PR #32971 完整报告

sgl-project/sglang

[unified-memory] Support MLA-hybrid-Mamba (Kimi-Linear) on the Triton backend

合并时间：2026-07-31 13:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/32971

执行摘要

- 一句话：统一内存池支持 MLA 混合 Mamba（Kimi-Linear）Triton 后端
- 推荐动作：值得精读。这个 PR 的索引抽象设计很有学习价值：把 page-major 信封直接当作 dense paged pool、用 torch.as_strided 构造重叠视图、以及 compaction 只管 virtual→physical 表从而让上层引用永不失效的思路，比引入新的物理布局或搬移数据优雅得多。阅读时建议重点对照三份文件：build_dense_mla_views（视图与寻址公式）、translate_kv_loc_dense（dense 空间约定）、HybridLinearKVPool 的 full_loc 路由（写路径接线）。合并前请确认两件事：P1 的 FNUZ dtype 归一化是否已在后续 PR 修复，以及 rebase 后的 CI 状态。

功能与动机

PR body 明确指出静态双池的痛点：一个池在空转、另一个池在收缩或 OOM（原文：one side idling while the other retracts or OOMs）。MHA-hybrid 的 unified memory 已合入上游，本 PR 补齐 MLA-hybrid 分支：此前 kv_cache_configurator.py 存在 assert not use_mla_backend，导致 Kimi-Linear 这类 MLA 全注意力 + KDA 线性注意力的模型无法启用 unified memory。作者选择 dense views 路线而非新建物理布局，理由是 MLA 各层行宽统一（kv_lora_rank + qk_rope_head_dim = 576），page-major 信封本身已是合法 dense paged pool，只需索引重编号即可复用上层引用与现有 MLA kernel。

实现拆解

1. 定义 MLA 子池规格与 dense 视图构建：python/sglang/srt/mem_cache/layout/page_major.py 新增 mla_entry_bytes 与 build_dense_mla_views，利用 MLA 各层统一行宽（kv_cache_dim = kv_lora_rank + qk_rope_head_dim，Kimi-Linear 为 576）把 page-major 信封整体视为合法 dense paged pool，用 torch.as_strided 构造逐层重叠但连续的视图，并对最后一层视图越界做硬校验。unified_memory_pool.py 新增 MLASubPoolSpec（V 是 latent 行前缀，无独立 V 区），UnifiedKVPool 增加 view_tail_pad_bytes 参数只扩展分配、不动 slot/watermark 数学，并修复 page_size > 1 时 sink 下限边界。
2. 扩展双端分配器：multi_ended_allocator.py 新增 kernel_page_multiplier 构造参数与 translate_kv_loc_dense 方法：dense id 为 (t // ps) * (ps * multiplier) + t % ps，multiplier 对 MLA 子池取层数；kernel_page_multiplier == 1 时回退到原 translate_kv_loc。文档化约定：compaction 与 in-flight 写集合必须继续使用物理语义的

translate_kv_loc，dense id 只面向 kernel，从而 compaction 重写 virtual→physical 表时上层引用永不失效。

3. 写路径路由与接线：memory_pool.py 中 HybridLinearKVPool 的 MLA 分支优先透传 KVWriteLoc.full_loc（unified 场景下即 dense loc），无则回退 loc；set_mla_kv_buffer / get_mla_kv_buffer 走 _full_translate 且只翻译一次。新增 UnifiedMLATokenToKVPool 完成整页信封粒度的 move_kv_cache。kv_cache_configurator.py 移除 assert not use_mla_backend 并转发 kv_lora_rank / qk_rope_head_dim；triton_backend.py 同步调整。Gating 保持保守：unified ⇒ page-major ⇒ full-attention 仅 Triton，linear/Mamba 侧仅 Triton，spec decode / PD disagg / hicache / DCP 仍关闭。
4. 修复上游 FLA 内核寻址 bug：python/sglang/kernels/ops/attention/fla/chunk_delta_h.py 的 chunk_gated_delta_rule_fwd_h 原先硬编码 linear-attention state 的 slot pitch 为 H・V・K，对 envelope-strided 池会错位寻址，改为 initial_state.stride(0) 并采用 int64 索引。该修复影响所有 KDA × page-major 配置，连续池行为不变。
5. 测试与端到端验证：新增 test_unified_mla_views.py（CPU 纯 torch：dense 寻址、tail pad、整页 move、跨 compaction 的 dense translate）、test_unified_mla_gpu_parity.py（真实 K3 几何 L=24/D=576，覆盖 Triton fallback 与 TMA JIT 两条 set_mla_kv_buffer 路径以及 page_size 1/64）、test_full_loc_fast_path.py 的 MLA 侧 full_loc 路由测试。B300 单机 8×B300 SXM6、TP1 上，Kimi-Linear-48B-A3B-Instruct（27 层 = 20 KDA + 7 MLA）GSM8K 200 题，unified 与 static 双池同为 0.910，精度完全持平。

