

PR #52789 完整报告

vllm-project/vllm

[Perf] Support internal prefill checkpoints for Mamba prefix caching, 9%~25% TTFT improvement

合并时间: 2026-08-23 01:32

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/52789>

执行摘要

- 一句话: Mamba 前缀缓存支持内部 checkpoint, TTFT 提升 9%~25%
- 推荐动作: 值得精读。重点看三处设计: 一是调度切分 (`_mamba_block_aligned_split`) 与缓存分配 (`_needs_internal_checkpoint`) 如何通过 MambaSpec 新字段建立跨层契约, 避免硬编码模型名; 二是 metadata 层用“block 边界 + FLASHKDA_CHUNK_SIZE 对齐”双条件做安全降级, 保证非对齐场景不产生错误 checkpoint; 三是 Triton 落盘 kernel 一次覆盖 conv 状态与 recurrent 状态的写法。建议与 #53196、#53212、#52971 以及 FlashKDA 上游 PR 连读, 理解一个性能特性从内核到调度器的完整演进链。

功能与动机

PR body 明确指出: Mamba prefix caching 目前会在最后一个 block 边界拆分 prefill, 导致同一请求被拆成两段模型 forward——以 8k 输入为例, 原来是 7680 token 的 FlashKDA 计算保存 checkpoint 后, 还要再走一次 320 token 的模型 forward 与 FlashKDA 调用。作者的目标是“avoid a second full-model pass through attention, MoE, routing, and TP collectives”, 即通过内部 checkpoint 把主段计算合并到一次前向中, 实测 8k 输入 TTFT 提升 9.4% (并发 1) / 25.5% (并发 4) / 20.7% (并发 16)。

实现拆解

按数据通路自底向上拆解为 6 步:

1. 能力声明与跨层契约 (`vllm/v1/kv_cache_interface.py`、`vllm/models/kimi_k3/nvidia/kda.py`、`vllm/v1/core/sched/scheduler.py`): MambaSpec 新增 `num_prefill_checkpoint_blocks` 字段, 并在 `max_memory_usage_bytes` 与 `is_uniform_with_collection` 中同步扩维, 保证容量预算和组内一致性判断覆盖新块类型; KDA 层重写 `get_kv_cache_spec()`, 仅当 `kda_prefill_backend == "flashkda"` 时通过 `dataclasses.replace` 把该字段置 1; Scheduler 汇总所有 kv cache group 得到 `mamba_has_prefill_checkpoint_blocks` (要求非 eagle 模式), 作为后续切分的总开关。
2. 调度切分放宽 (`scheduler.py` 的 `_mamba_block_aligned_split`): 当内部 checkpoint 可用且本次 chunk 起点块对齐 (`start % block_size == 0`) 时, 把 `last_cache_position` 置 0 并跳过强制对齐分支, 允许一整段 8000 token 只触发一次模型 forward, FlashKDA 内部完成“7680 主段 + 320 尾段”的计算与状态导出。

3. 缓存块分配 (vllm/v1/core/single_type_kv_cache_manager.py) : `MambaManager` 新增 `_needs_internal_checkpoint()` 判断与 `_num_checkpoint_blocks` 字典;
`get_num_blocks_to_allocate` 在估算块数时叠加 `checkpoint_block`,
`allocate_new_blocks` 用 `num_skipped_blocks - checkpoint_block` 调整 null 块填充区间,
把 `checkpoint` 块夹在最后一个计算块与 `running` 块之间, 不破坏“最后一块保存运行态”
的不变量; `pop_blocks_for_free` 同步清理状态。
4. metadata 契约 (vllm/models/kimi_k3/nvidia/kda_metadata.py) : 新增
`KDACheckpointMetadata(checkpoint_offsets, state_indices)` 并挂到
`KimiK3KDAMetadata.checkpoint; build()` 在 `mamba_cache_mode == "align"` 时逐请求
计算导出偏移: 仅当 `seq_len` 未对齐、当前 query 真实跨过最后一个 `block` 边界、且偏移
对 `FLASHKDA_CHUNK_SIZE = 16` 对齐时才导出, 否则静默降级为不导出。
5. 模型侧单次导出 (kda.py、csrc/flashkda_registration.cpp) : `_flashkda_prefill` 新增可
选参数 `checkpoint_state / checkpoint_offsets`, `STABLE_TORCH_LIBRARY` 注册同步扩
展; `_prefill_conv` 在 `checkpoint` 存在时以全量 query 调用一次 `FlashKDA`, 随后用新增
Triton kernel `_store_cache_checkpoints_kernel` 把 `conv` 输入尾窗与 `recurrent`
`checkpoint` 一次性写入目标缓存块, 替代逐序列 Python 循环。
6. 测试与构建配套: `test_internal_checkpoint_split` (chunk 切分)、
`test_partial_hit_then_internal_checkpoint_uses_distinct_mamba_blocks` (部分命中与
`checkpoint` 块共存)、`test_internal_checkpoint_requires_block_aligned_start` 与
`test_internal_checkpoint_metadata_skips_unaligned_offset` (非对齐起点退化)、
`test_flashkda_correctness` (带 `checkpoint` 的 `FlashKDA` 数值一致性) 共 5 个测试文件
覆盖; `cmake/external_projects/flashkda.cmake` 同步 `FlashKDA` 子模块引用。

