

PR #56107 完整报告

vllm-project/vllm

[PCP][Spec Decode] Adds PCP support for single-module MTP and replicated DSpark.

合并时间: 2026-09-11 13:07

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56107>

执行摘要

- 一句话: PCP 下解锁单模块 MTP 与 DSpark 投机解码
- 推荐动作: 值得精读, 尤其是负责 MRv2、PCP/DCP 与投机解码的工程师。重点看三处:
 - (1) PCPManager.prepare_draft_prefill / restore_draft_prefill 定义的“首个 draft forward 复用 rank-local batch、后续 replicated draft 刷新全局 block table”契约, 这是本 PR 的核心设计决策;
 - (2) partition_batch 中 positions / seq_lens / logits_indices 的重写 (positions 必须来自 GPU 全局 batch、本地 logits 不再采样), 它解释了为什么必须移除 req_states 依赖;
 - (3) sparse_attn_indexer.py 与 check_attention_cp_compatibility 的口径放宽, 判断是否接受这种“隐式形状判定 + 校验只覆盖 target 层”的取舍。若要在生产启用, 建议先补一条 PCP + MTP/DSpark 的 e2e 用例, 并重跑作者未完成的整模型 acceptance 验证 (含 dummy/profile 路径)。

功能与动机

此前 PCPManager.validate_config 对投机解码是硬拒绝: raise NotImplementedError("MRV2 PCP does not support speculative decoding yet."), 导致 TP2 + PCP2 这类长上下文前置并行部署无法与 GLM-5.2 MTP / DSpark 这类 drafter 共存。PR 正文的目标是 “Adds V2 model-runner PCP support for single-module MTP and replicated DSpark.”, 并明确这是在当前 main 上 “consolidates #53653 and #53660 ... with a smaller implementation”, 即把两条早期实现路线合并、收敛为更小实现。核心矛盾是 MTP 首个 draft forward 仍需 rank-local 分片视角, 而后续 draft step 变为 replicated 的全局视角, 必须在同一批 buffer / block table / hidden states 上做视角切换; 同时 DSpark 整体 replicated 而 target 保持 PCP 分片, 需要避免 draft 侧重复分片。

实现拆解

1. 放宽准入校验: vllm/v1/worker/gpu/pcp_manager.py::PCPManager.validate_config 删除对 speculative_config 的硬拒绝; DSpark 要求 DCP=1 或 DCP=PCP, 否则只允许 method == "mtp" 且非 multi-module MTP; 同时移除 sparse MLA + PCP 必须 cudagraph_mode=NONE 的旧限制, 只保留拒绝 full CUDA graph。影响: 新组合可启动, 校验失败信息改为在启动期给出具体冲突值 (DCP/PCP)。
2. PCP 管理器补齐 draft prefill API: 新增 get_draft_input_buffers (返回 draft_prefill_batch or input_buffers)、prepare_draft_prefill (用 partition_batch() 保存的 _local_gather_idx 把全局 input_ids 重排进 _local_batch.input_ids)、

`restore_draft_prefill` (把 rank-local 的 `last_hidden_states` all-gather 回全局顺序, 必要时一并还原第二个 hidden states)。同时 `partition_batch` 去掉 `req_states` 依赖: `positions` 改为从 `global_batch.positions.index_select` (因为投机拒绝后 CPU 游标会滞后), `seq_lens` 由本地末位 `position + 1` 推导, `logits_indices` 直接取 `local_query_start_loc[1:] - 1` (本地 logits 不再被采样, 先整体还原 hidden states 再用全局 `InputBatch` 采样), 并删除 `combine_sampled_and_draft_tokens / prepare_pos_seq_lens` 调用与 `if input_batch.num_draft_tokens > 0: raise` 分支。

