

PR #50894 完整报告

vllm-project/vllm

[Bugfix] Scale KV page size for hidden states extraction with TP

合并时间: 2026-09-14 22:07

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/50894>

执行摘要

- 一句话: TP 下 hidden state 提取按需放大 KV page, 修复启动断言崩溃
- 推荐动作: 值得精读, 尤其是维护 vllm/v1/core/kv_cache_utils.py 或调试 extract_hidden_states + TP 组合的人。关注三个设计决策: 一是用 $\text{cd}(\text{div}(\text{max_per_token}, \text{common_page}))$ 求最小整数倍数, 而不是无条件翻倍, 控制显存浪费; 二是同一 scale 同时作用于 block_size 与已显式设置的 page_size_padded, 维持 spec 内部比例; 三是把缩放放在 math.gcd 之前, 使 hidden-state 层仍能取到合法公约数。同时建议读者顺手确认两件事: 复合 UniformTypeKVCacheSpecs 场景下 spec 是否被漏缩, 以及后续是否需要为 $\text{scale} \geq 3$ 、MambaSpec 与已 padded 的 spec 补测试用例。

功能与动机

Issue #51016 报告: 使用 `--speculative-config '{"method": "extract_hidden_states", ...}'` 时, poolside/Laguna-S-2.1 (117.6B MoE, 混合 sliding/full attention, 48 层) 在 TP=1 可启动, $\text{TP} \geq 2$ 直接在 KVCacheSpecBase.__post_init__ 抛 AssertionError (`assert self.page_size_padded >= real_page_size`)。根因是 TP 把 KV head 分片 ($\text{num_kv_heads} / \text{TP}$) 压缩了 common KV page, 而 hidden-state cache spec 存的是完整未分片的 hidden_size, 每 token 成本可能反超被压缩的 page; issue 明确指出这“typically [affects] large models that require $\text{TP} \geq 2$ for serving”。PR body 给出的方案是: 在把 hidden-state 规格对齐到 common page 之前, 当 hidden-state 每 token 成本超过 common page 时放大所有目标 group 的 block size (“Scale up all target group block sizes when the hidden-state per-token cost exceeds the common page, before aligning hidden-state specs”)。

实现拆解

1. 问题定位在 get_kv_cache_groups (vllm/v1/core/kv_cache_utils.py) 的 hidden-state 收尾分支: 先由 get_uniform_page_size 得到 common_page, 再为每个 hidden spec 计算 $\text{per_token} = \text{num_kv_heads} * \text{head_size} * \text{get_dtype_size}(\text{dtype})$, 随后 `replace(spec, block_size=new_bs, page_size_padded=common_page)`。TP > 1 时 attention spec 的 num_kv_heads 已被分片, common_page 缩小, 而 hidden spec 的 $\text{num_kv_heads}/\text{head_size}$ 仍然承载完整 hidden_size, 于是 $\text{per_token} > \text{common_page}$, page_size_padded 小于真实每 token 成本, 触发 KVCacheSpecBase.__post_init__ 断言。

2. 新增缩放 helper `_ensure_min_page_size(groups, common_page, hidden_specs) -> tuple[list[KVCacheGroupSpec], int]`: 取所有 hidden spec 中最大的 `max_per_token`; 若 `max_per_token <= common_page` 直接原样返回 (TP = 1 与常规模型零开销早退); 否则 `scale = cdiv(max_per_token, common_page)`、`common_page *= scale`, 并对每个 group 用 `replace(s, block_size=s.block_size * scale)` 生成新 spec。对于 `AttentionSpec/MambaSpec` 且 `page_size_padded` is not None 的 spec, 额外把 `page_size_padded * scale` 一并放大, 避免 `block_size` 与 `padded page` 的比例关系被破坏。
3. 主流程接入: 在 hidden-state 分支的 `common_page = get_uniform_page_size([g.kv_cache_spec for g in groups])` 之后插入一行 `groups, common_page = _ensure_min_page_size(groups, common_page, hidden_specs)`。顺序很关键——缩放发生在 `group_block_size = math.gcd(*(g.kv_cache_spec.block_size for g in groups))` 之前, 因此 hidden-state 层算出的 `new_bs` 仍是缩放后所有 group block size 的公约数, 共享 block table 的整除前提不被破坏; `wasted_bytes` 日志与 `aligned = replace(spec, block_size=new_bs, page_size_padded=common_page)` 逻辑保持不变。
4. 行为变化: 修复把“不可能对齐”变成“可对齐但有少量浪费”。以新增测试的数值推导, `scale = cdiv(6144, 4096) = 2`, 目标 group 的 block size 从 16 变 32、common page 从 4096 变 8192, hidden-state 层最终得到 `block_size = 1`、`page_size_padded = 8192`, 每 block 浪费 2048 字节 (25%)。这也意味着在 hidden state 提取场景下, 用户 `--block-size` 指定的值可能被内部改写。
5. 测试配套: `tests/v1/core/test_kv_cache_utils.py` 新增 `test_hidden_states_with_tp_scales_page_size`, 用 `new_kv_cache_spec(block_size=16, num_kv_heads=2, head_size=64, dtype=bfloat16)` 模拟 TP=4 分片后的 4096 字节 page, 用 `HiddenStateCacheSpec(num_kv_heads=6, head_size=512, dtype=bfloat16)` 构造 6144 字节的每 token 成本, 断言 hidden-state 层仍在分组中且非 hidden group 的 block size 被放大; 该用例在修复前正是触发断言的组合。其余提交为 8 次 main 同步 merge、1 次采纳 reviewer docstring 建议的 co-authored commit 与 1 次 ruff 格式化, 无返工式重写。

