PR #29867 完整报告

sgl-project/sglang

feat(short-conv): shared ShortConvAttnBackend for ZAYA1 CCA + LFM2 short conv

合并时间：2026-07-02 11:08

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29867

执行摘要

- 一句话：提取短卷积状态后端，统一 ZAYA1 和 LFM2 状态管理
- 推荐动作：值得精读。该 PR 展示了如何优雅地解耦模型与状态管理，是注意力后端架构的良好扩展案例。重点关注 ShortConvAttnBackend 的索引解析策略、ShortConvHybridAttnBackend 的包装模式，以及如何通过 init_forward_metadata 避免每层重复计算。

功能与动机

多个混合短卷积模型（ZAYA1 CCA、LFM2 系列）均使用 MambaPool 管理 per-request 卷积状态，但状态解析逻辑（槽索引、前缀掩码、cuda-graph 缓冲区）高度重复。PR 旨在消除重复，降低新短卷积模型的接入成本，同时修复因状态索引延迟解析导致的 CUDA graph 非法内存访问和配置导入路径错误。

实现拆解

1. 新增 ShortConvAttnBackend 后端（short_conv_backend.py）：继承 MambaAttnBackendBase，提供 conv_state_metadata() 方法返回 ShortConvMetadata（包含 layer_cache、cache_indices、query_start_loc、has_initial_state 等）。索引解析提升到 init_forward_metadata / init_forward_metadata_out_graph 中，每步仅执行一次，避免层间重复。
2. 引入 ShortConvHybridAttnBackend 适配器（hybrid_linear_attn_backend.py）：作为 full-attn 后端的包装，暴露 conv_state_metadata 并委托给 ShortConvAttnBackend。
3. 重构模型文件（zaya.py、lfm2.py、lfm2_moe.py）：移除直接池访问和索引构建代码，改为调用 get_attn_backend().conv_state_metadata()。ZAYA1 的卷积核 (cca_extend/cca_decode) 移至 zaya.py 并使用返回的元数据。
4. 更新注意力注册表（attention_registry.py）：将 ZayaConfig、Lfm2Config、Lfm2MoeConfig 路由到新后端；修复了 Lfm2MoeConfig 和 Lfm2VlConfig 的导入路径错误（原从 configs.lfm2 导入，实际模块不同）。
5. 索引类型规范化（short_conv_backend.py、causal_conv1d.py）：将 cache_indices 改为 int64 主类型，在调用 causal_conv1d 时窄化为 int32，消除模型层的类型转换操作。
6. 测试更新（test_zaya_cca.py）：新增 _MockShortConvBackend 模拟后端行为，验证单步索引解析仅在首层执行。

关键文件：

- python/sglang/srt/layers/attention/linear/short_conv_backend.py（模块 注意力后端；类别 source；类型 dependency-wiring；符号 ShortConvMetadata, ShortConvAttnBackend, init, _reset_step_state）：新增核心后端，封装所有短卷积状态管理逻辑，是 PR 的核心抽象。
- python/sglang/srt/models/zaya.py（模块 模型定义；类别 source；类型 data-contract；符号 cca_extend, cca_decode, cca_conv1d_fn, cca_conv1d_update）：ZAYA1 CCA 模型重构主力文件，移除池直接访问，改为调用后端元数据，卷积核提取为独立函数。
- test/registered/unit/models/test_zaya_cca.py（模块 单元测试；类别 test；类型 test-coverage；符号 _MockShortConvBackend, init, _resolve_indices, _resolve_slot_ids）：CPU 单元测试新增 _MockShortConvBackend 模拟后端行为，验证索引解析跨层缓存。
- python/sglang/srt/layers/attention/hybrid_linear_attn_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 ShortConvHybridAttnBackend, init, conv_state_metadata）：新增 ShortConvHybridAttnBackend 包装类，将短卷积后端作为线性侧车注册。
- python/sglang/srt/models/lfm2.py（模块 模型定义；类别 source；类型 data-contract）：LFM2 短卷积层重构，移除池直接访问，改用后端元数据。
- python/sglang/srt/models/lfm2_moe.py（模块 模型定义；类别 source；类型 data-contract）：LFM2-MoE 短卷积层重构，逻辑与 lfm2.py 一致。
- python/sglang/srt/layers/attention/attention_registry.py（模块 注册表；类别 source；类型 dependency-wiring）：注册新后端路由，修复 LFM2 配置导入路径错误。
- python/sglang/srt/layers/attention/mamba/causal_conv1d.py（模块 卷积核；类别 source；类型 core-logic）：支持 int64 索引窄化到 int32 的边界转换。

