PR #47699 完整报告

vllm-project/vllm

[Frontend] Overlap preprocessing and computation for pooling models offline inference

合并时间：2026-07-16 22:00

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/47699

执行摘要

- 一句话：pooling 离线推理新增多线程预处理与 tiling 重叠调度
- 推荐动作：值得精读。重点看三点：① request_factory + render 的逐请求流式预处理抽象，如何把批处理拆成可并行单元交给线程池；② _run_tiling_engine 的窗口式调度（max_num_seqs * 2）与异常清理逻辑；③ 作者对 _render_and_run_requests 与 tiling 语义差异的分析，这解释了为什么需要做接口破坏性重构。同时可把 conversations 分支死代码、executor.map 无界提交作为后续跟踪项。

功能与动机

PR body 指出 pooling 离线推理当前不采用多线程预处理，且所有输入预处理完成后 GPU 才开始工作；更糟的是「preprocessing time currently grows with input length, even though the data will be truncated to a small prompt size afterward」——典型 embedding 场景（如对几十万 token 的 wiki 页预处理后截断到 512 token）耗时极长。关联 issue #45733 报告 LLM.embed 在 0.19 后出现回归，#41390 报告 Llama-Nemotron embedding 离线 batch-32 相对 Transformers 慢约 3 倍。作者希望通过多线程 + tiling 重叠尽可能逼近 HF 路径的性能，思想源自 #28458 与 #41678 的讨论。

实现拆解

1. 离线接口流式化：在 vllm/entrypoints/pooling/base/io_processor.py 中把原来的 pre_process_offline 拆成 get_request_factory_offline（返回 request_factory 生成器工厂与请求数）和 render（单条请求渲染，返回 PoolingEngineInput）。生成器每次 yield 一条 RenderParams，天然适配线程池逐条消费，这是 tiling 能够工作的前提。
2. 类型契约拆分：vllm/entrypoints/pooling/typing.py 把 OfflineInputsContext 拆为 OfflineEncodeInputsContext / OfflineScoringInputsContext / OfflinePluginInputsContext 三个具体上下文，并新增 RenderParams 系 TypedDict（EncodeCMPLRenderParams / EncodeChatRenderParams / ScoringRenderParams）与 PoolingEngineInput、RequestFactory、RequestGenerator 类型，使各任务的输入在类型层面分叉。
3. tiling 引擎实现重叠执行：vllm/entrypoints/pooling/offline.py 新增 _run_tiling_engine，复用 renderer 的线程池 self._executor.map(io_processor.render, request_factory()) 做多线程预处理；主循环以 max_requests_in_core = max_num_seqs * 2 为窗口，用 next(it) 逐个补充请求进引擎，每次循环调用一次 llm_engine.step() 消费完成输出，异常时

对已加入请求调用 abort_request 清理，最后按 request_id 排序还原输入顺序。

4. 各子类适配与入口调整：scoring/io_processor.py（CrossEncoder、LateInteraction、JinaRanking）、embed/io_processor.py（JinaForRanking）、pooling/io_processor.py（plugin）分别实现自己的 get_request_factory_offline / render；vllm/entrypoints/llm.py 中取消 pooling runner 下 renderer_num_workers 的无效警告（pooling 离线已真正使用该线程池）。
5. 测试与 CI 配套：新增 tests/entrypoints/pooling/basic/test_tiling_engine.py，覆盖基本路径、超多请求、PoolingParams 单 / 多 /None 及 step 抛异常时的 abort 清理；tests/models/language/pooling/test_multi_vector_retrieval.py 将 dtype 从 half 改为 float（作者称 half 本地不过且暂无时间排查）；.buildkite/test_areas/models_language.yaml 将 Extended Pooling 测试超时从 70 分钟提到 120 分钟并绑定 h200_35gb（近期 CI 超时）。

