PR #51704 完整报告

vllm-project/vllm

[5/N][KV-Cache Layout Refactor] Backend-published KV packing via customize_spec

合并时间：2026-08-14 21:47

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51704

执行摘要

- 一句话：后端发布 KV 打包规格，spec 回归纯数据
- 推荐动作：值得精读。尤其关注 customize_spec 的临时钩子设计——它平衡了当前架构约束与最终目标（后端直接构建 spec），并通过配置时间探针接入展示了如何渐进式迁移。review 中发现的遗漏问题也提示了此类重构中全局查找调用点的重要性。

功能与动机

RFC #42082 指出当前 attention 后端存在语义和物理上不同的 KV cache 布局，导致 KV-connector 代码充满 is_mamba / is_mla 标志，并与后端紧密耦合。本 PR 先把打包尺寸知识从 spec 中剥离，交给拥有它的后端发布，使 spec 成为纯数据，为标准化 [B, H, N, C] 页面布局做准备。

实现拆解

1. AttentionSpec 增加打包字段并统一页面大小推导（vllm/v1/kv_cache_interface.py）：新增 num_head_slots（逻辑页面 H）和 state_content_bytes（逻辑页面 C），在 __post_init__ 中默认 head_size_v = head_size；unpadded_page_size_bytes 变为 num_heads * storage_block_size * state_content_size_bytes；real_page_size_bytes 成为其别名。同时 KVQuantMode 从单个 TURBOQUANT 拆分为 K8V4 / 4BIT_NC / K3V4_NC / 3BIT_NC 四种。
2. 引入 AttentionBackend.customize_spec 钩子（vllm/v1/attention/backend.py）：默认原样返回 spec；FlashInfer（NVFP4 将 K/V 拆成两个 head slot）、Triton（per-token-head 内联 fp32 scale）、TurboQuant（K+V 合并进单 slot）和 MLACommonBackend（per-token-head 打包与 fp8_ds_mla 656 字节布局）分别覆写。
3. 模型层与平台层接入钩子：vllm/model_executor/layers/attention/attention.py 的 get_kv_cache_spec 在 SW 分支通过 customize_spec 计算每 token 页面大小，并删除 turboquant 特判；vllm/platforms/interface.py 的 per_token_page_bytes 与 _align_hybrid_block_size 也改为经 customize_spec 计算（review 指出最初遗漏，后续提交补齐）。
4. 删除 TQFullAttentionSpec 与相关注册：single_type_kv_cache_manager.py 移除 TQFullAttentionSpec 注册，tests/v1/test_kv_cache_spec_registry.py 相应删减。

5. 测试配套：tests/v1/core/test_kv_cache_utils.py 改用 TritonAttentionBackend.customize_spec 验证 per-token-head 大小；tests/quantization/test_turboquant.py 与 test_dsv4_packed_zeroer_geometry.py 适配新的打包发布方式。

关键文件：

- vllm/v1/kv_cache_interface.py（模块 缓存接口；类别 source；类型 core-logic；符号 post_init, num_heads, state_content_size_bytes, real_page_size_bytes）：核心变更文件：AttentionSpec 增加 num_head_slots / state_content_bytes 打包字段，页面大小统一推导，删除 TQFullAttentionSpec，KVQuantMode 拆分。
- vllm/model_executor/layers/attention/attention.py（模块 注意力层；类别 source；类型 data-contract；符号 get_kv_cache_spec）：get_kv_cache_spec 中 SW 分支改用 customize_spec 计算页面大小，删除 turboquant 特判分支，是模型层接入钩子的关键入口。
- vllm/v1/attention/backends/flashinfer.py（模块 注意力后端；类别 source；类型 core-logic；符号 customize_spec）：FlashInfer 后端新增 customize_spec，将 NVFP4 打包知识收归己有，是后端发布打包的典型示例。
- vllm/v1/attention/backend.py（模块 后端基类；类别 source；类型 core-logic；符号 customize_spec）：定义 customize_spec 钩子的基类，并说明其作为临时兼容 API 的定位。
- vllm/model_executor/layers/attention/mla_attention.py（模块 MLA 层；类别 source；类型 data-contract；符号 customize_spec, get_kv_cache_spec）：MLACommonBackend 实现 customize_spec，处理 per-token-head 量化打包，并在 get_kv_cache_spec 中为 fp8_ds_mla 发布 656 字节 state_content_bytes。
- vllm/v1/attention/backends/triton_attn.py（模块 注意力后端；类别 source；类型 core-logic；符号 customize_spec）：TritonAttentionBackend 新增 customize_spec，将 per-token-head 量化的内联 fp32 scale 计入 state_content_bytes。
- vllm/v1/attention/backends/turboquant_attn.py（模块 注意力后端；类别 source；类型 core-logic；符号 customize_spec）：TurboQuantAttentionBackend 新增 customize_spec，以 TurboQuantConfig 计算 slot 大小并发布 state_content_bytes，替代被删除的 TQFullAttentionSpec。
- vllm/platforms/interface.py（模块 平台层；类别 source；类型 dependency-wiring；符号 per_token_page_bytes, _align_hybrid_block_size）：配置时间页面探针接入 customize_spec，review 中曾指出三处遗漏，后续提交补齐。
- vllm/v1/core/single_type_kv_cache_manager.py（模块 缓存管理；类别 source；类型 core-logic；符号 register_all_kvcache_specs）：移除 TQFullAttentionSpec 的注册，因为该 spec 类已删除，TurboQuant 页面大小改由后端发布。
- tests/v1/core/test_kv_cache_utils.py（模块 缓存测试；类别 test；类型 test-coverage；符号 test_unpadded_page_size_includes_per_token_head_scales）：测试改为通过 TritonAttentionBackend.customize_spec 验证 per-token-head 打包大小，并删除不再有意义的对照测试。