3. 自回归 MTP speculator 接线: `vllm/v1/worker/gpu/spec_decode/autoregressive/speculator.py` 的 `propose()` 在拷贝 `hidden_states` 前调用 `pcp_manager.prepare_draft_prefill()` 并用 `draft_prefill_batch` 覆盖 `num_tokens / num_tokens_padded`; 在 `num_speculative_steps == 1` 早退之后、`prepare_decode_inputs()` 之前调用 `block_tables.gather_block_tables(input_batch.idx_mapping, num_reqs_padded=num_reqs)`, 把 block table 刷新回全局顺序供 replicated draft 使用; `_run_model()` 通过 `get_draft_input_buffers()` 选择输入缓冲并把 `is_padding` 传入 `set_forward_context`; `_prefill()` 结尾调用 `restore_draft_prefill()` 还原 hidden states。
4. DSpark 保持 replicated: `vllm/v1/worker/gpu/spec_decode/dflash/speculator.py::__init__` 在 `prefill_context_parallel_size > 1` 时浅拷贝 `vllm_config` 并把 draft 侧 PCP size 置 1, 使 draft 自身不构建 PCP 机制; `propose()` 在 dummy-run 早退之后同样先 `gather_block_tables()` 再 `prepare_dflash_inputs()`。基类 `speculate/speculator.py` 新增 `pcp_manager` 字段、`draft_is_prefilling` 缓冲, 并把 `is_prefilling` 透传进 `_build_draft_attn_metadata()` 的 `build_attn_metadata()`。
5. 校验与 kernel 口径修正: `vllm/v1/worker/cp_utils.py::check_attention_cp_compatibility` 新增 `target_layer_names` 参数, 只对 target 层断言 `supports_pcp()` (draft 是复制副本, 可继续用 FLASH-ATTN 这类不支持 PCP 的后端); `vllm/v1/worker/gpu/model_runner.py` 在初始化时把 `self.pcp_manager` 注入 speculator, 并在 `sample_tokens()` 中保留 PCP 还原前的 `draft_hidden_states` 供 draft 使用、对 `aux_hidden_states` 逐个 `restore_hidden_states()`; `vllm/model_executor/layers/sparse_attn_indexer.py` 把 PCP 折算条件从 `if use_pcp:` 改为 `if use_pcp and num_tokens > k.shape[0]:`, 兼容 replicated draft 的未展开 slot mapping; `mtp/speculator.py` 在 `PCP > 1` 时关闭 `share_mtp_topk_indices`。
6. 测试配套: `tests/v1/worker/test_gpu_pcp_manager.py` 移除 `RequestState` 构造与依赖、给 global decode batch 补 `positions` (呼应第 2 步的 `positions` 口径变化); `tests/v1/spec_decode/test_eagle_draft_attn_metadata.py` 与 `tests/v1/worker/test_gpu_autoregressive_speculator.py` 为 fake speculator 补 `draft_is_prefilling` 字段。无新增 e2e 用例, 验证依赖作者手动的 4 卡 GLM-5.2-NVFP4 smoke (TP2 + PCP2 + DCP1 + EP4, PIECEWISE)。

关键文件:

- `vllm/v1/worker/gpu/pcp_manager.py` (模块 上下文并行; 类别 source; 类型 core-logic; 符号 `PCPManager.get_draft_input_buffers`, `PCPManager.prepare_draft_prefill`, `PCPManager.restore_draft_prefill`, `PCPManager.partition_batch`): 本 PR 的核心: 新增 `get_draft_input_buffers / prepare_draft_prefill / restore_draft_prefill` 三个 draft

prefill API, 放宽 validate_config 对 spec decode 的拒绝, 并重写 partition_batch 的 positions / seq_lens / logits_indices 口径、移除 req_states 依赖。