关键文件：

- vllm/entrypoints/pooling/scoring/io_processor.py（模块 评分预处理；类别 source；类型 core-logic；符号 pre_process_offline, get_request_factory_offline, _preprocess_cmpl_offline, request_factory）：CrossEncoder / LateInteraction / JinaRanking 等评分路径的核心改造：pre_process_offline 重构为 get_request_factory_offline + render，新增 token_type_ids 压缩与单条打分渲染逻辑，是最复杂的子类适配。
- vllm/entrypoints/pooling/base/io_processor.py（模块 基类处理；类别 source；类型 core-logic；符号 pre_process_offline, get_request_factory_offline, request_factory, render）：PoolingIOProcessor 基类的接口契约变更：新增 get_request_factory_offline、render、_lora_request_to_seq、_priority_to_seq，删除共享的离线批处理入口，是所有 pooling 子类的基座。
- vllm/entrypoints/pooling/typing.py（模块 类型契约；类别 source；类型 data-contract；符号 OfflineEncodeInputsContext, OfflineScoringInputsContext, OfflinePluginInputsContext, RenderParams）：离线上下文与渲染参数的类型契约全部在此定义：上下文按任务拆分为三类，RenderParams 系 TypedDict 与 PoolingEngineInput 成为流式预处理的数据载体。
- vllm/entrypoints/pooling/offline.py（模块 离线调度；类别 source；类型 core-logic；符号 _run_tiling_engine）：tiling 引擎 _run_tiling_engine 的核心实现：线程池并行渲染 + 窗口化请求注入 + 每轮 step 消费 + 异常 abort，是重叠预处理与计算的关键所在。
- vllm/entrypoints/pooling/pooling/io_processor.py（模块 插件接入；类别 source；类型 core-logic；符号 pre_process_offline, get_request_factory_offline）：plugin 任务的离线接入适配：把 plugin 扩展后的 prompts 与聚合后的 pooling_params 转交给基类工厂，保持插件语义不变。
- vllm/entrypoints/pooling/embed/io_processor.py（模块 嵌入适配；类别 source；类型 core-logic；符号 pre_process_offline, get_request_factory_offline）：JinaForRanking 模型适配新接口，校验至少两个输入后委托基类工厂。
- vllm/entrypoints/llm.py（模块 入口配置；类别 source；类型 configuration）：消除 pooling runner 下 renderer_num_workers 的误导性警告，因为这个参数现在对 pooling

离线推理真正生效。

- tests/entrypoints/pooling/basic/test_tiling_engine.py（模块 调度测试；类别 test；类型 test-coverage；符号 llm, test_tiling_engine_basic, test_tiling_engine_many_requests, test_tiling_engine_with_pooling_params）：新增 tiling 引擎专项测试：覆盖基本路径、窗口切分、PoolingParams 三种形态与 step 异常时的 abort 清理。
- tests/models/language/pooling/test_multi_vector_retrieval.py（模块 多向量测试；类别 test；类型 test-coverage）：将 dtype 从 half 改为 float，作者称 half 在本地不通过且暂无时间排查，但该改动削弱了 fp16 精度覆盖。
- .buildkite/test_areas/models_language.yaml（模块 CI 配置；类别 config；类型 configuration）：Extended Pooling CI 超时从 70 分钟放宽到 120 分钟并绑定 h200_35gb，属于为通过 CI 的配套调整。

关键符号：get_request_factory_offline, render, _run_tiling_engine, _lora_request_to_seq, _priority_to_seq, _preprocess_cmpl_offline, encode, score