关键文件:

- `vllm/v1/core/kv_cache_utils.py` (模块 KV 缓存管理; 类别 source; 类型 core-logic; 符号 `_ensure_min_page_size, get_kv_cache_groups`): 唯一源码改动文件: 新增 `_ensure_min_page_size` 并在 `get_kv_cache_groups` 的 hidden-state 分支接入, 是断言崩溃的直接修复点。
- `tests/v1/core/test_kv_cache_utils.py` (模块 KV 缓存管理; 类别 test; 类型 test-coverage; 符号 `test_hidden_states_with_tp_scales_page_size`): 新增回归用例, 用 TP = 4 分片后的 KV page 与更大的 hidden-state 每 token 成本复现原断言场景, 并校验 block size 被放大。

关键符号: `_ensure_min_page_size, get_kv_cache_groups, _largest_divisor_at_most, test_hidden_states_with_tp_scales_page_size`

关键源码片段

vllm/v1/core/kv_cache_utils.py

唯一源码改动文件：新增 `_ensure_min_page_size` 并在 `get_kv_cache_groups` 的 `hidden-state` 分支接入，是断言崩溃的直接修复点。

vllm/v1/core/kv_cache_utils.py —— 本次新增的缩放 helper

```
def _ensure_min_page_size(
    groups: list[KVCacheGroupSpec],
    common_page: int,
    hidden_specs: dict[str, HiddenStateCacheSpec],
) -> tuple[list[KVCacheGroupSpec], int]:
    """Scale up group block sizes so the common page is at least as large as
    the biggest hidden-state per-token cost.
```

This protects the hidden state extraction feature, where we store hidden states in the KV cache, and return them to clients using a custom hidden state connector.

Returns the (possibly scaled) groups and updated common page size.

"""

hidden state 存的是完整 hidden_size，不做 TP 分片，
因此其每 token 成本可能大于已被 num_kv_heads / TP 缩小后的 common page。

```
max_per_token = max(
    spec.num_kv_heads * spec.head_size * get_dtype_size(spec.dtype)
    for spec in hidden_specs.values()
)
```

if max_per_token <= common_page:

page 已经足够大（例如 TP = 1）：完全不缩放，保持既有显存布局。
 return groups, common_page

向上取整得到最小的整数缩放倍数，避免多放大一档白白占用显存。

scale = cdiv(max_per_token, common_page)

common_page *= scale

scaled: list[KVCacheGroupSpec] = []

for g in groups:

s = g.kv_cache_spec

kw: dict[str, int] = {"block_size": s.block_size * scale}

显式 padded 过的 Attention / Mamba spec 必须同步放大，

否则 block_size 与 page_size_padded 的比例关系会被破坏。

if isinstance(s, (AttentionSpec, MambaSpec)) and s.page_size_padded is not None:

kw["page_size_padded"] = s.page_size_padded * scale

scaled.append(KVCacheGroupSpec(g.layer_names, replace(s, **kw)))

return scaled, common_page

get_kv_cache_groups 的 hidden-state 收尾分支（前置分支已处理 uniform spec、

packed groups、unify fallback 等场景，此处只保留与本修复相关的部分）

Add hidden-state layers back with page aligned to the common page.

