PR #50931 完整报告

vllm-project/vllm

[ModelRunner v2] Enable decoder token-wise pooling

合并时间：2026-08-07 06:21

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/50931

执行摘要

- 一句话：MRV2 放开 decoder 模型 token 级 pooling 任务
- 推荐动作：值得精读。虽然是 8 个文件的小改动，但它体现了两个可复用的设计决策：一是任务门控从保守过滤收敛为模型能力声明（get_supported_tasks 只依赖模型自身），二是推理引擎在异步调度下对张量所有权的处理（clone 条件精细到 chunk 与 async 开关）。MRV2 相关开发者应重点关注 AllPool.forward 的克隆策略。

功能与动机

PR body 明确提到 Advances #41286，目标是在 Model Runner V2 上启用 decoder token-wise pooling。池化本身已支持 chunked accumulation，但 MRV2 对非 encoder 模型过滤了 token_embed 与 token_classify；移除该过滤可解锁文本 decoder embedding、reward/process-reward 模型与 reranker。复现用例表明 main 上会抛出 ValueError: Model Runner V2 supports pooling tasks ['classify', 'embed'] ...。

实现拆解

1. 解锁任务门控：vllm/v1/worker/gpu/pool/pooling_runner.py 删除 _TOKEN_TASKS 与 _get_enabled_tasks，PoolingRunner.__init__ 改为直接以 _SUPPORTED_TASKS 校验模型选中的任务；get_supported_tasks 的签名从 (model, model_config) 简化为 (model)，调用方 vllm/v1/worker/gpu/model_runner.py 同步更新。这样 token 级任务不再因 attn_type == encoder_only 被过滤，decoder 型 embedding、reward/PRM 与 reranker 均可进入 MRV2。
2. 补齐输出所有权：vllm/model_executor/layers/pooler/tokwise/methods.py 的 AllPool 读取 scheduler_config.async_scheduling 得到 clone_finished。非 chunked 路径直接克隆输出；chunked 路径在缓存时按“未完成”或“async 且缓存为空”条件克隆，确保跨 step 保留或返回给 PoolerHead 的数据不依赖可能被复用的输入 buffer。
3. 测试配套：test_splade_sparse_pooler.py 将 decoder token 任务断言从 rejects/filters 改写为 supports/keeps；test_pooler_methods.py 扩展 _FakeSchedulerConfig 支持 async_scheduling 并新增所有权断言；test_all_pooling_plus_chunked_prefill.py 的 test_embed_models 改为 test_decoder_token_embed_model_runner_v2 并强制 VLLM_USE_V2_MODEL_RUNNER=1；test_reward.py 与 test_jina_reranker_v3.py 在非 CPU 平台启用 MRV2 并断言 use_v2_model_runner。

4. 配置与部署：无新增配置项；行为由 VLLM_USE_V2_MODEL_RUNNER（仍为试验开关）与 async_scheduling 联动，CPU 平台 reward 测试保留 MRV1 路径。

关键文件：

- vllm/v1/worker/gpu/pool/pooling_runner.py（模块 池化运行器；类别 source；类型 core-logic；符号 _get_enabled_tasks, get_supported_tasks）：核心变更文件：删除 _TOKEN_TASKS 过滤与 _get_enabled_tasks，开放 decoder 模型的 token 级 pooling 任务，并简化 get_supported_tasks 签名。
- vllm/model_executor/layers/pooler/tokwise/methods.py（模块 分词池化；类别 source；类型 data-contract；符号 AllPool, clone_finished）：AllPool 新增 clone_finished 逻辑，解决 async scheduling 下 hidden states buffer 复用导致输出悬空的问题，是正确性关键。
- vllm/v1/worker/gpu/model_runner.py（模块 模型运行器；类别 source；类型 data-contract；符号 get_supported_tasks）：get_supported_tasks 调用点适配新签名，属于接口联动。
- tests/models/language/pooling/test_splade_sparse_pooler.py（模块 稀疏池化；类别 test；类型 test-coverage；符号 test_pooling_runner_supports_decoder_token_classification, test_pooling_runner_keeps_decoder_token_classification, test_pooling_runner_keeps_decoder_token_embedding, test_pooling_runner_supports_embed_token_classification）：将 decoder token 任务的断言从拒绝 / 过滤改为支持 / 保留，直接验证门控放开。
- tests/model_executor/layers/test_pooler_methods.py（模块 池化方法；类别 test；类型 test-coverage；符号 _make_all_pool, test_chunked_prefill_single_shot_matches_non_chunked, test_non_chunked_owns_output_under_async_scheduling）：为 AllPool 增加 async scheduling 所有权语义测试，覆盖 clone 路径。
- tests/models/language/pooling/test_all_pooling_plus_chunked_prefill.py（模块 池化集成；类别 test；类型 test-coverage；符号 test_decoder_token_embed_model_runner_v2）：将原 MRV1 的 decoder token embedding 测试切换为 MRV2 并断言开关生效，是本 PR 的端到端复现用例。
- tests/models/language/pooling/test_reward.py（模块 奖励模型；类别 test；类型 test-coverage；符号 test_prm_models, test_prm_models_with_golden_outputs）：PRM/reward 模型切换到 MRV2 验证，CPU 平台保留 MRV1。
- tests/models/language/pooling/test_jina_reranker_v3.py（模块 重排序器；类别 test；类型 test-coverage；符号 test_offline）：reranker 集成测试启用 MRV2，验证 decoder reranker 场景。

