PR #43447 完整报告

vllm-project/vllm

[Prefix Caching] DeepSeekv4 - Support selective prefix-cache retention for sliding-window KV cache

合并时间：2026-06-04 15:48

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43447

执行摘要

- 一句话：DeepSeek V4 滑动窗口 KV cache 选择性保留与回收优化
- 推荐动作：值得精读，尤其是 prepend_n + free_blocks 的回收优先级设计以及 _validate_prefix_cache_retention_interval 的输入校验模式。建议在 DeepSeek V4 以外的滑动窗口模型（如 Mistral）上验证兼容性，并考虑将 retention 机制推广到 Mamba 组（当前 TODO）.

功能与动机

DeepSeek v4 prefix-cache 有效容量很低，例如在 TP8 8xB300 上，KV cache 容量约为 14.5 倍并发，但实际 prefix-cache 保留容量远低于此。微基准测试表明第二个请求发送后重放第一个请求即开始 miss。根因是新请求的滑动窗口块分配即使不断释放，也会频繁冲掉空闲队列中已有的缓存块。需要优先重用无缓存块，并选择性保留缓存检查点。

实现拆解

1. 新增 prepend_n 方法（vllm/v1/core/kv_cache_utils.py）：在 FreeKVCacheBlockQueue 中添加 prepend_n，将块插入空闲队列头部，与已有的 append_n 对称。
2. free_blocks 支持前置插入（vllm/v1/core/block_pool.py）：free_blocks 新增 prepend 参数，为 True 时调用 prepend_n 而非 append_n。
3. remove_skipped_blocks 分离有 / 无缓存块（vllm/v1/core/single_type_kv_cache_manager.py）：在 SWAManager.remove_skipped_blocks 中将释放的块分为 removed_uncached_blocks 与 removed_cached_blocks，然后分别调用 free_blocks（无缓存块 prepend=True，有缓存块 prepend=False）。
4. cache_blocks 传递 retention_interval：SingleTypeKVCacheManager.cache_blocks 新增 retention_interval 参数，替代原来的 alignment_tokens；KVCacheCoordinator 从环境变量读取 VLLM_PREFIX_CACHE_RETENTION_INTERVAL 并传给各 manager。
5. SlidingWindowManager 选择性缓存：在 cache_blocks 中根据 retention_interval 使用 reachable_block_mask 或 cache_blocks_at_boundaries 仅缓存指定边界处的块（如每 interval 保留一个 tail，或仅保留最新 replay boundary）。

6. 环境变量配置与验证（vllm/envs.py 添加 VLLM_PREFIX_CACHE_RETENTION_INTERVAL；vllm/v1/core/kv_cache_coordinator.py 添加 _validate_prefix_cache_retention_interval 检查非负且为 scheduler_block_size 的倍数，以及模型是否包含滑动窗口组）。
7. 单元测试覆盖：tests/v1/core/test_prefix_caching.py 新增 6 个测试用例，覆盖保留间隔对齐、无效值拒绝、回收后持久化、仅最新 boundary、MTP 场景、空闲队列顺序；tests/v1/core/test_kv_cache_utils.py 新增 test_free_kv_cache_block_queue_prepend_n。