关键符号：ShortConvMetadata, ShortConvAttnBackend, ShortConvHybridAttnBackend, conv_state_metadata, cca_extend, cca_decode, _refresh_cache_indices, attn_backend_wrapper

关键源码片段

python/sglang/srt/layers/attention/linear/short_conv_backend.py

新增核心后端，封装所有短卷积状态管理逻辑，是 PR 的核心抽象。

```
from __future__ import annotations
from typing import TYPE_CHECKING, Any, List, NamedTuple, Optional
import torch
from sglang.srt.layers.attention.hybrid_linear_attn_backend import MambaAttnBackendBase
from sglang.srt.model_executor.forward_batch_info import ForwardBatch

if TYPE_CHECKING:
    from sglang.srt.model_executor.model_runner import ModelRunner

class ShortConvMetadata(NamedTuple):
    """Per-(layer, step) conv - state handle handed to a model's conv kernel.
```

```
    ``layer_cache`` exposes the per-layer pool views ( ``conv[0]`` = conv state,
    ``conv[1]`` = an optional second state such as ZAYA1's ``prev_hs``,
    ``temporal`` = SSM state, unused by pure short convs). The device tensors are
    cuda-graph-static on the decode/replay path; the ``*_cpu`` host mirrors are
    built once per step only for models whose extend path runs a host loop
    (e.g. ZAYA1 v1) and are ``None`` on decode.
    """

    layer_cache: Any
    cache_indices: torch.Tensor # int64 canonical index tensor
    # cu-seqlens for the varlen prefill conv (device, int32). None on decode.
    query_start_loc: Optional[torch.Tensor] = None
    # Per-request "resumes a cached prefix" mask (device bool). None on decode.
    has_initial_state: Optional[torch.Tensor] = None
    # Host mirror of cache_indices for extend host loops. None on decode.
    slot_ids_cpu: Optional[List[int]] = None
    # Host mirror of has_initial_state for extend host loops. None on decode.
    has_prefix_cpu: Optional[List[bool]] = None


class ShortConvAttnBackend(MambaAttnBackendBase):
    """Owns the short-conv per-request state plumbing (see module docstring)."""

    needs_cpu_seq_lens: bool = False # extend path reads host seq-lens from batch

    def __init__(self, model_runner: ModelRunner):
        super().__init__(model_runner)
        mamba_cache = self.req_to_token_pool.mamba_pool.mamba_cache
        # conv[0] == conv_state: [n_layers, n_slots, conv_dim, conv_kernel - 1]
        self.conv_states_shape = mamba_cache.conv[0].shape

        # Per-step state, resolved ONCE per step in init_forward_metadata /
        # init_forward_metadata_out_graph (never per conv layer).
        self._has_initial_state: Optional[torch.Tensor] = None
        self._slot_ids_cpu: Optional[List[int]] = None
        self._has_prefix_cpu: Optional[List[bool]] = None
        self._cache_indices: Optional[torch.Tensor] = None
        self._cache_indices_buf: Optional[torch.Tensor] = None

    def _reset_step_state(self):
        self._has_initial_state = None
        self._slot_ids_cpu = None
        # ... 其他 reset
```

python/sglang/srt/models/zaya.py

ZAYA1 CCA 模型重构主力文件，移除池直接访问，改为调用后端元数据，卷积核提取为独立函数。

```
# zaya.py（重构后的 CCA 前向 extend 路径）def forward_extend(self, hidden_states, forward_batch):    # 通过后端获取状态元数据（每步仅解析一次）      meta = get_attn_backend().conv_state_metadata(self.layer_idx, forward_batch)    conv_state = meta.layer_cache.conv[0]    prev_hs_state = meta.layer_cache.conv[1]    # 卷积核接收 int64 cache_indices（后端已预先解析）      qk, v2_input = cca_extend(          hidden_states, self.q_proj, self.k_proj, self.v_proj2,          conv_state, prev_hs_state,          meta.cache_indices, # int64 索引          meta.query_start_loc,          meta.has_initial_state,          self.cca_time0, self.cca_time1,      )    return qk, v2_input
```