关键符号：PoolingRunner.get_supported_tasks, PoolingRunner._get_enabled_tasks, AllPool.forward, AllPool.init, ModelRunner.get_supported_tasks

关键源码片段

vllm/v1/worker/gpu/pool/pooling_runner.py

核心变更文件：删除 _TOKEN_TASKS 过滤与 _get_enabled_tasks，开放 decoder 模型的 token 级 pooling 任务，并简化 get_supported_tasks 签名。

```
# vllm/v1/worker/gpu/pool/pooling_runner.py
# MRV2 的 pooling 任务全集；`embed&token_classify` 会把定长 embedding
# 与逐 token 权重拼接，输出长度随 prompt 增长，因此归入 token 级任务。
_SUPPORTED_TASKS: frozenset[PoolingTask] = frozenset(
    {'embed', 'classify', 'token_embed', 'token_classify', 'embed&token_classify'}
)


class PoolingRunner:
    def __init__(self, model: nn.Module, vllm_config: VllmConfig):
        self.model = cast(VllmModelForPooling, model)
        self.model_config = vllm_config.model_config
        self.max_num_reqs = vllm_config.scheduler_config.max_num_seqs
        model_tasks = tuple(sorted(self.model.pooler.get_supported_tasks()))
        selected_task = self.model_config.get_pooling_task(model_tasks)
        # 不再按 encoder/decoder 过滤：decoder 模型的 token 级任务同样开放，
        # 只要任务属于 MRV2 支持全集即可通过校验。
        if selected_task not in _SUPPORTED_TASKS:
            hint = (
                'Set an explicitly supported task or VLLM_USE_V2_MODEL_RUNNER=0.'
                if _SUPPORTED_TASKS.intersection(model_tasks)
                else 'Set VLLM_USE_V2_MODEL_RUNNER=0 to use this model.'
            )
            raise ValueError(
                'Model Runner V2 supports pooling tasks '
                f'{sorted(_SUPPORTED_TASKS)}, but this model selects '
                f'{selected_task!r} from {list(model_tasks)}. {hint}'
            )
        self.supported_tasks = frozenset(self.get_supported_tasks(model))
        if not self.supported_tasks:
            raise ValueError(
                'Model Runner V2 supports pooling tasks '
                f'{sorted(_SUPPORTED_TASKS)}, but this model supports '
                f'{list(model_tasks)}. '
                'Set VLLM_USE_V2_MODEL_RUNNER=0 to use this model.'
            )

    @staticmethod
    def get_supported_tasks(model: nn.Module) -> list[PoolingTask]:
        if not is_pooling_model(model):
            return []
        # 直接求模型任务与 MRV2 全集之交，任务集合由模型自身能力决定。
        return sorted(model.pooler.get_supported_tasks() & _SUPPORTED_TASKS)
```