关键源码片段

vllm/entrypoints/pooling/scoring/io_processor.py

CrossEncoder / LateInteraction / JinaRanking 等评分路径的核心改造：pre_process_offline 重构为 get_request_factory_offline + render，新增 token_type_ids 压缩与单条打分渲染逻辑，是最复杂的子类适配。

```
def render(
    self,
    render_params: EncodeCMPLRenderParams
    | EncodeChatRenderParams
    | ScoringRenderParams,
) -> PoolingEngineInput:
    # 仅接受 ScoringRenderParams，其余类型直接报错，保证类型安全
    if 'data_1' not in render_params:
        raise ValueError(
            f'Unsupported render_params type {render_params.__class__.__name__}'
        )
    render_params = cast(ScoringRenderParams, render_params)

    arrival_time = time.time()
    tok_params = render_params['tok_params']
    params = render_params['params']
    prompt_extras = render_params['prompt_extras']

    # 按 query / document 各自的 token 上限截断并拼接 score prompt
    max_tokens_per_query, max_tokens_per_doc = self._get_token_limits(
        pooling_params=params
    )
    _, engine_prompt = self.get_score_prompt(
        data_1=render_params['data_1'],
        data_2=render_params['data_2'],
```

```
        encode_kwargs=tok_params.get_encode_kwargs(),
        chat_template=render_params['chat_template'],
        max_tokens_per_query=max_tokens_per_query,
        max_tokens_per_doc=max_tokens_per_doc,
        chat_template_kwargs=prompt_extras.get('chat_template_kwargs')
        if prompt_extras else None,
    )

    # 必须先执行 post-tokenization（可能 pad / truncate）再压缩
    # token_type_ids，否则 query / document 边界会按旧序列计算
    tok_params.apply_post_tokenization(self.tokenizer, engine_prompt)
    if token_type_ids := engine_prompt.pop('token_type_ids', None):
        params = params.clone()
        compressed = compress_token_type_ids(token_type_ids)
        params.extra_kwargs = {'compressed_token_type_ids': compressed}

    engine_input = self.renderer.process_for_engine(engine_prompt, arrival_time)

    return PoolingEngineInput(
        prompts=engine_input,
        params=params,
        lora_requests=render_params['lora_requests'],
        priorities=render_params['priorities'],
    )
```

vllm/entrypoints/pooling/base/io_processor.py

PoolingIOProcessor 基类的接口契约变更：新增 get_request_factory_offline、render、_lora_request_to_seq、_priority_to_seq，删除共享的离线批处理入口，是所有 pooling 子类的基座。

```
  def get_request_factory_offline(
      self, ctx: ALLOfflineInputsContext
  ) -> tuple[RequestFactory, int]:
      assert isinstance(ctx, OfflineEncodeInputsContext)
      prompts_seq = prompt_to_seq(ctx.prompts)
      num_requests = len(prompts_seq)
      pooling_task = ctx.pooling_task

      # 在进入并行渲染前，先把 prompt 解析成模型输入单元，并展开
      # pooling_params / lora_request / priority 到与请求数等长的序列
      parsed_prompts = [
          (prompt if isinstance(prompt, bytes)
           else parse_model_prompt(self.model_config, prompt))
          for prompt in prompts_seq
      ]
      tok_params = self.renderer.default_cmpl_tok_params.with_kwargs(
          **(ctx.tokenization_kwargs or {})
      )
      pooling_params: PoolingParams | Sequence[PoolingParams]
```

```
    if ctx.pooling_params is None:
        pooling_params = PoolingParams()
    else:
        pooling_params = ctx.pooling_params
    params_seq = self._params_to_seq(pooling_params, num_requests)

    # 校验 task：plugin 允许覆盖 pooling_task，其余任务类型必须一致
    for param in params_seq:
        if param.task is None:
            param.task = pooling_task
        elif pooling_task == 'plugin':
            pass # 插件用自己的 io_processor.parse_request 校验输入
        elif param.task != pooling_task:
            msg = f'You cannot overwrite {param.task=!r} with {pooling_task=!r}!'
            raise ValueError(msg)

    seq_lora_requests = self._lora_request_to_seq(ctx.lora_request, num_requests)
    seq_priority = self._priority_to_seq(ctx.priorities, num_requests)

    # 返回「每调用一次就产出单个请求渲染参数」的生成器工厂，
    # 使线程池可以逐条消费，支撑 tiling 流水线
    def request_factory() -> RequestGenerator:
        for i in range(num_requests):
            yield EncodeCMPLRenderParams(
                prompts=parsed_prompts[i],
                tok_params=tok_params,
                prompt_extras=None,
                skip_mm_cache=False,
                params=params_seq[i],
                lora_requests=seq_lora_requests[i],
                priorities=seq_priority[i],
            )

    return request_factory, num_requests
```