- vllm/v1/worker/gpu/spec_decode/autoregressive/speculator.py (模块 投机解码; 类别 source; 类型 core-logic; 符号 AutoregressiveSpeculator.propose, AutoregressiveSpeculator._run_model, AutoregressiveSpeculator._prefill) : MTP 投机解码主路径: propose() 接入 PCP 本地 batch 并在 replicated draft 前刷新 block table, _run_model() 切换输入缓冲并透传 is_padding, _prefill() 还原 hidden states。
- vllm/v1/worker/gpu/model_runner.py (模块 模型执行器; 类别 source; 类型 dependency-wiring; 符号 initialize_kv_cache, sample_tokens) : 把 PCPManager 注入 speculator、把 CP 兼容性校验限定到 target 层, 并在采样前保留 PCP-local 的 draft_hidden_states、还原 aux_hidden_states, 是 PCP 与 draft 数据流的接线点。
- vllm/v1/worker/gpu/spec_decode/dflash/speculator.py (模块 DFlash 投机; 类别 source; 类型 core-logic; 符号 DFlashSpeculator.init, DFlashSpeculator.propose) : DSpark 保持 replicated 的关键实现: 构造时浅拷贝 config 并把 draft 侧 PCP size 置 1, propose 中在 dummy-run 之后同样刷新全局 block table。
- vllm/v1/worker/gpu/spec_decode/speculator.py (模块 投机解码基类; 类别 source; 类型 dependency-wiring; 符号 DraftModelSpeculator.init, DraftModelSpeculator._build_draft_attn_metadata) : 基类新增 pcp_manager 字段与 draft_is_prefilling 缓冲, 并把 is_prefilling 透传进 build_attn_metadata, 让 draft 的 attention metadata 能区分 prefill/decode 行。
- vllm/v1/worker/cp_utils.py (模块 并行校验; 类别 source; 类型 core-logic; 符号 check_attention_cp_compatibility) : check_attention_cp_compatibility 新增 target_layer_names 参数, 只对 target attention 层断言 supports_pcp(), 是 DSpark 使用 FLASH_ATTN draft 后端的前提。
- vllm/model_executor/layers/sparse_attn_indexer.py (模块 稀疏注意力; 类别 source; 类型 data-contract; 符号 sparse_attn_indexer) : 把 PCP token 折算条件从 if use_pcp: 改为 if use_pcp and num_tokens > k.shape[0]:, 兼容 replicated MTP draft 的未展开 slot mapping, 是最容易漏掉的 kernel 侧口径修正。
- vllm/v1/worker/gpu/spec_decode/mtp/speculator.py (模块 MTP 投机; 类别 source; 类型 core-logic; 符号 load_draft_model) : MTP 侧的配套约束: PCP > 1 时关闭 share_mtp_topk_indices, 因为 token 被分片后无法复用 target 的 top-k index。
- tests/v1/worker/test_gpu_pcp_manager.py (模块 并行单测; 类别 test; 类型 test-coverage; 符号 test_partition_defers_dcp_metadata_to_post_partition_batch) : 唯一实质更新的单测: 移除 RequestState 依赖并给 global decode batch 补 positions, 直接对应 partition_batch 的口径变化。

关键符号: PCPManager.prepare_draft_prefill, PCPManager.restore_draft_prefill, PCPManager.get_draft_input_buffers, PCPManager.partition_batch, PCPManager.validate_config, AutoregressiveSpeculator.propose, AutoregressiveSpeculator._run_model, AutoregressiveSpeculator._prefill, DFlashSpeculator.propose, check_attention_cp_compatibility, sparse_attn_indexer, DraftModelSpeculator._build_draft_attn_metadata, load_draft_model

关键源码片段

vllm/v1/worker/gpu/pcp_manager.py

本 PR 的核心：新增 `get_draft_input_buffers` / `prepare_draft_prefill` / `restore_draft_prefill` 三个 draft prefill API，放宽 `validate_config` 对 spec decode 的拒绝，并重写 `partition_batch` 的 `positions/seq_lens/logits_indices` 口径、移除 `req_states` 依赖。

```
def get_draft_input_buffers(
    self, input_buffers: InputBuffers
) -> InputBatch | InputBuffers:
    # 首个 draft forward 复用 PCP 的 rank-local batch；之后回落到
    # speculator 自己的 InputBuffers，后续 draft step 不再走分片视角。
    return self.draft_prefill_batch or input_buffers

def prepare_draft_prefill(
    self,
    input_batch: InputBatch,
    input_ids: torch.Tensor,
) -> None:
    # 只对「当前这一步的 global batch」生效；dummy run 或已过期的 batch
    # 直接返回，避免把陈旧的 rank-local 状态泄漏给下一次 draft。
    if input_batch is not self._global_batch or self._local_batch is None:
        return
    local_batch = self._local_batch
    assert self._local_gather_idx is not None
    num_local_tokens = self._local_gather_idx.shape[0]
    # 复用 partition_batch() 阶段存下的 gather 索引，把全局顺序的
    # input_ids 重排成 rank-local 顺序写入本地输入缓冲。
    torch.index_select(
        input_ids,
        0,
        self._local_gather_idx,
        out=local_batch.input_ids[:num_local_tokens],
    )
    self.draft_prefill_batch = local_batch

def restore_draft_prefill(
    self,
    last_hidden_states: torch.Tensor,
    hidden_states: torch.Tensor,
) -> tuple[torch.Tensor, torch.Tensor]:
    if self.draft_prefill_batch is None:
        # 非 PCP draft prefill 路径：原样透传，调用方无需分支。
        return last_hidden_states, hidden_states
    local_last_hidden_states = last_hidden_states
    # 把 rank-local 的 draft 输出 all-gather 回全局顺序，供后续采样使用。
    last_hidden_states = self.restore_hidden_states(local_last_hidden_states)
    # 部分 MTP 模型会额外返回 feedback hidden；若它与前一个张量指向同一块
```

```

# 内存，则无需重复 all-gather ( 避免多余通信 )。
hidden_states = (
    last_hidden_states
    if local_last_hidden_states is hidden_states
    else self.restore_hidden_states(hidden_states)
)
# 一次性上下文，用完立即失效，防止后续 step 误用 rank-local batch。
self.draft_prefill_batch = None
return last_hidden_states, hidden_states