关键文件:

- `vllm/models/kimi_k3/nvidia/kda.py` (模块 `KDA` 内核; 类别 `source`; 类型 `data-contract`
; 符号 `_store_cache_checkpoints_kernel`, `_flashkda_prefill`, `get_kv_cache_spec`) : 核
心实现文件: 新增 `_store_cache_checkpoints_kernel` Triton 落盘 kernel,
`_flashkda_prefill` 扩展 `checkpoint` 参数, `_prefill_conv` 支持单次前向导出,
`get_kv_cache_spec` 向调度层声明能力; 同时 `_flashkda_buffer_specs` 增加一块
`checkpoint` 缓冲区。
- `vllm/models/kimi_k3/nvidia/kda_metadata.py` (模块 `KDA` 元数据; 类别 `source`; 类型
`data-contract`; 符号 `KDACheckpointMetadata`, `build`) : 新增 `KDACheckpointMetadata`
数据契约并挂到 `KimiK3KDAMetadata.checkpoint; build()` 逐请求计算 `checkpoint` 偏移
与目标缓存块下标, 用 `block` 边界 + `FLASHKDA_CHUNK_SIZE` 对齐双条件保证非对齐场
景安全降级。
- `vllm/v1/core/single_type_kv_cache_manager.py` (模块 缓存管理器; 类别 `source`; 类型
`core-logic`; 符号 `_needs_internal_checkpoint`, `get_num_blocks_to_allocate`,
`allocate_new_blocks`) : `MambaManager` 新增 `_needs_internal_checkpoint` 判断与
`_num_checkpoint_blocks` 记账, `get_num_blocks_to_allocate / allocate_new_blocks` 的
块数估算与 null 块填充逻辑都围绕 `checkpoint` 块扩展, 是调度分配正确性的关键。
- `tests/v1/core/prefix_cache/test_partial_prefix_cache_hits.py` (模块 前缀缓存; 类别 `test`
; 类型 `test-coverage`; 符号 `test_partial_hit_then_internal_checkpoint_uses_distinct_m`

amba_blocks, test_internal_checkpoint_requires_block_aligned_start)：新增两个关键不变量测试：部分前缀命中与内部 checkpoint 必须使用不同 Mamba 块，以及非对齐起点不得分配 checkpoint 块，防止 CoW 与 checkpoint 块互相踩踏。

- vllm/v1/core/sched/scheduler.py (模块 调度器；类别 source；类型 dependency-wiring；符号 _mamba_block_aligned_split, mamba_has_prefill_checkpoint_blocks)：新增 mamba_has_prefill_checkpoint_blocks 总开关并在 _mamba_block_aligned_split 中放宽最后一个缓存边界的强制停靠，使整段 prefill 单次 forward，是收益的主要来源。
- tests/v1/core/test_mamba_align_chunk_split.py (模块 切分测试；类别 test；类型 test-coverage；符号 _make_hybrid_kv_cache_manager, test_internal_checkpoint_split)：test_internal_checkpoint_split 验证带 checkpoint 时整段 prefill 不再被强制切分，test_unaligned_resume_never_runs_past_its_block 扩展覆盖 checkpoint 模式下的 mid-block resume 不越界。
- tests/models/kimi_k3/test_kda_metadata.py (模块 元数据测试；类别 test；类型 test-coverage；符号 test_internal_checkpoint_metadata_targets_last_aligned_boundary, test_internal_checkpoint_metadata_skips_unaligned_offset)：验证 metadata 层 checkpoint 偏移计算：目标必须是最后一个对齐边界，非对齐 offset (如 47 非 16 倍数) 必须跳过导出，守护安全降级语义。
- tests/models/kimi_k3/test_kda.py (模块 KDA 测试；类别 test；类型 test-coverage；符号 _flashkda_prefill, _store_cache_checkpoints_kernel)：test_flashkda_correctness 扩展验证带 checkpoint_state/checkpoint_offsets 的 FlashKDA 输出与朴素实现数值一致，并直接测试 _store_cache_checkpoints_kernel 的 conv 状态与 recurrent 状态写入。
- vllm/v1/kv_cache_interface.py (模块 缓存接口；类别 source；类型 core-logic；符号 MambaSpec, max_memory_usage_bytes, is_uniform_with_collection)：MambaSpec 新增 num_prefill_checkpoint_blocks 契约字段，并同步 max_memory_usage_bytes 与 is_uniform_with_collection，是跨层信息传递的根基。
- csrc/flashkda_registration.cpp (模块 算子注册；类别 source；类型 core-logic)：STABLE_TORCH_LIBRARY 注册新增可选参数 checkpoint_state 与 checkpoint_offsets，是 FlashKDA 算子 ABI 扩展点。
- tests/v1/core/test_prefix_caching.py (模块 缓存测试；类别 test；类型 test-coverage)：最小配套调整，为新字段补一处构造参数，确保既有前缀缓存测试在新契约下不失配。
- cmake/external_projects/flashkda.cmake (模块 构建配置；类别 other；类型 configuration)：同步 FlashKDA 子模块版本，确保带 checkpoint 支持的原生算子被正确编译链接。