vllm/model_executor/layers/pooler/tokwise/methods.py

AllPool 新增 clone_finished 逻辑，解决 async scheduling 下 hidden states buffer 复用导致输出悬空的问题，是正确性关键。

```
# vllm/model_executor/layers/pooler/tokwise/methods.py
```

```
class AllPool(TokenPoolingMethod):
    def __init__(self):
        super().__init__()
        vllm_config = get_current_vllm_config()
        scheduler_config = vllm_config.scheduler_config
        self.enable_chunked_prefill = scheduler_config.enable_chunked_prefill
        # async scheduling 下，下一步可能复用 hidden states 输入 buffer，
        # 输出在被拷贝到 CPU 之前必须拥有独立存储，因此需要克隆。
        self.clone_finished = bool(scheduler_config.async_scheduling)

    def forward(self, hidden_states: torch.Tensor,
                pooling_metadata: PoolingMetadata) -> list[TokenPoolingMethodOutputItem]:
        pooling_cursor = pooling_metadata.get_pooling_cursor()
        # 用已在 CPU 上的 num_scheduled_tokens 做 split，避免 GPU 到 CPU 同步。
        hidden_states_lst = list(
            torch.split(hidden_states,
                        pooling_cursor.num_scheduled_tokens_cpu.tolist())
        )

        # 非 chunked 路径：async scheduling 时直接克隆，防止返回悬空视图。
        if not self.enable_chunked_prefill:
            if self.clone_finished:
                return [hs.clone() for hs in hidden_states_lst]
            return hidden_states_lst

        pooling_states = pooling_metadata.pooling_states
        finished_mask = pooling_cursor.is_finished().tolist()

        # chunked 路径：先把当前 chunk 缓存进 pooling_states。
        # 未完成的 chunk 需跨 step 保留，必须克隆；
        # 已完成的 chunk 在 async scheduling 且缓存为空时也要克隆，
        # 否则缓存的是输入 buffer 视图，buffer 复用后内容会被覆盖。
        for p, hs_chunk, finished in zip(
            pooling_states, hidden_states_lst, finished_mask
        ):
            needs_owned_storage = not finished or (
                self.clone_finished and not p.hidden_states_cache
            )
            p.hidden_states_cache.append(
                hs_chunk.clone() if needs_owned_storage else hs_chunk
            )

        # prefill 全部完成时，把缓存交给 PoolerHead，未完成保持 None。
        output_list: list[TokenPoolingMethodOutputItem] = []
        for p, finished in zip(pooling_states, finished_mask):
            if finished:
                hidden_states_cache = p.hidden_states_cache
                if len(hidden_states_cache) == 1:
                    output_list.append(hidden_states_cache[0])
```

```
            else:
                output_list.append(torch.concat(hidden_states_cache, dim=0))
            p.clean()
        else:
            output_list.append(None)
    return output_list
```

评论区精华

核心讨论来自维护者 njhill。他在 approve 时说明补了一个 commit：I pushed an additional commit with some small changes related to cloning the tensors。动机是 async scheduling 下下一个 step 可能覆盖 hidden states 输入 buffer，输出拷贝到 CPU 之前必须持有独立存储。原测试语义从 rejects/filters decoder token 任务改为 supports/keeps，也直接体现了门控放宽。此外，Claude Code 自动评审因 fork PR 被禁用，未产生额外技术评论；19 条 issue 评论均为 /ci 触发与 Buildkite 状态。

- async scheduling 下张量所有权 (correctness): AllPool 根据 scheduler_config.async_scheduling 在需要时克隆已完成输出与跨 step 缓存 chunk。
- 移除 encoder-only 门控 (design): 删除 _TOKEN_TASKS 与 _get_enabled_tasks，token 级任务对所有模型开放。
- CPU 平台 reward 测试保留 MRV1 (testing): 非 CPU 平台启用 MRV2，CPU 保留 MRV1 路径。
- fork PR 自动评审禁用 (other): 未触发 Claude review；njhill 直接人工 approve。

风险与影响

- 风险：
  - 内存放大：AllPool 在 async scheduling 下会对已完成输出或跨 step 缓存做 clone，token_embed/token_classify 输出长度随 prompt 增长，长 prompt 场景下显存与 CPU 拷贝开销可能显著上升。
  - 行为回归：移除 encoder_only 过滤后，MRV2 下任何 decoder pooling 模型都可能选择 token 级任务；若模型池化实现并未实际支持 chunked 累积或语义不同，会产生错误结果。
  - 接口兼容性：get_supported_tasks 移除了 model_config 参数，仓库内部调用已更新，但依赖旧签名的第三方扩展会编译失败。
  - 覆盖空白：测试集中于 GPU、Qwen3-Embedding、PRM 与 Jina reranker，CPU 仍走 MRV1，PP/TP、cudagraph 与更多 decoder embedding 模型组合未覆盖。
- 影响：
  - 用户侧：decoder 文本 embedding、reward/process-reward、reranker 类模型可在 VLLM_USE_V2_MODEL_RUNNER=1 下运行，统一到 MRV2 执行路径。
  - 系统侧：pooling 任务校验模型从“按注意力类型过滤”简化为“模型能力与 MRV2 全集求交”；async scheduling 下输出所有权语义更安全，避免悬空视图。

- 团队侧：MRV2 对 pooling 模型的支持面扩大，向默认化目标前进一步；后续需补齐不同后端与模型族的回归。
- 风险标记：MRV2 任务门控放宽，async scheduling 输出克隆开销，decoder token 输出内存放大，测试覆盖集中于 GPU

关联脉络

- PR #50613 [Attention][MLA] Per-request scheduling for MLA chunked context: 同属 chunked prefill 执行路径的调度优化；本 PR 放开 decoder token 级 pooling 后，chunked accumulation 场景与其形成互补。
- PR #51113 [Bugfix] Keep mamba align prefill chunks block-aligned past last_cache_position: 同为 chunked prefill 边界正确性修复，与本 PR 的 chunk 累积输出正确性属同一关注面。