```

vllm/v1/worker/gpu/spec_decode/autoregressive/speculator.py

MTP 投机解码主路径: `propose()` 接入 PCP 本地 batch 并在 replicated draft 前刷新 block table, `_run_model()` 切换输入缓冲并透传 `is_padding`, `_prefill()` 还原 hidden states。

```

# ===== AutoregressiveSpeculator.propose() 中的 PCP 分支 =====
# 首次 draft forward 复用 target 的 rank-local batch: 把全局 input_ids
# 重排成本地顺序，并把 token 数切换为本地（含 padding）数量。
if self.pcp_manager is not None:
    self.pcp_manager.prepare_draft_prefill(
        input_batch,
        self.input_buffers.input_ids[:num_tokens_padded],
    )
    prefill = self.pcp_manager.draft_prefill_batch
    if prefill is not None:
        num_tokens = prefill.num_tokens
        num_tokens_padded = prefill.num_tokens_after_padding
    self.hidden_states[:num_tokens_padded].copy_(hidden_states)

# ... prefill forward 与采样省略 ...
if self.num_speculative_steps == 1:
    # 只投机 1 个 token: 首步产出即足够，直接返回。
    return self.draft_tokens[:num_reqs, :1]

# 后续 draft step 是 replicated 的（不再按上下文分片），必须先把
# block table 刷新回全局顺序，否则会读到 PCP 本地视角的 cache 位置，
# 这正是 review 中 GirasoleY 指出的 input_block_tables 未初始化问题。
if self.pcp_manager is not None and not dummy_run:
    self.block_tables.gather_block_tables(
        input_batch.idx_mapping, num_reqs_padded=num_reqs
    )

# ===== AutoregressiveSpeculator._run_model() 输入缓冲选择 =====
# 默认用 speculator 自己的 InputBuffers; PCP 的首次 draft prefill 会被
# 替换成 PCPManager 持有的 rank-local batch。
input_buffers: InputBatch | InputBuffers = self.input_buffers
is_padding = None
if self.pcp_manager is not None:
    input_buffers = self.pcp_manager.get_draft_input_buffers(self.input_buffers)
    # rank-local batch 尾部是补齐的 token，必须把 padding 掩码透传给

```

```

# forward context, 让 attention 后端按 padding 路径处理。
is_padding = input_buffers.is_padding[:num_tokens]
batch_descriptor = BatchDescriptor(num_tokens=num_tokens)
with set_forward_context(
    attn_metadata,
    self.vllm_config,
    num_tokens=num_tokens,
    num_tokens_across_dp=num_tokens_across_dp,
    slot_mapping=slot_mappings,
    batch_descriptor=batch_descriptor,
    is_padding=is_padding,
):
    model_inputs = dict(
        input_ids=input_buffers.input_ids[:num_tokens],
        positions=input_buffers.positions[:num_tokens],
        hidden_states=self.hidden_states[:num_tokens],
    )
    # PW 图或 eager 路径执行 draft forward; 输出经
    # pcp_manager.restore_draft_prefill() 还原为全局顺序。
    ...

