,
            num_kv_splits=self.cuda_graph_num_kv_splits,
            kv_indptr=self.kv_indptr[: bs + 1],
            kv_indices=self.cuda_graph_kv_indices,
            qo_indptr=None,
            custom_mask=None,
            mask_indptr=None,
        )
    elif forward_mode.is_target_verify():
        return ForwardMetadata(
            attn_logits=None,
            attn_lse=None,
            max_extend_len=self.num_draft_tokens,
            num_kv_splits=None,
            kv_indptr=self.kv_indptr[: bs + 1],
            kv_indices=self.cuda_graph_kv_indices,
            qo_indptr=self.qo_indptr[: bs + 1],
            custom_mask=self.cuda_graph_custom_mask,
            mask_indptr=self.mask_indptr[: bs + 1],
        )
    else:
        raise ValueError(f"Invalid forward mode: {forward_mode=} for CUDA Graph.")

def init_forward_metadata_capture_cuda_graph(
    self,
    bs: int,
    num_tokens: int,
    req_pool_indices: torch.Tensor,
    seq_lens: torch.Tensor,
    encoder_lens: Optional[torch.Tensor],
```

```
    forward_mode: ForwardMode,
    spec_info: Optional[SpecInput],
):
    assert encoder_lens is None, "Not supported"
    # 多步推测路径：kv 缓冲来自 spec_info，不经过 replay 路径
    if forward_mode.is_decode_or_idle() and spec_info is not None:
        self.forward_metadata = ForwardMetadata(
            attn_logits=self.cuda_graph_attn_logits,
            attn_lse=self.cuda_graph_attn_lse,
            max_extend_len=None,
            num_kv_splits=self.cuda_graph_num_kv_splits,
            kv_indptr=spec_info.kv_indptr,
            kv_indices=spec_info.kv_indices,
            qo_indptr=None,
            custom_mask=None,
            mask_indptr=None,
        )
        return
    # 常规路径：委托给 replay 完成缓冲更新，然后用 _build_cuda_graph_forward_metadata
    冻结元数据
    self.init_forward_metadata_replay_cuda_graph(
        bs=bs,
        req_pool_indices=req_pool_indices,
        seq_lens=seq_lens,
        seq_lens_sum=None,
        encoder_lens=encoder_lens,
        forward_mode=forward_mode,
        spec_info=spec_info,
        seq_lens_cpu=None,
    )
    self.forward_metadata = self._build_cuda_graph_forward_metadata(
        bs, forward_mode, spec_info
    )
```

评论区精华

本 PR 在合并前无 reviewer 技术讨论。作者在描述中详细说明了每个后端的差异化修改，并通过在评论中执行 /rerun-test 手动触发多项 CI 测试（test_basic_sanity、test_mla_flashinfer、test_eagle_infer 等），所有测试结果均通过。此外，作者在 GB300 上对 CutlassMLABackend 和 WaveAttnBackend 进行了完整的精度与功能验证，确保重构后无回归。

- WaveAttnBackend 修复 get_num_kv_splits 在 capture 阶段缺失 (correctness): 通过将 capture 委托给 replay 并在之后调用 _build_cuda_graph_forward_metadata 来修复，GB300 验证通过。
- 多项 CI 测试确认正确性 (testing): CI 测试结果均为绿色，无失败用例。

风险与影响

- 风险：主要风险来自对 5 个 attention 后端核心 CUDA 图路径的大幅改动，尽管作者在 GB300 上完成了全面验证，但其他硬件平台（如 H100、AMD MI300 等）未经相同范围的覆盖。此外，还有多个后端（flashattention、trtllm 等）沿用旧的 Pattern C，未来统一时需注意接口和语义的一致性。整体风险可控，但建议在更多 CI runner 上补充回归测试。
- 影响：对用户无功能影响，推理结果完全一致。对开发团队，显著降低了 CUDA 图相关代码的维护成本，新增 attention 后端可以直接沿用已提取的公共模式。架构上为后续统一所有后端奠定了基础。团队需要继续完成 Pattern C 后端的迁移，并更新相关内部文档。
- 风险标记：核心路径变更，多个后端同步修改，缺少测试配套改动

关联脉络

- 暂无明显关联 PR