vllm/entrypoints/pooling/offline.py

tiling 引擎 _run_tiling_engine 的核心实现：线程池并行渲染 + 窗口化请求注入 + 每轮 step 消费 + 异常 abort，是重叠预处理与计算的关键所在。

```
def _run_tiling_engine(
    self,
    io_processor: PoolingIOProcessor,
    request_factory: RequestFactory,
    num_requests: int,
    use_tqdm: bool | Callable[..., tqdm] = True,
):
    # 核心窗口：保持 max_num_seqs * 2 个请求在引擎中即可让 GPU 充分饱和，
    # 其余请求停留在预处理侧，形成「边预处理、边计算」的流水线。
    max_requests_in_core = (
```

```
    self.llm_engine.vllm_config.scheduler_config.max_num_seqs * 2
)
num_requests_in_core = 0
num_waited_requests = num_requests

if use_tqdm:
    tqdm_func = use_tqdm if callable(use_tqdm) else tqdm
    pbar = tqdm_func(total=num_requests, desc='Processed prompts',
                     dynamic_ncols=True)

outputs: list[PoolingRequestOutput] = []
added_request_ids: set[str] = set()

# 将 request_factory 产出的渲染参数逐条交给线程池预处理;
# 注意 ThreadPoolExecutor.map 会先同步提交全部任务，真正的
# 前瞻窗口约束由下方「按窗口补请求、按 step 消费输出」保证。
it = self._executor.map(io_processor.render, request_factory())

try:
    while num_waited_requests or self.llm_engine.has_unfinished_requests():
        requests = []
        # 只补足到窗口上限，避免一次性把整批输入塞进引擎
        for _ in range(max_requests_in_core - num_requests_in_core):
            if num_waited_requests == 0:
                break
            try:
                request = next(it)
                requests.append(request)
            except StopIteration:
                num_waited_requests = 0
                break
            num_waited_requests -= 1
            num_requests_in_core += 1

        if requests:
            request_ids = self._render_and_add_requests(
                prompts=[x['prompts'] for x in requests],
                params=[x['params'] for x in requests],
                lora_requests=[x['lora_requests'] for x in requests],
                priorities=[x['priorities'] for x in requests],
            )
            for request_id in request_ids:
                # 撤销 assign_request_id 追加的内部后缀，恢复外部 id 用于输出匹配
                request_id = request_id.split('-', 1)[0]
                added_request_ids.add(request_id)

        step_outputs = self.llm_engine.step()
        for output in step_outputs:
            assert isinstance(output, PoolingRequestOutput)
```

```
                assert output.finished # pooling 为单步任务，step 返回即结束
                outputs.append(output)
                added_request_ids.discard(output.request_id)
                num_requests_in_core -= 1
                if use_tqdm:
                    pbar.update(1)
        except Exception:
            # 出错时清空引擎内未完成请求，避免 KV cache 泄漏到下一次调用
            if added_request_ids:
                self.llm_engine.abort_request(list(added_request_ids))
            raise
        finally:
            if use_tqdm:
                pbar.close()

        # 引擎完成顺序与输入顺序不一致，按 request_id（纯数字）还原原始顺序
        return sorted(outputs, key=lambda x: int(x.request_id))
```