关键文件：

- python/sglang/srt/mem_cache/unified_memory_pool.py（模块 统一内存池；类别 source；类型 core-logic；符号 MLASubPoolSpec, kv_cache_dim, entry_bytes, view_tail_pad_bytes）：核心实现文件：新增 MLASubPoolSpec、UnifiedMLATokenToKVPool、view_tail_pad_bytes 与 sink 下限修复，移除对 MLA 的禁用断言，是整套 unified MLA 支持的枢纽。
- python/sglang/srt/mem_cache/multi_ended_allocator.py（模块 双端分配器；类别 source；类型 core-logic；符号 translate_kv_loc_dense, kernel_page_multiplier）：新增 kernel_page_multiplier 与 translate_kv_loc_dense，是 dense 索引空间的核心定义处；明确划分物理语义（compaction 用）与 dense 语义（kernel 用）。
- python/sglang/srt/mem_cache/layout/page_major.py（模块 视图布局；类别 source；类型 core-logic；符号 build_dense_mla_views, mla_entry_bytes）：新增 build_dense_mla_views 与 mla_entry_bytes，是整个 dense 视图方案的核心数学：重叠视图构造、统一 block table 与尾填充校验。
- test/registered/unit/mem_cache/test_unified_mla_views.py（模块 视图测试；类别 test；类型 test-coverage；符号 TestMLASubPoolSpec, TestDenseMLAViews, _build_mla_views）：CPU 纯 torch 单元测试，覆盖 MLASubPoolSpec 字节数学、dense 寻址公式、缺尾填充报错、跨 compaction 的 dense 翻译等核心不变量。
- test/registered/unit/mem_cache/test_unified_mla_gpu_parity.py（模块 GPU 对比；类别 test；类型 test-coverage；符号 TestUnifiedMLAPoolGPUParity, set_mla_kv_buffer, _dense）：GPU byte-parity 测试，在真实 K3 几何（L=24、D=512+64）上对比统一池与

stock MLATokenToKVPool，覆盖 Triton fallback 与 TMA JIT 两条 set_mla_kv_buffer 路径及 page_size 1/64。

- test/registered/unit/mem_cache/test_full_loc_fast_path.py（模块 路由测试；类别 test；类型 test-coverage；符号 TestHybridLinearMLARouting, _RecordingMLAPool, test_mla_writes_full_loc_from_write_loc）：补充 MLA 侧 full_loc 快速路径路由测试：set_kv_buffer 透传 full_loc、缺省回退 loc、set/get_mla_kv_buffer 只翻译一次。
- python/sglang/srt/mem_cache/memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 HybridLinearKVPool, KVWriteLoc, full_loc, _full_translate）：HybridLinearKVPool 的 MLA 写路径路由：KVWriteLoc.full_loc 透传与 _full_translate hook，是 dense loc 进入 kernel 的最后一跳。
- python/sglang/srt/layers/attention/triton_backend.py（模块 注意力后端；类别 source；类型 core-logic）：Triton attention 后端适配 unified MLA 池的调用约定，是本次功能实际生效的前向路径。
- python/sglang/srt/mem_cache/kv_cache_configurator.py（模块 KV 缓存配置；类别 source；类型 configuration）：移除 assert not use_mla_backend 并转发 kv_lora_rank / qk_rope_head_dim，是解锁 MLA-hybrid-Mamba 统一内存的开关。
- python/sglang/kernels/ops/attention/fla/chunk_delta_h.py（模块 FLA 内核；类别 infra；类型 infrastructure；符号 chunk_gated_delta_rule_fwd_h）：上游 FLA 内核寻址 bug 修复：硬编码 pitch 改为 initial_state.stride(0) + int64 索引，影响所有 KDA × page-major 配置。