关键符号：_store_cache_checkpoints_kernel, _flashkda_prefill, get_kv_cache_spec, KDACheckpointMetadata, build, _needs_internal_checkpoint, get_num_blocks_to_allocate, allocate_new_blocks, _mamba_block_aligned_split, max_memory_usage_bytes

关键源码片段

[vllm/models/kimi_k3/nvidia/kda_metadata.py](#)

新增 KDACheckpointMetadata 数据契约并挂到 KimiK3KDAMetadata.checkpoint; build() 逐请求计算 checkpoint 偏移与目标缓存块下标, 用 block 边界 + FLASHKDA_CHUNK_SIZE 对齐双条件保证非对齐场景安全降级。

文件: vllm/models/kimi_k3/nvidia/kda_metadata.py (KimiK3KDAMetadataBuilder.build)

说明: 为每个 prefill 请求计算“内部 checkpoint”的落点。目标是让 FlashKDA

在一次调用里导出倒数第二个 block 边界处的状态, 供后续 chunk 复用。

```
checkpoint = None
```

```
if (
```

```
    num_prefills > 0
```

```
    and self.kv_cache_spec.num_prefill_checkpoint_blocks > 0
```

```
    and self.vllm_config.cache_config.mamba_cache_mode == "align"
```

```
):
```

```
    # 逐请求计算: offset 表示 checkpoint 相对本次 query 起点的偏移;
```

```
    # 只有当 seq_len 未 block 对齐、当前 query 真实跨过最后一个 block 边界、
```

```
    # 且偏移对 FLASHKDA_CHUNK_SIZE (16) 对齐时才导出, 否则退化为不导出
```

```
    assert m.seq_lens_cpu_upper_bound is not None
```

```
    request_rows = list(range(m.num_reqs))
```

```
    if spec_sequence_masks_cpu is not None:
```

```
        # spec decode 场景只取非草稿请求行
```

```
        request_rows = active_non_spec_mask_cpu.nonzero().flatten().tolist()
```

```
    all_query_lens = query_start_loc_cpu.diff().tolist()
```

```
    query_lens = [all_query_lens[row] for row in request_rows]
```

```
    seq_lens = m.seq_lens_cpu_upper_bound.tolist()
```

```
    block_size = self.kv_cache_spec.block_size
```

```
    checkpoint_splits = []
```

```
    checkpoint_cols = []
```

```
    for row, query_len in zip(request_rows, query_lens):
```

```
        seq_len = seq_lens[row]
```

```
        # 最后一个 block 边界 (seq_len 向下取整) 相对 query 起点的距离
```

```
        offset = seq_len // block_size * block_size - (seq_len - query_len)
```

```
        valid = (
```

```
            seq_len % block_size != 0
```

```
            and 0 < offset < query_len
```

```
            and offset % FLASHKDA_CHUNK_SIZE == 0
```

```
        )
```

```
        offset = offset if valid else 0
```

```
        first_len = offset or query_len
```

```
        # ( 前段长度, 尾段长度 ); 尾段为 0 表示无需导出 checkpoint
```

```
        checkpoint_splits.append((first_len, query_len - first_len))
```

```
        # 目标缓存块列号: 最后一个 block 的块下标, 无效时置 -1
```

```
        checkpoint_cols.append(seq_len // block_size - 1 if valid else -1)
```

```
    if any(tail for _, tail in checkpoint_splits):
```

```
        checkpoint_offsets_tensor = async_tensor_h2d(
```

```
            [first if tail else 0 for first, tail in checkpoint_splits],
```

```
            dtype=torch.int32,
```

```
            device=query_start_loc.device,
```

```
        )
```

```

request_rows_tensor = async_tensor_h2d(
    request_rows, dtype=torch.int64, device=query_start_loc.device
)
checkpoint_cols_tensor = async_tensor_h2d(
    checkpoint_cols, dtype=torch.int64, device=query_start_loc.device
)
# 从 block table 取目标块下标; 无效行以 NULL_BLOCK_ID 填充
checkpoint_state_indices = m.block_table_tensor[
    request_rows_tensor, checkpoint_cols_tensor
]
checkpoint_state_indices = torch.where(
    checkpoint_cols_tensor >= 0,
    checkpoint_state_indices,
    NULL_BLOCK_ID,
)
checkpoint = KDACheckpointMetadata(
    checkpoint_offsets_tensor,
    checkpoint_state_indices,
)