关键文件：

- 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 free, cache_blocks, reachable_block_mask, remove_skipped_blocks）：核心逻辑变更：free 和 cache_blocks 方法修改回收策略与缓存保留间隔；SlidingWindowManager 新增 cache_blocks_at_boundaries 支持选择性缓存。
- vllm/v1/core/kv_cache_utils.py（模块 缓存工具；类别 source；类型 core-logic；符号 prepend_n）：新增 FreeKVCacheBlockQueue.prepend_n，实现块前置插入，是回收优先级调整的基础。
- vllm/v1/core/block_pool.py（模块 块池；类别 source；类型 core-logic；符号 free_blocks）：free_blocks 新增 prepend 参数，控制空闲块插入方向，连接队列基础工具与上层回收策略。
- vllm/v1/core/kv_cache_coordinator.py（模块 协调器；类别 source；类型 dependency-wiring；符号 _validate_prefix_cache_retention_interval, init）：新增 _validate_prefix_cache_retention_interval 函数，读取环境变量并校验；在 __init__ 中传递给各 manager。
- vllm/v1/core/kv_cache_utils.py（模块 测试；类别 test；类型 test-coverage；符号 test_free_kv_cache_block_queue_prepend_n）：测试核心队列操作，确保 prepend_n 正确维护双向链表。
- tests/v1/core/test_prefix_caching.py（模块 测试；类别 test；类型 test-coverage；符号 test_hybrid_local_kv_retention_interval_aligns_in_manager, test_hybrid_local_kv_retention_interval_rejects_invalid, test_hybrid_local_kv_retention_interval_survives_recycling, fill_request）：全面测试 retention_interval 的各种模式（对齐、无效值、回收、MTP、空闲队列顺序）。
- vllm/envs.py（模块 环境配置；类别 source；类型 core-logic）：定义 VLLM_PREFIX_CACHE_RETENTION_INTERVAL 环境变量及其默认值。

关键符号：prepend_n, free_blocks, _validate_prefix_cache_retention_interval, cache_blocks_at_boundaries, reachable_block_mask, free, cache_blocks

关键源码片段

vllm/v1/core/single_type_kv_cache_manager.py

核心逻辑变更：free 和 cache_blocks 方法修改回收策略与缓存保留间隔；SlidingWindowManager 新增 cache_blocks_at_boundaries 支持选择性缓存。

```
# vllm/v1/core/single_type_kv_cache_manager.py (SlidingWindowManager.free 核心片段 )
def free(self, request_id: str) -> None:
```

```
    req_blocks = self.req_to_blocks.pop(request_id, [])
    if not req_blocks:
        self.num_cached_block.pop(request_id, None)
        return
    # 先调用 remove_skipped_blocks 将滑动窗口中被跳过的块释放
    self.remove_skipped_blocks(request_id, force=True)
    # 分离有哈希（缓存块）和无哈希（纯临时）的块
    uncached = [b for b in req_blocks if b.block_hash is None]
    cached = [b for b in req_blocks if b.block_hash is not None]
    # 无缓存块前置到空闲队列头部，优先被新请求重用，避免冲刷缓存
    if uncached:
        self.block_pool.free_blocks(uncached, prepend=True)
    if cached:
        self.block_pool.free_blocks(cached, prepend=False)
    # 清理缓存计数
    self.num_cached_block.pop(request_id, None)
```

vllm/v1/core/kv_cache_utils.py

新增 FreeKVCacheBlockQueue.prepend_n，实现块前置插入，是回收优先级调整的基础。

```
# vllm/v1/core/kv_cache_utils.py (FreeKVCacheBlockQueue.prepend_n)
def prepend_n(self, blocks: list[KVCacheBlock]) -> None:
    """Put a list of blocks at the front of the free list."""
    if len(blocks) == 0:
        return

    # 获取当前第一个真实块（fake head 之后）
    first_block = self.fake_free_list_head.next_free_block
    assert first_block is not None, (
        "next_free_block of fake_free_list_head should always exist"
    )

    # 将 blocks 逐个链接到 fake_head 之后
    prev_block = self.fake_free_list_head
    for block in blocks:
        block.prev_free_block = prev_block
        prev_block.next_free_block = block
        prev_block = block

    # 将原第一个块接到新 blocks 末尾
    prev_block.next_free_block = first_block
    first_block.prev_free_block = prev_block

    self.num_free_blocks += len(blocks)
```

vllm/v1/core/block_pool.py

free_blocks 新增 prepend 参数，控制空闲块插入方向，连接队列基础工具与上层回收策略。

```
# vllm/v1/core/block_pool.py (BlockPool.free_blocks)
```

```
def free_blocks(
    self, ordered_blocks: Iterable[KVCacheBlock], prepend: bool = False
) -> None:
    """Free a list of blocks. The blocks should be ordered by their eviction
    priority, where the first block will be evicted first.

    Args:
        ordered_blocks: A list of blocks to free ordered by eviction priority.
        prepend: Whether to put newly-freed blocks at the front of the free
            queue to be prioritized for reuse.
    """
    blocks_list = list(ordered_blocks)
    for block in blocks_list:
        block.ref_cnt -= 1
    # 只释放引用计数归零且非 null 的块
    freed_blocks = [
        block for block in blocks_list
        if block.ref_cnt == 0 and not block.is_null
    ]
    if prepend:
        self.free_block_queue.prepend_n(freed_blocks)
    else:
        self.free_block_queue.append_n(freed_blocks)
```