关键符号：build_dense_mla_views, translate_kv_loc_dense, MLASubPoolSpec.entry_bytes, MLASubPoolSpec.kv_cache_dim, UnifiedKVPool.init, UnifiedMLATokenToKVPool.move_kv_cache, KVWriteLoc.full_loc, chunk_gated_delta_rule_fwd_h

关键源码片段

python/sglang/srt/mem_cache/unified_memory_pool.py

核心实现文件：新增 MLASubPoolSpec、UnifiedMLATokenToKVPool、view_tail_pad_bytes 与 sink 下限修复，移除对 MLA 的禁用断言，是整套 unified MLA 支持的枢纽。

```
# python/sglang/srt/mem_cache/unified_memory_pool.py
@dataclass(frozen=True, kw_only=True)
class MLASubPoolSpec(SubPoolSpec):
    """MLA 形态子池的逐 slot 布局。

    每层每个 token 只有一条 latent 行（`kv_lora_rank + qk_rope_head_dim`），
    V 是该行的一个前缀切片，因此没有独立的 V 区域。它不是
    `MHASubPoolSpec` 的子类——MHA 的 K+V 字节运算和 `v_head_dim > 0`
    不变量在这里都不成立。
    """

    kv_lora_rank: int
    qk_rope_head_dim: int
```

```
    store_dtype: torch.dtype

    def __post_init__(self):
        super().__post_init__()
        assert self.kv_lora_rank > 0, f'kv_lora_rank 必须为正；当前为 {self.kv_lora_rank}'
        assert self.qk_rope_head_dim > 0, f'qk_rope_head_dim 必须为正；当前为 {self.qk_rope_
        head_dim}'

    @property
    def kv_cache_dim(self) -> int:
        # Kimi-Linear 为 512 + 64 = 576
        return self.kv_lora_rank + self.qk_rope_head_dim

    def entry_bytes(self) -> int:
        return self.layer_num * self.kv_cache_dim * self.store_dtype.itemsize

    def get_dtype(self) -> torch.dtype:
        return self.store_dtype
```

python/sglang/srt/mem_cache/multi_ended_allocator.py

新增 kernel_page_multiplier 与 translate_kv_loc_dense，是 dense 索引空间的核心定义处；明确划分物理语义（compaction 用）与 dense 语义（kernel 用）。

```
# python/sglang/srt/mem_cache/multi_ended_allocator.py
def translate_kv_loc_dense(
    self,
    virt_tokens: torch.Tensor,
    *,
    out: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    """把 virtual token id 翻译为 DENSE（kernel 侧）id。

    dense(t) = (t // ps) * (ps * kernel_page_multiplier) + t % ps，即
    `translate_kv_loc` 的页步长按 `kernel_page_multiplier`（对 dense-view 的
    MLA 子池等于层数 `layer_num`，见 `build_dense_mla_views`）放大后的版本。

    关键约定：compaction、in-flight 写集合等内部机制必须继续使用
    `translate_kv_loc`，dense id 只允许交给 kernel——这样 compaction 重写
    virtual->physical 表时，上层持有的引用永远不失效。

    tombstone 钳制把 -1 条目映射到 dense id 0（每层视图都在 page-0 保留
    sink 内）。支持 `out=` 以维持 cuda-graph 缓冲区稳定性。
    """
    if self.kernel_page_multiplier == 1:
        # 非 dense 子池（Mamba / MHA）：退回原有物理语义
        return self.translate_kv_loc(virt_tokens, out=out)

    if out is not None:
        assert out.dtype == torch.int64, (
```

```
            f'translate_kv_loc_dense: out= 必须是 int64（与 v2p 一致），'
            f'当前为 {out.dtype}'
        )
        assert out.shape == virt_tokens.shape, (
            f'translate_kv_loc_dense: out= 形状 {tuple(out.shape)} 必须与 '
            f'virt_tokens 形状 {tuple(virt_tokens.shape)} 一致'
        )

    with record_function('MultiEndedAlloc.translate_kv_loc_dense'):
        dense_page_stride = self.page_size * self.kernel_page_multiplier
        if self.page_size == 1:
            # dense = phys * multiplier；tombstone -1 缩放为负值后钳到 0
            if out is not None:
                # canonical 调用方做 in-place `translate(ids, out=ids)`，
                # index_select(out=) 禁止索引 / 输出别名，故走临时张量 + copy_
                tmp = torch.index_select(self.virtual_to_physical, 0, virt_tokens)
                tmp = torch.clamp_min(tmp * dense_page_stride, 0)
                out.copy_(tmp)
                return out
            result = torch.index_select(self.virtual_to_physical, 0, virt_tokens)
            return torch.clamp_min(result * dense_page_stride, 0)

        # page_size > 1：先取页号与页内偏移，再按 dense 页步长缩放
        virt_pages = virt_tokens // self.page_size
        offsets = virt_tokens % self.page_size
        if out is not None:
            torch.index_select(self.virtual_to_physical, 0, virt_pages, out=out)
            out.mul_(dense_page_stride)
            out.add_(offsets)
            out.clamp_(min=0) # tombstoned 页：-1 * ps + offset 落在 [-ps, -1]
            return out
        phys_pages = self.virtual_to_physical[virt_pages]
        result = phys_pages * dense_page_stride + offsets
        return torch.clamp_min(result, 0)
```