```

vllm/v1/core/single_type_kv_cache_manager.py

MambaManager 新增 `_needs_internal_checkpoint` 判断与 `_num_checkpoint_blocks` 记账, `get_num_blocks_to_allocate / allocate_new_blocks` 的块数估算与 null 块填充逻辑都围绕 checkpoint 块扩展, 是调度分配正确性的关键。

- # 文件: vllm/v1/core/single_type_kv_cache_manager.py (MambaManager)
- # 说明: 判断本次 allocation 是否需要多预留一个“内部 checkpoint”块,
- # 并把该块插到 running 块之前, 维持“最后一块保存运行态”的不变量。

```

def _needs_internal_checkpoint(
    self,
    request_id: str,
    num_tokens: int,
    num_computed_tokens: int,
) -> bool:
    assert isinstance(self.kv_cache_spec, MambaSpec)
    # checkpoint 块位于倒数第二个块 (cdiv(num_tokens, block_size) - 2)
    checkpoint_idx = cdiv(num_tokens, self.block_size) - 2
    blocks = self.req_to_blocks[request_id]
    return (
        self.kv_cache_spec.num_prefill_checkpoint_blocks > 0
        # 仅当整段 prefill 未在 block 边界结束才需要导出
        and num_tokens % self.block_size != 0
        # 且本次排程从 block 边界开始 (对齐起点约束, 见 #53212)
        and num_computed_tokens % self.block_size == 0
        and checkpoint_idx >= 0
        # 该槽位尚未被真实块占用, 避免覆盖已有状态
        and (checkpoint_idx >= len(blocks) or blocks[checkpoint_idx].is_null)
    )

```

```
# 在 get_num_blocks_to_allocate 中调用并暂存结论，供 allocate_new_blocks 复用：
# checkpoint_block = int(self._needs_internal_checkpoint(...))
# if not apply_admission_cap:
# self._num_checkpoint_blocks[request_id] = checkpoint_block
# allocate_new_blocks 里用 num_skipped_blocks - checkpoint_block 控制 null 块填充，
# 使 checkpoint 块保持真实块身份，running 块顺延一位。
```

评论区精华

核心讨论与结论如下：

- `num_prefill_checkpoint_blocks` 是否必要的质疑：审阅者 ZJY0516 在 `vllm/v1/kv_cache_interface.py` 评论 “looks like no one change this value. And I don't think this variable is needed”。作者回应：该字段在 `kda.get_kv_cache_spec()` 里由 `replace()` 写入，调度器依赖它避免第二次模型 forward，并承认 “Not very elegant now”，计划后续 PR 支持其他后端。
- 大 batch 性能疑问：ZeldaHuang 询问 “Will these multiple flashkda calls cause performance degradation under large batch size(like bs64)?”。作者确认这是明确优化点，希望通过 FlashKDA 上游 PR (FlashKDA#7) 落地后合并为单次原生调用，本 PR 先行合入。
- 冗余判断清理：njhill 建议把 `use_eagle` 并入 `mamba_has_prefill_checkpoint_blocks` 的定义（附带 TODO 注释），避免在 `_mamba_block_aligned_split` 里重复判断，作者已修复。
- spec-decoding 分工协调：作者原本计划在 follow-up 支持 spec decoding；ZeldaHuang 指出其 `vllm#52971` “already supports spec-decoding”，无需重复劳动，作者随即取消该计划，双方约定 #52971 在本 PR 之后 rebase。
- 已知限制：ZJY0516 在 issue 评论提醒 “current implementation may not support internal checkpoint when request has multimodal input”，属于明确的未覆盖场景。
 - `num_prefill_checkpoint_blocks` 字段是否多余 (question)：字段保留；作者承认当前写法不够优雅，后续 PR 会扩展到其他 KDA 后端。
 - 大 batch 下多次 FlashKDA 调用是否拖慢性能 (performance)：本 PR 先行合入，优化由 FlashKDA#7 与 `vllm#52971` 承接。
 - `use_eagle` 判断冗余 (style)：作者已按建议修复。
 - spec-decoding 支持的分工协调 (design)：spec decoding 由 #52971 承担，本 PR 不再实现。
 - 多模态输入不支持内部 checkpoint (other)：作为已知限制接受，未在本 PR 内修复。