评论区精华

Review 中主要讨论了以下几点：

- MTP 支持：用户 dafeliton 询问 PR 是否支持 MTP（此前未覆盖），ivanium 指出需要额外调整，wzhao18 随后添加了 MTP 支持并得到确认。
- 环境变量设计：ivanium 建议去掉 CLI knob --prefix-cache-retention-interval 改用环境变量 VLLM_PREFIX_CACHE_RETENTION_INTERVAL，wzhao18 采纳并更新。
- Mamba 支持：ivanium 提到该特性也可适用于 Mamba 组，但 wzhao18 认为需进一步协调 Mamba 模式，推迟到后续 PR。
- prepend 优化：ivanium 指出即使无 retention 机制，将无缓存块前置到空闲队列也能独立帮助当前 DSV4 性能，wzhao18 表示同意并在 free 中添加了类似逻辑。
  - MTP 支持缺失与修复 (question): wzhao18 添加了 MTP 支持，dafeliton 确认测试通过（命中率提升至 19.8%）。
  - 环境变量与 CLI knob 取舍 (design): wzhao18 采纳，改为 VLLM_PREFIX_CACHE_RETENTION_INTERVAL 环境变量。
  - Mamba 组支持范围 (design): 推迟到未来 PR，当前仅作用于滑动窗口组。
  - prepend 逻辑的独立价值 (performance): wzhao18 同意并在 free 中实现了分离模式（无缓存 prepend，有缓存 append）。

风险与影响

- 风险：
  1. 核心路径变更风险：BlockPool.free_blocks 和 SingleTypeKVCacheManager.free 是 KV cache 的核心回收路径，新增 prepend 参数可能影响所有模型的块回收行为。验证逻辑确保默认 prepend=False 保持旧行为，但回归测试需覆盖多种 attention 类型（全量、滑动窗口、Mamba、MLA）。
  2. MTP 特殊处理：MTP 场景需要额外丢弃一个 lookahead block，当前实现可能未完全覆盖所有 speculative decoding 配置（如不同 K 值），需进一步测试。
  3. 配置错误风险：VLLM_PREFIX_CACHE_RETENTION_INTERVAL 若设为非 scheduler_block_size 倍数的正数会报错，但若用户误设为 0（仅保留最新 boundary）可能意外导致历史前缀全丢失，需文档强调。
  4. 与现有功能兼容性：cache_blocks 参数从 alignment_tokens 改为 retention_interval，影响 KVCacheCoordinator 和 HybridKVCacheManager 的调用点，需确认所有调用方已适配。 - 影响：用户侧：DeepSeek V4 用户通过设置 VLLM_PREFIX_CACHE_RETENTION_INTERVAL（例如 0 或 32768）可大幅提升 prefix-cache 命中率，降低首 token 延迟和总推理时间。其他模型不受影响（默认 None 保持原行为）。系统侧：空闲队列操作变为 O(1) 头插，无额外开销；保留间隔检查仅影响缓存步骤的掩码计算，复杂度与块数线性，可忽略。团队侧：该 PR 修改了 SingleTypeKVCacheManager 和 BlockPool 接口，未来扩展新的 attention 类型时需注意回收语义。
- 风险标记：核心路径变更，MTP 支持待更多测试，配置默认值影响，回收策略变更需回归

关联脉络

- PR #44230 optimize the compressor 128 split cutedsl kernel: 同一 DeepSeek V4 模型的性能优化（压缩内核），属于同一模型的不同优化方向。