```

vllm/model_executor/layers/sparse_attn_indexer.py

把 PCP token 折算条件从 `if use_pcp:` 改为 `if use_pcp and num_tokens > k.shape[0]:`, 兼容 replicated MTP draft 的未展开 slot mapping, 是最容易漏掉的 kernel 侧口径修正。

```

# slot_mapping 既可能来自 PCP 展开后的 target batch, 也可能来自
# replicated 的 MTP draft decode (本地未展开的 slot mapping),
# 所以不能只用 use_pcp 判断是否折算 token 数。
# 这里保留 PCP padding, 保证每个 rank 的 all-gather 形状一致。
num_tokens = slot_mapping.shape[0]
if k is not None:
    # 仅当 token 数大于本地 k 的形状 (说明它是被 PCP 展开过的) 时才按
    # world_size 折算; 否则 replicated draft 会被错误截短, 导致 KV
    # 写入位置错位或漏写。
    if use_pcp and num_tokens > k.shape[0]:
        num_tokens //= get_pcp_group().world_size
    k = k[:num_tokens]
if not skip_k_cache_insert:
    # 按 num_tokens 选出 top-k 位置并写入 KV cache。
    ...

```

评论区精华

1) 正确性问题被 reviewer 抓出 (已修复): GirasoleY 在 [vllm/v1/worker/gpu/pcp_manager.py](#) 第 578 行附近留言 “I just realize that we may need to update `input_block_tables` in mtp's attn metadata, otherwise mtp will be attending the wrong cache.”, LucasWilkinson 回复 “good catch! thanks! updated”。对应实现即两处 `block_tables.gather_block_tables(...)`, 以及 commit “Gather PCP block tables before

draft decode”。2) 实现风格的收敛获得认可：GirasoleY 评价 “Thank you for the simplification! The change looks good :) Left one accuracy comment inline.”，随后 APPROVED “lgtm!”，ywang96 亦 APPROVED 并合并。23 个 commit 中前 4 个由 GirasoleY 完成 (prefill-only MTP、decode/ 混合 batch、加固 buffer、复用本地 buffer)，后 15 个由 LucasWilkinson 以 “Minimize PCP draft integration” “Remove residual PCP draft plumbing” “Encapsulate PCP draft input selection” 等标题反复精简，说明该 PR 经历了明显的设计返工与收敛。3) 遗留验证疑虑 (未在 review 中闭环)：PR 正文写明 “These acceptance figures predate the block-table fix; full-model acceptance has not been rerun after it.”，即 66.7% (MTP2) 与 87.5% (DSpark8) 的接受率数据早于 block-table 修复，修复后未重跑整模型 acceptance。

- MTP draft 未刷新 input_block_tables，会访问错误的 KV cache (correctness): 新增 `block_tables.gather_block_tables(input_batch.idx_mapping, num_reqs_padded=num_reqs)` (autoregressive 与 dflash 的 `propose()` 各一处)，在首个 rank-local draft prefill 之后、后续 replicated draft decode 之前把 block table 刷新回全局顺序；对应 commit “Gather PCP block tables before draft decode”。
- 实现方案从“封装式 prefill context”收敛为 PCP 管理器直连 draft (design): Reviewer 明确认可该收敛: “Thank you for the simplification! The change looks good :)” 并 APPROVED; PR 正文亦自述是在 main 上合并 #53653 与 #53660 的 “smaller implementation”。
- block-table 修复后的整模型验证未闭环 (testing): Reviewer 与 maintainer 仍 APPROVED 并合并，52 个聚焦单测通过；整模型 acceptance 复测与 e2e 用例留待后续，属于已知的验证缺口。