风险与影响

- 风险：风险集中在以下四点：
 1. 调度器与模型层的契约耦合：`_mamba_block_aligned_split` 的行为依赖 `MambaSpec.num_prefill_checkpoint_blocks` 与 `mamba_has_prefill_checkpoint_blocks` 的 `all()` 汇总逻辑，任一层的字段不一致都会导致优化静默关闭或块分配偏差；后续新增 Mamba 后端时必须同步维护该契约。

2. 大 batch 性能未验证: ZeldaHuang 提出的 bs64 多段 FlashKDA 调用叠加 Triton 落盘 kernel 的开销尚无基准数据, 作者将其挂起到 FlashKDA 上游与 #52971, 存在长尾场景回归可能。
3. 覆盖边界明确收窄: 仅 FlashKDA 后端生效; Triton 后端、eagle/spec decode、多模态输入均不走内部 checkpoint 路径。调度器在 `mamba_has_prefill_checkpoint_blocks` 中显式排除 `use_eagle`, 且对齐约束 (起点块对齐、偏移为 16 的倍数) 不满足时静默降级, 行为差异需要测试守护。
4. 内存预留敏感性: `MambaSpec.max_memory_usage_bytes` 已把 checkpoint 块计入容量预算, 但若其他 kv cache manager 分支或未来新缓存模式未同步该字段, 可能出现块预算低估导致的调度等待。- 影响: 影响评估如下:
 - 用户 / 服务: 开启 prefix caching 的 Kimi-K3 + FlashKDA 场景 TTFT 提升 9.4% (并发 1) / 25.5% (并发 4) / 20.7% (并发 16), 提示吞吐提升 1.4%~4.0%, 端到端时延下降 0.2%~3.8%, 并发越高收益越大。
 - 系统行为: 消除了 prefill 尾段的第二次全模型 forward (attention、MoE、routing、TP collectives), 同一请求从两次调度、两次 kernel 启动收敛为一次; 代价是每个 checkpoint 请求多预留一个 Mamba 缓存块。
 - 团队协作: 本 PR 与 FlashKDA 上游 (FlashKDA#7) 和 vllm#52971 形成分层配合, Kimi-K3 KDA 性能线进入快速迭代阶段; 同时为其他 Mamba 模型复刻“单次前向导出 checkpoint”提供了可参考的契约模板。
 - 风险标记: 核心调度路径变更, 仅 FlashKDA 后端生效, 多模态输入未覆盖, 大 batch 性能未验证, 跨模块契约耦合

关联脉络

- PR #52971 Kimi-K3 FlashKDA 后续 PR (讨论中提及, 标题待核实): review 讨论中 ZeldaHuang 明确说明 #52971 已支持 spec-decoding, 计划在本 PR 合并后 rebase 其上, 双方避免重复实现。
- PR #53196 [Kimi K3][Perf] Export FlashKDA checkpoints in one packed call: 该 PR 是随 main 合并进本分支的基础能力提交, 实现“单次打包导出 checkpoint”, 本 PR 的元数据与 kernel 设计直接构建其上。
- PR #53212 [Kimi K3] Allocate internal checkpoints only from aligned starts: 同样经 main 合并进本分支, 确立了“仅从对齐起点分配内部 checkpoint”的约束, 本 PR 的 `_needs_internal_checkpoint` 与 metadata 对齐条件都延续该约束。
- PR #52606 [ROCm][Perf] Kimi-K3 Fused kernels for KDA prefill: 同属 Kimi-K3 KDA prefill 性能优化主线 (ROCm 侧融合内核), 与本 PR 一起体现 KDA prefill 在多个硬件后端的持续演进。