python/sglang/srt/mem_cache/layout/page_major.py

新增 build_dense_mla_views 与 mla_entry_bytes，是整个 dense 视图方案的核心数学：重叠视图构造、统一 block table 与尾填充校验。

```
# python/sglang/srt/mem_cache/layout/page_major.py
def mla_entry_bytes(*, layer_num: int, kv_cache_dim: int, itemsize: int) -> int:
    """一个 MLA slot 跨所有层占用的字节数（单条 latent 行，无独立 V 区）。"""
    return layer_num * kv_cache_dim * itemsize


def build_dense_mla_views(
    raw: torch.Tensor,
    *,
    layer_num: int,
```

```
    kv_cache_dim: int,
    store_dtype: torch.dtype,
    page_size: int,
    num_pages: int,
    anchor_bytes: int = 0,
) -> List[torch.Tensor]:
    """在 page-major layout 的 `raw` 上构造逐层 DENSE 视图。

    page 信封布局为 `[L0_latent * ps | L1_latent * ps | ...]`。因为 MLA 各层
    共享统一的行宽 `kv_cache_dim`，整个信封本身就是一个合法的 dense paged
    pool，只需重编号索引：把逐层偏移 `l * ps * kv_cache_dim` 折进每个视图的
    `storage_offset`，第 l 层视图就是普通的连续 `(num_pages * layer_num * ps,
    1, kv_cache_dim)` 张量，由层无关的 dense id 寻址：

        dense(t) = (t // ps) * (ps * layer_num) + t % ps   # t 为物理 token id

    这样所有层共享一张 block table（页号 = page * layer_num），要求
    `.view(-1, page_size, kv_cache_dim)` 的 kernel（trtllm / cutlass /
    flashmla）可以直接作用于这些视图。

    视图之间互相重叠（视图 l+1 是视图 l 平移 ps 行）：安全的前提是第 l 层
    永远只按 dense id 索引，而 dense id 相对视图 l 的起点总是落在第 l 层的
    字节区间内。最后一层视图会超出最后一个 page 信封 `(layer_num - 1) * ps`
    行，因此 `raw` 必须多分配一页信封的尾部填充
    （`UnifiedKVPool.view_tail_pad_bytes`）。
    """
    itemsize = store_dtype.itemsize
    row_bytes = kv_cache_dim * itemsize
    page_bytes = page_size * layer_num * row_bytes
    n_dense = num_pages * layer_num * page_size
    assert anchor_bytes % itemsize == 0

    # 越界硬校验：最后一层视图的末尾必须在 raw 容量内，否则报错提示分配 tail pad
    last_view_end = (
        anchor_bytes + (layer_num - 1) * page_size * row_bytes + n_dense * row_bytes
    )
    assert last_view_end <= raw.numel() * raw.itemsize, (
        f'build_dense_mla_views: layer {layer_num - 1} 的视图结束于字节 '
        f'{last_view_end}，但 raw 只有 {raw.numel() * raw.itemsize} 字节；'
        f'请通过 view_tail_pad_bytes 预留一页信封（{page_bytes} B）'
    )

    as_dtype_view = raw.view(store_dtype)
    views = []
    for layer in range(layer_num):
        base_bytes = anchor_bytes + layer * page_size * row_bytes
        assert base_bytes % itemsize == 0
        # torch.as_strided 制造重叠且连续的逐层视图，不改动底层存储
        views.append(
```

```
            torch.as_strided(
                as_dtype_view,
                size=(n_dense, 1, kv_cache_dim),
                stride=(kv_cache_dim, kv_cache_dim, 1),
                storage_offset=base_bytes // itemsize,
            )
        )
    return views
```