关键符号：customize_spec, num_heads, state_content_size_bytes, unpadded_page_size_bytes, real_page_size_bytes, post_init, get_kv_cache_spec, per_token_page_bytes, _align_hybrid_block_size, merge

关键源码片段

vllm/v1/kv_cache_interface.py

核心变更文件：AttentionSpec 增加 num_head_slots / state_content_bytes 打包字段，页面大小统一推导，删除 TQFullAttentionSpec，KVQuantMode 拆分。

```
@dataclass(frozen=True, kw_only=True)
class AttentionSpec(KVCacheSpec):
    # 基础字段：KV 头数、头维度、dtype 等
    num_kv_heads: int
    head_size: int
    dtype: torch.dtype
    head_size_v: int = None # type: ignore[assignment]
    kv_quant_mode: KVQuantMode = KVQuantMode.NONE
    page_size_padded: int | None = None
    indexes_kv_by_block_stride: bool = False
    # 以下两个字段由后端通过 customize_spec 发布：
    # num_head_slots —— 标准布局 [B, H, N, C] 中的 H（当 K/V 分成独立槽位时）
    # state_content_bytes —— 每个 (head slot, state) 单元的字节数 C
    num_head_slots: int | None = None
    state_content_bytes: int | None = None

    def __post_init__(self):
        # head_size_v 默认与 head_size 相同；frozen dataclass 需要 object.__setattr__
        if self.head_size_v is None:
            object.__setattr__(self, "head_size_v", self.head_size)

    @property
    def num_heads(self) -> int:
        # 打包布局可能把每个 KV head 拆成多个 head slot（如 NVFP4 的 K/V 分离），
        # 此时 num_head_slots 覆盖默认的 num_kv_heads。
        if self.num_head_slots is not None:
            return self.num_head_slots
        return self.num_kv_heads

    @property
    def state_content_size_bytes(self) -> int:
        # 每个 (head slot, state) 单元的字节数；后端显式发布时使用，
        # 否则回退到稠密 K/V 的 (head_size + head_size_v) * dtype 大小。
        if self.state_content_bytes is not None:
            return self.state_content_bytes
        return (self.head_size + self.head_size_v) * get_dtype_size(self.dtype)

    @property
```

```
    def unpadded_page_size_bytes(self) -> int:
        # 页面大小统一推导：H * N * C，其中 N 是 storage_block_size。
        return self.num_heads * self.storage_block_size * self.state_content_size_bytes
```

vllm/model_executor/layers/attention/attention.py

get_kv_cache_spec 中 SW 分支改用 customize_spec 计算页面大小，删除 turboquant 特判分支，是模型层接入钩子的关键入口。

```
  def get_kv_cache_spec(self, vllm_config: VllmConfig) -> KVCacheSpec | None:
      # ... 前置检查，encoder-only 返回 None ...
      quant_mode = get_kv_quant_mode(self.kv_cache_dtype)
      if self.sliding_window is not None:
          # SW 场景：块大小与主注意力解耦，需要先探测每 token 页面大小
          shared_page = vllm_config.cache_config.skip_page_size_padded
          # 打包方式由后端拥有，调用 customize_spec 获取打包后的每 token 大小
          sw_per_token = self.attn_backend.customize_spec(
              SlidingWindowSpec(
                  block_size=1,
                  num_kv_heads=self.num_kv_heads,
                  head_size=self.head_size,
                  head_size_v=self.head_size_v,
                  dtype=self.kv_cache_torch_dtype,
                  kv_quant_mode=quant_mode,
                  sliding_window=self.sliding_window,
              )
          ).real_page_size_bytes
          # 根据打包后的大小选择合适的内核 block size
          sw_block_size = _largest_kernel_block_within(
              self.attn_backend, vllm_config, sw_per_token, shared_page, block_size
          )
          return SlidingWindowSpec(
              block_size=sw_block_size,
              num_kv_heads=self.num_kv_heads,
              head_size=self.head_size,
              head_size_v=self.head_size_v,
              dtype=self.kv_cache_torch_dtype,
              kv_quant_mode=quant_mode,
              sliding_window=self.sliding_window,
              page_size_padded=shared_page,
          )
      else:
          # 普通 full attention：直接返回 spec，打包信息由后端在后续环节 customize
          return FullAttentionSpec(
              block_size=block_size,
              num_kv_heads=self.num_kv_heads,
              head_size=self.head_size,
              head_size_v=self.head_size_v,
              dtype=self.kv_cache_torch_dtype,
              kv_quant_mode=quant_mode,
```