（注：实际中 *cca_extend* 和 *cca_decode* 函数体包含详细的两阶段卷积实现，此处展示核心调用模式。）

test/registered/unit/models/test_zaya_cca.py

CPU 单元测试新增 _MockShortConvBackend 模拟后端行为，验证索引解析跨层缓存。

```
class _MockShortConvBackend:
    """Stand - in for ``ShortConvHybridAttnBackend`` in CPU unit tests."""

    def __init__(self, pool: "_MockReqToTokenPool"):
        self.req_to_token_pool = pool
        self.token_to_kv_pool = None
        # 每步缓存：id(forward_batch) -> 设备索引 / 主机列表
        self._step_indices = {}
        self._step_slot_ids = {}

    def _resolve_indices(self, forward_batch):
        key = id(forward_batch)
        indices = self._step_indices.get(key)
        if indices is None:
            indices = self.req_to_token_pool.get_mamba_indices(
                forward_batch.req_pool_indices
            ).to(torch.long) # int64
            self._step_indices[key] = indices
        return indices

    def conv_state_metadata(self, layer_id, forward_batch):
        from sglang.srt.layers.attention.linear.short_conv_backend import ShortConvMetadata
        layer_cache = self.req_to_token_pool.mamba2_layer_cache(layer_id)
        indices = self._resolve_indices(forward_batch)
        if forward_batch.forward_mode.is_decode_or_idle():
            return ShortConvMetadata(layer_cache=layer_cache, cache_indices=indices)
        # extend 路径：补充主机端槽位和前缀标志
        slot_ids = self._resolve_slot_ids(forward_batch, indices)
        has_prefix = [int(p) > 0 for p in forward_batch.extend_prefix_lens_cpu]
        return ShortConvMetadata(
            layer_cache=layer_cache,
            cache_indices=indices,
            slot_ids_cpu=slot_ids,
            has_prefix_cpu=has_prefix,
```

)

评论区精华

Codex 自动审查提出三个关键问题：

- CPU 图捕获索引未初始化（P2）：init_forward_metadata_capture_cpu_graph 未填充 _cache_indices，导致解码回放时 None。已在最后一个提交中修复。
- NPU 后端缺少 conv_state_metadata（P1）：NPU 的 AscendMamba2AttnBackend 未实现该方法，运行短卷积模型会引发 AttributeError。已在注册表中增加快速失败（fail-fast）。
- CUDA graph 填充行索引越界（P2）：ZAYA1 的 cca_decode 使用 index_select/index_copy_，不识别 PAD_SLOT_ID（-1），可能导致越界访问。已在重构基线中遗留，作者标记为已知问题，计划单独跟踪。
  - CPU 图捕获时索引未初始化 (correctness): 已在提交 081c76 中修复：覆盖 init_forward_metadata_capture_cpu_graph，填充 _cache_indices。
  - NPU 后端缺少 conv_state_metadata (design): 在注册表中增加快速失败：NPU 上遇到短卷积模型时主动报错，避免运行时崩溃。
  - CUDA graph 填充行索引越界 (correctness): 作者确认该问题在重构前就已存在，且 LFM2 不受影响，决定单独跟踪。

风险与影响

- 风险：
  1. GPU e2e 回归风险：重构了 ZAYA1 和 LFM2 的整个状态管理路径，尽管作者声明 LFM2 输出与基线字节一致，但 ZAYA1 仅 bs=1 验证，bs>1 存在已知 segfault（与基线行为一致）。
  2. NPU 兼容性风险：虽然添加了快速失败，但 NPU 上短卷积模型暂不可用，可能影响 AMD 等非 NVIDIA 平台。
  3. CUDA graph 稳定性：ZAYA1 bs>1 的 CUDA graph 回放有填充行问题，可能导致非法内存访问。
  4. 索引类型变更：从 int32 切换为 int64 主类型，需要确保所有下游操作正确窄化，否则 causal_conv1d 可能崩溃。 - 影响：用户：使用 ZAYA1、LFM2 及 LFM2-MoE 的用户性能不变，但状态管理抽象化，未来支持新短卷积模型更简单。NPU 用户无法使用这些模型（需等待后端实现）。系统：get_attn_backend().conv_state_metadata() 成为短卷积模型的通用入口；注意力注册表新增路由分支，不影响其他混合注意力模型。团队：贡献者接入新短卷积模型只需实现卷积核并调用 conv_state_metadata，无需了解 MambaPool 内部。
- 风险标记：NPU 暂不支持，CUDA graph 填充未处理，cpu 图路径曾有未初始化，ZAYA1 bs>1 图回放基线退化

关联脉络

- PR #29678 feat(mem_cache): unified memory pool for hybrid Mamba / SWA models: 更改了 MambaPool 的接口和布局，与 ShortConvAttnBackend 的状态管理高度相关。