评论区精华

唯一的 review 评论来自 chatgpt-codex-connector[bot]，标记为 P1：

> 在 gfx94/MI300 上使用 --kv-cache-dtype fp8_e4m3 时，dtype 解析会落到 torch.float8_e4m3fnuz，但 _store_dtype_for 没有把该 dtype 归一化为 uint8，因此新的 MLA spec 会构建 native-FNUZ 视图；而 MLATokenToKVPool 会把自己的 store_dtype 归一化为 uint8。继承的写路径会把输入按 uint8 重新解释再做数值转换进 FNUZ 目标，静默损坏缓存的 KV 值。建议把 torch.float8_e4m3fnuz 纳入归一化。

该评论出现在 unified_memory_pool.py 第 1060 行（assert not use_mla_backend 移除处附近），没有任何作者回复或修复记录，合并时疑似未解决。这是本 PR 最需要后续跟踪的风险点。

- FNUZ dtype 未归一化为 uint8 导致 KV 静默损坏（P1） (correctness): 无作者回复与修复记录，合并时该 P1 疑似未解决，建议后续 PR 跟踪修复。

风险与影响

- 风险：
  1. FNUZ dtype 数据损坏（P1，未解决）：unified_memory_pool.py 的 _store_dtype_for 未将 torch.float8_e4m3fnuz 归一化为 uint8，与 MLATokenToKVPool 的归一化行为不一致，在 AMD gfx94/MI300 上启用 --kv-cache-dtype fp8_e4m3 且使用 unified MLA 池时会静默损坏 KV 缓存。这是合并时遗留的开放问题。
  2. 核心内存路径变更：unified_memory_pool.py / multi_ended_allocator.py / memory_pool.py 是 KV 缓存核心，改动会影响所有启用 unified memory 的部署；其中 sink 下限修复和 view_tail_pad_bytes 对 page_size > 1 的 MHA 路径同样生效，需要回归验证。
  3. 上游内核修复影响面：chunk_delta_h.py 对 chunk_gated_delta_rule_fwd_h 的 stride 修改影响所有 KDA × page-major 配置，作者声明连续池行为不变，但缺少专门的回归测试佐证。
  4. 合并状态风险：PR body 自述基于 9f5655340，合入时 main 已前进约 24 个 commit，需要 rebase；CI 的 PR Test 与 PR Test (Extra) 双跑均为 ❓ 失败状态。
  5. 平台覆盖不足：新增 GPU byte-parity 测试只在 B300（CUDA 13.0）验证，未覆盖 AMD 平台，恰是 P1 问题暴露的环境。 - 影响：用户侧：Kimi-Linear 等 MLA-hybrid-Mamba 模型现在可以配合 --enable-unified-memory + Triton 后端部署，

缓解静态双池的空间空转 / 单侧 OOM 问题，GSM8K 精度与静态基线完全持平（0.910）。系统侧：这是统一内存池在 MLA 方向的第一个落地，gating 仍保守（full-attention仅 Triton，linear/Mamba 仅 Triton，spec decode / PD disagg / hicache / DCP 关闭），不会影响未开启 unified memory 的默认路径。团队侧：virtual / physical / dense 三坐标系 + 逐层重叠 dense 视图 + kernel_page_multiplier 的抽象为后续开放 pagedMLA backends（trtllm_mla、flashinfer prefill、cutedsl_mla、tokenspeed_mla）铺平了道路，是 unified-memory 功能线的重要里程碑。 - 风险标记：FNUZ dtype 未归一化（P1 未解决），核心 KV 缓存路径变更，需 rebase（基于旧 main），CI 双跑失败，上游内核修复缺专项回归

关联脉络

- 暂无明显关联 PR