)
```

### vllm/v1/attention/backends/flashinfer.py

FlashInfer 后端新增 customize_spec，将 NVFP4 打包知识收归己有，是后端发布打包的典型示例。

```
@classmethod
def customize_spec(cls, spec: "AttentionSpec") -> "AttentionSpec":
    """NVFP4 将 K 和 V 存储为独立的 per-head 槽位：fp4 数据 + fp8 block scales。"""
    # 若已有显式打包信息或非 NVFP4，保持不变
    if spec.state_content_bytes is not None or not spec.kv_quant_mode.is_nvfp4:
        return spec
    # nvfp4_kv_cache_full_dim 将 head_dim 换算为 fp4 打包后的完整维度
    hs_k = nvfp4_kv_cache_full_dim(spec.head_size)
    hs_v = nvfp4_kv_cache_full_dim(spec.head_size_v)
    assert hs_k == hs_v, "nvfp4 with asymmetric K/V head sizes not yet supported"
    # K/V 各占一个 head slot，因此 num_head_slots 翻倍；
    # state_content_bytes 为单个 head slot 的 fp4 数据字节数。
    return replace(
        spec,
        num_head_slots=2 * spec.num_kv_heads,
        state_content_bytes=hs_k * get_dtype_size(spec.dtype),
    )
```

评论区精华

AMD 工程师 okorzh-amd 在 review 中指出关键问题：vllm/platforms/interface.py 中 _align_hybrid_block_size 是四个配置时间页面探针中唯一使用 customize_spec 的地方，其余三处（per_token_page_bytes、同函数下面的 else 分支、attention.py 的 sw_per_token）仍直接读 .page_size_bytes，会得到稠密页面大小而不是打包后的大小。LucasWilkinson 回复 "good catch fixed in ..." 并在后续提交中修复。另外 okorzh-amd 质疑 test_unpadded_page_size_without_quant_matches_real_page 是否还需要，LucasWilkinson 同意并删除。CI 讨论中 njhill 提示 test_dsv4_packed_zeroer_geometry 的 shape 错误可能与本 PR 相关，LucasWilkinson 通过调整测试适配；AndreasKaratzas 确认 AMD 视角一切正常。

- 配置时间探针遗漏 customize_spec 调用 (correctness): LucasWilkinson 回复 "good catch fixed in ..."，并在后续提交 ab2a4342 中补齐其余调用点。
- test_unpadded_page_size_without_quant_matches_real_page 是否保留 (testing): 确认删除，因为该测试在打包知识移出 spec 后已无意义。
- CI 失败 test_dsv4_packed_zeroer_geometry (correctness): 测试文件被修改（+1 行），CI 最终通过。
- AMD CI 回归确认 (testing): AndreasKaratzas 最终确认 "From AMD tests perspective, everything looks good :)"，PR 合并。

风险与影响
```

- 风险：页面大小计算逻辑集中到后端，若某后端未正确实现 customize_spec 或遗漏调用点，会导致 KV cache 分配尺寸错误（如 review 中发现的三处探针遗漏）。real_page_size_bytes 变为别名后，TPU 等后端可能受影响（代码中有 TODO 标记需跟进）。删除 TQFullAttentionSpec 涉及多处引用迁移，若遗漏会引入运行时导入错误。fp8_ds_mla 的 656 字节布局硬编码在 mla_attention.py，对模型版本（如 deepseek_v4）敏感，测试覆盖有限。
- 影响：影响面覆盖 KV cache 分配、页大小计算、多注意力后端（FlashInfer、Triton、TurboQuant、MLA）以及平台层配置探测。对使用 NVFP4、per-token-head 量化、TurboQuant、fp8_ds_mla 的用户存在潜在行为变化，需关注回归。对团队而言，这是 KV 缓存布局标准化的关键一步，降低了后续 6/N 布局标准化和注意力后端重构（#42449）的耦合成本。
- 风险标记：核心页面大小计算重构，后端钩子遗漏风险，跨后端行为变更，配置时间探针不一致，删除 TQ spec 迁移风险

关联脉络

- PR #51612 [4/N][KV-Cache Layout Refactor] Promote local KV cache specs via a class-changing replace helper: 直接前置 PR，引入 replace_as 帮助函数，本 PR 在其基础上叠加并依赖该机制。
- PR #52148 [Attention] Fix FlashInfer SM12x prefill with sinks: PR 评论中引用的 main 分支失败修复，与 FlashInfer 后端相关。