if hidden_specs:

common_page = get_uniform_page_size([g.kv_cache_spec for g in groups])

```

# TP may shrink the common page below hidden-state per-token cost.
groups, common_page = _ensure_min_page_size(groups, common_page, hidden_specs)
# 注意顺序：先缩放再取 gcd，hidden-state 层的 block size 才仍是
# 缩放后所有 group block size 的公约数，共享 block table 的前提不被破坏。
group_block_size = math.gcd(*(g.kv_cache_spec.block_size for g in groups))
for name, spec in hidden_specs.items():
    per_token = spec.num_kv_heads * spec.head_size * get_dtype_size(spec.dtype)
    max_block_size = max(common_page // per_token, 1)
    new_bs = _largest_divisor_at_most(group_block_size, max_block_size)
    wasted_bytes = common_page - new_bs * per_token
    logger.info(
        "Using block size %d for hidden-state cache layer %s; "
        "page alignment wastes %d bytes (%.2f%%) per block",
        new_bs,
        name,
        wasted_bytes,
        wasted_bytes / common_page * 100,
    )
    aligned = replace(spec, block_size=new_bs, page_size_padded=common_page)
    groups.append(KVCacheGroupSpec([name], aligned))

_annotate_eagle_groups(vllm_config, kv_cache_spec, groups)
_warn_if_unannotated_eagle_mamba(vllm_config, groups)
return groups

```

tests/v1/core/test_kv_cache_utils.py

新增回归用例，用 TP = 4 分片后的 KV page 与更大的 hidden-state 每 token 成本复现原断言场景，并校验 block size 被放大。

```

# tests/v1/core/test_kv_cache_utils.py
def test_hidden_states_with_tp_scales_page_size():
    """When TP shrinks KV pages below the hidden-state per-token cost,
    get_kv_cache_groups must scale up target block sizes so that the
    common page accommodates the unsharded hidden states."""
    # 模拟 TP = 4 把 8 个 KV head 切成每 rank 2 个：
    # KV page = block_size(16) * num_kv_heads(2) * head_size(64) * dtype(2)
    # = 4096 字节。
    kv_spec = new_kv_cache_spec(
        block_size=16,
        num_kv_heads=2,
        head_size=64,
        dtype=torch.bfloat16,
    )
    # hidden-state 每 token 成本 = num_hidden_states(6) * hidden_size(512)
    # * dtype(2) = 6144 字节，已经超过 4096 字节的 KV page。
    hs_spec = HiddenStateCacheSpec(
        block_size=16,
        num_kv_heads=6,
        head_size=512,
    )

```

```

        dtype=torch.bfloat16,
    )
    specs = {
        "target.0.attn": kv_spec,
        "target.1.attn": kv_spec,
        "cache_only_layers.48": hs_spec,
    }

    groups = get_kv_cache_groups(_grouping_config(), specs)

    # 断言 1: hidden-state 层仍在分组中, 且 KVCacheSpecBase 的
    # page_size_padded >= real_page_size 断言不再被触发。
    all_layers = {name for g in groups for name in g.layer_names}
    assert "cache_only_layers.48" in all_layers

    # 断言 2: 目标 group 的 block size 被放大。
    # 推导: scale = cdiv(6144, 4096) = 2, 即 block size 16 -> 32,
    # common page 4096 -> 8192; hidden-state 层则落到 block_size = 1、
    # page_size_padded = 8192, 每 block 浪费 2048 字节 (25%) 。
    for g in groups:
        if "cache_only_layers.48" not in g.layer_names:
            assert g.kv_cache_spec.block_size > kv_spec.block_size

```

评论区精华

讨论集中在两处, 均由维护者 benchislett 提出并被作者采纳: 其一是在 diff 上追问 “is there no helper function for ‘ensure page size is at least N’?”——初版实现把缩放逻辑 (计算 max_per_token、cdiv 求 scale、遍历 group 做 replace) 直接内联在 `get_kv_cache_groups` 里, 作者回复 “extracted into helper function”, 最终抽出 `_ensure_min_page_size` 并让它同时返回更新后的 commonpage, 使主流程收敛为一行调用。其二是对 helper docstring 的 nit (line 2241), benchislett 直接在 suggestion 中给出补充文本, 说明这是为 “hidden state extraction” 服务——hidden states 存放在 KV cache 中并通过自定义 hidden state connector 回传客户端, 作者按建议合并。最终意见为 “One more comment, otherwise approved. Will mark as ready, please run CI once last comment is addressed.”, 随后进入 CI 循环。另有流程性信息: claude[bot] 声明该 PR 来自 fork, 自动 review 被禁用, 需维护者手动触发; mergify[bot] 多次提示 main 分支漂移导致的冲突, 作者共触发 4 次 CI build (#87686、#88334、#88803、#88806)。