风险与影响

- 风险：1) 正确性回归 (核心)：MTP 的 draft 在 `propose()` 中使用 `self.block_tables`，若 `gather_block_tables()` 的调用位置或 `num_reqs_padded` 口径与后续 `prepare_decode_inputs()` 不一致，仍可能读到 PCP 本地视角的 cache；该风险正是 review 抓出的问题类型，修复后未做整模型回归验证 (正文自述)。同时 `dummy_run` 被显式排除在 `gather` 之外，`profile` 路径与真实路径行为不一致，需靠后续 e2e 覆盖。2) 隐式形状判定：`sparse_attn_indexer.py` 用 `num_tokens > k.shape[0]` 判断是否需要按 PCP world size 折算 token 数，属于“靠形状猜语义”的启发式。一旦某个后端下 `k.shape[0]` 恰好不小于展开后的 token 数 (例如不同的 indexer 布局或 padding 组合)，就会漏折算或误折算，导致 KV 写入越界或漏写，且没有断言兜底。3) 配置浅拷贝 hack：`DFlashSpeculator.__init__` 对 `vllm_config` 做 `copy.copy` 后替换 `parallel_config.prefill_context_parallel_size=1`，任何共享子对象被下游修改都可能污染主配置或反过来被主配置影响；同时该校验“绕过”让 DSpark 的 PCP 兼容性只由 `PCPManager.validate_config` 中的 DCP 约束兜住。4) 校验放宽带来的组合爆炸：`check_attention_cp_compatibility(..., target_layer_names)` 只对 target 层断言 `supports_pcp()`，draft 层不再被检查；若某 drafter 实际需要 PCP 支持但被误判为 replicated，问题会推迟到运行期表现为错误结果而非启动报错。另外 sparse MLA + PCP 的 CUDA graph 限制被移除，属于超出 spec decode 范围的放开，回归面更大。5) 兼容性 / 性能面：只支持 PIECEWISE CUDA graph，full graph 仍报错；`_run_model` 新增

is_padding 传参要求 attention 后端正确消费 padding 掩码，draft 侧 buffer 从 InputBuffers 切换为 InputBatch（来自 PCP 本地 batch）时字段语义需完全对齐（num_tokens、num_tokens_after_padding、is_padding），否则会出现静默错算。

- 影响：用户侧：TP2 + PCP2（乃至 DCP/PCP 组合）+ MTP / DSpark 的 GLM-5.2-NVFP4 部署从“无法启动”变为可用，长上下文前置并行与投机解码可以叠加，直接提升该类配置的吞吐与长序列能力；受影响入口是 vLLM serve 的 --prefill-context-parallel-size 与 --speculative-config 组合。系统侧：MRv2 的 draft 路径新增 rank-local / replicated 两套视角切换，PCPManager 的职责从“只重写 target batch”扩展为“同时托管 draft prefill 上下文”，model_runner.sample_tokens 的 hidden states 流向（draft_hidden_states、aux_hidden_states 还原）随之改变，属于核心执行路径变更。团队侧：23 个 commit、两名作者、明显多轮重构，说明该组合的实现复杂度高，后续维护者需要理解 PCP 本地 / 全局视角切换的约束；tests/v1/worker/test_gpu_pcp_manager.py、tests/v1/spec_decode/test_eagle_draft_attn_metadata.py、tests/v1/worker/test_gpu_autoregressive_speculator.py 三个单测被同步改动，但缺少端到端回归用例。
- 风险标记：核心路径变更，准入校验放宽，缺少整模型回归验证，config 浅拷贝 hack，形状隐式判定，仅支持 PIECEWISE

关联脉络

- PR #53653 早期 PCP MTP 支持实现（PR 正文提及，标题未在上下文中给出）：本 PR 正文声明“consolidates #53653 and #53660 on current main with a smaller implementation”，即它是被本 PR 取代 / 合并的来源实现之一。
- PR #53660 早期 PCP MTP 支持实现（PR 正文提及，标题未在上下文中给出）：同上，与本 PR 属于同一条 PCP + MTP 功能线，本 PR 是其精简后的落地版本。
- PR #56145 [Core] MRV2 support for fast-prefill: 同样在 MRv2（vllm/v1/worker/gpu/model_runner.py、attn_utils.py、config/vllm.py）中打开 PCP 相关能力，并同样受“PIECEWISE CUDA graphs only”约束，与本 PR 共享 PCP 与图模式的演进脉络。
- PR #55819 [Perf] Use UVA-backed contents for MRV2 apply_write: 同属 MRv2 GPU worker 路径的 buffer/ 写入机制改造，与本 PR 在 draft/decoding 阶段对 positions 与 slot mapping 的 buffer 语义调整相互关联。
- PR #55239 [ROCm][Bugfix] Route GLM-5.3-Flash MTP through ragged sparse MLA: 同属 MTP 与 sparse MLA attention 分派的交叉领域；本 PR 改动了 sparse_attn_indexer 的 PCP token 折算与移除 sparse MLA + PCP 的 CUDA graph 限制，与该分派逻辑直接相关。