评论区精华

> DarkLight1337（issue 内追问）：OfflineInferenceMixin._render_and_run_requests 本来就通过 generator 重叠预处理与计算，pooling 为什么不适用？
>
> noooop：_render_and_run_requests 每次调用必须把所有 unfinished_request 跑完，不支持 tiling；而 run_tiling_engine 需要迭代 _render_and_add_requests 的全部输入（prompts、params、lora、priority），_render_and_run_requests 只迭代 prompts，因此必须整体重设计接口。
>
> claude[bot]：指出异常路径 abort_request 的外部 / 内部 id 语义问题（测试断言 internal=True 与实现不符）、CrossEncoder 的 token_type_ids 压缩时序问题、render() conversations 分支 engine_input[0] 会 KeyError，以及 ThreadPoolExecutor.map 会同步提交全部 render 任务削弱有界 windows 设计。
>
> noooop（对 map 问题）：贴出 _BoundedPreprocessor（deque + Future 的滑动窗口）草稿，但认为「A simple BoundedPreprocessor implementation will slow things down, so I won't let this small optimization block the PR — I'll improve it later」，留待后续。
>
> DarkLight1337（最终）：「Ok let's get this merged then, thanks for the explanations」

- abort_request 使用外部 id 与 internal 参数语义 (correctness): 最终实现不传 internal（等价 False），对外部 id 是正确的；测试已改为仅断言 request_ids 为非空 list。
- CrossEncoder 的 token_type_ids 压缩时序 (correctness): 最新 head 已调整为先 apply_post_tokenization 再 pop 并压缩，与在线路径语义一致。
- render 的 conversations 分支对 EngineInput 取 [0] 会 KeyError (correctness): 未修复，合入时作为已知隐患；将来接线 chat 离线流时必须修正。
- ThreadPoolExecutor.map 同步提交全部任务，削弱有界窗口 (performance): 作者认为简单实现反而变慢，决定后续再优化，不在本 PR 阻塞合入。

- tiling 是否应下沉到通用 _render_and_run_requests (design): 本轮限定 pooling 实现，生成模型路径（含 beam_search）留待未来重构复用。
- CI timeout / device 与 dtype 改动是否夹带 (other): 合入时接受；dtype 改动保留但未深入验证 fp16 精度覆盖损失。

风险与影响

- 风险:
  1. 破坏性接口变更：pre_process_offline 被移除，所有自定义 IOProcessor 插件与下游代码需迁移到 get_request_factory_offline + render，PR body 也为此道歉。
  2. 异常路径中止一致性：_run_tiling_engine 靠 request_id.split('-', 1)[0] 还原外部 id，且 abort_request 不传 internal 参数（等价 internal=False），依赖 assign_request_id 内部 id 生成规则的隐式约定；若引擎 id 规则变化，异常时 abort 可能静默失效，未完成请求泄漏在 core 中占用 KV cache。
  3. 内存峰值风险：ThreadPoolExecutor.map 会先同步提交全部 render 任务，大输入批时所有预处理结果在内存中累积，与「窗口外等待」的设计意图不符（作者已确认并计划后续修复）。
  4. 死代码隐患：base/io_processor.py 的 render() conversations 分支对 EngineInput dict 执行 engine_input[0] 会在未来 chat 离线流接入时直接 KeyError（当前无代码产出 EncodeChatRenderParams）。
  5. 测试覆盖削弱：test_multi_vector_retrieval.py 从 half 改 float，丢失 fp16 精度一致性验证；CI timeout 放宽可能掩盖慢速回归。 - 影响：对用户：pooling 离线推理（embed / classify / score / plugin）在长输入、多请求场景吞吐显著提升，13 万 token 输入约 8.5 倍加速，renderer_num_workers 参数在 pooling runner 下首次真正生效。对系统：仅在入口层新增 tiling 调度循环，在线 serve 路径完全不变，kernel / worker 无改动。对团队：PoolingIOProcessor 拆成 online（原流程）与 offline（流式化）两套语义，维护成本上升；作者已计划把 tiling 优化推广到生成模型与 beam_search 路径。整体影响面集中、可控，属于中高重要度的前端功能演进。 - 风险标记：接口破坏性变更，异常路径中止一致性，内存峰值风险，死代码分支隐患，测试覆盖削弱

关联脉络

- 暂无明显关联 PR