- 内联的 page 缩放逻辑是否应抽成独立 helper (design): 作者回复 “extracted into helper function”, 抽出 `_ensure_min_page_size` 并让其同时返回缩放后的 groups 与更新后的 common_page, 主流程收敛为一行调用, 可读性与复用性提升。
- helper docstring 需要解释 hidden_specs 的业务含义 (documentation): 建议文本被完整合并 (commit a883956 以 Co-authored-by 形式采纳), helper 的 docstring 明确了它保护的是 hidden state extraction 特性。
- fork PR 的自动化 review 与合并前 CI 流程 (other): benchislett 以 “One more comment, otherwise approved. Will mark as ready, please run CI once last comment is

addressed.” 批准并在最后一条意见解决后放行，PR 最终合并（merged_by: benchislett）。

风险与影响

- 风险：1) 回归面：_ensure_min_page_size 会放大所有非 hidden group 的 block_size，因此在 extract_hidden_states 场景下，--block-size 的用户配置可能被内部改写（如 16 → 32）。这会改变 KV cache 物理页布局、prefix cache 的复用粒度（hash 以 block 为单位）以及 attention backend / CUDA graph 对 block size 的隐含假设，属于“仅在特定特性开启时触发、但触发后是全局行为改变”的类别。2) 兼容性：缩放只对 isinstance(s, (AttentionSpec, MambaSpec)) and s.page_size_padded is not None 的 spec 同步放大 padded page；若 _get_kv_cache_groups_uniform_page_size 某条路径产出的 group spec 是复合的 UniformTypeKVCacheSpecs（内部子 spec 各自持有 block_size），当前实现只替换外层 block_size，内层子 spec 与其 page_size_padded 是否随之同步无法从现有材料确认，存在比例不一致的隐患（材料不足以定论，建议维护者复核该分支）。3) 性能 / 内存：scale 用 cdiv 取最小整数倍，未过度放大；但 hidden-state 层在新 common page 下 new_bs 可能退化为 1（测试数值下即为 1），意味着每个 token 一个 block，block table 与调度开销上升；同时目标 group block 变大也会降低 prefix cache 的复用粒度，KV cache 可用 block 数在固定显存预算下相应减少。4) 测试覆盖：新增用例只覆盖 scale = 2、单 group、全为 AttentionSpec 的组合；未覆盖 scale ≥ 3、MambaSpec、page_size_padded 已显式设置、packed groups 提前返回分支以及 --disable-hybrid-kv-cache-manager 路径。5) 正确性边界：wasted_bytes = common_page - new_bs * per_token 在缩放后保证非负，缩放逻辑本身不改变 hidden state 的写入量，只改变对齐方式，因此不存在数据截断风险。
- 影响：用户侧：依赖 extract_hidden_states 采集训练数据的大模型（如 117.6B MoE 的 poolside/Laguna-S-2.1）在 TP ≥ 2 时从“启动即崩”恢复为可正常启动并产出 hidden states；TP = 1 或 hidden state 成本未超 page 的模型走早退分支，行为与显存布局完全不变。系统侧：KV cache group 构建新增一次页大小协商决策，block size 不再严格等于用户配置值，调度器分配给各 group 的 block 数与 prefix cache 命中粒度会随之变化；hidden-state 层与目标注意力层仍需共享 block table，本次改动保证该整除关系成立。团队侧：kv-cache-manager 子系统需要把“block size 可被内部放大以容纳 hidden-state 每 token 成本”纳入既有语义认知，后续改动该文件的开发者需要同时检查这条缩放路径。
- 风险标记：核心路径变更，TP 场景 block size 被内部放大，prefix cache 粒度变化，复合 spec 缩放未验证，测试数值组合单一

关联脉络

- PR #56791 [Bugfix] Trim stale consequence claims from unannotated-eagle warning: 改动同一文件 vllm/v1/core/kv_cache_utils.py 与同一测试文件 tests/v1/core/test_kv_cache_utils.py，且本次新增 helper 紧邻该 PR 涉及的 _warn_if_unannotated_eagle_mamba，同属 KV cache group 构建阶段对特殊 spec（Eagle、hidden state）的边界处理。

- PR #55823 [Bugfix][KV Offload] Reuse in-flight async lookup probes: 同属 v1 缓存子系统 (vllm/v1/kv_offload 与 vllm/v1/core) 的边界修复, 都围绕 block/page 粒度与缓存复用行为, 可作为同类“特定组合下才暴露”的问题治理思路参照。
- PR #54934 [CI][CPU] Add speculative-decoding coverage to CPU CI: 投机解码特性线的配套改动, 说明 extract_hidden_states 所属的 speculative-decoding 路径近期在持续补平台与场景覆盖, 本 PR 属于该线上“TP 组合”缺口的补齐。