

PR #54364 完整报告

vllm-project/vllm

[Bugfix][Frontend] Truncate pooling prompts before padding them

合并时间: 2026-08-31 13:12

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54364>

执行摘要

- 一句话: 修复 pooling 先补位后左截断导致 prompt 全为 pad
- 推荐动作: 值得精读。这是典型的 " 参数处理顺序即语义 " 型 bug: 单看每个校验器都正确, 组合顺序错误却产生静默的纯 pad embedding。值得借鉴的点: 1) 用 Transformers 的 pad_truncation 语义作为行为基准; 2) 测试用 sentinel id 而非抛异常断言来证明退化值到达消费者; 3) 对主修复立即追问 " 是否存在同构的平行逻辑 ", claude[bot] 的补充发现与作者的两个提交展示了如何系统性收口。

功能与动机

PR body 指出 `TokenizeParams._validate_tokens` 的校验器执行顺序与参数命名的 Transformers 流水线相反: `padding` 文档声称与 Transformers tokenizer 同名, 但 Transformers 是先截断再补齐 (`pad_truncation`)。 `PoolingBasicRequestMixin` 在同一请求模型上暴露 `truncate_prompt_tokens`、`padding` 与 `truncation_side`, 且自 #51157 起 `padding` 可经 HTTP 设置, 一个请求即可组合触发: tokenizer 返回约 3 个 id → `_token_padding` 补到 512 → 左侧截断保留最后 8 个 token → 8 个全是 pad token → `_token_len_check` 通过 → 200 OK 返回纯 padding 的 embedding。更隐蔽的是 `truncation_side` 不必显式发送, `_token_truncation` 会回退到 `tokenizer.truncation_side`, 任何默认左侧截断的 tokenizer 都会命中; 而右侧截断时结果恰好与 Transformers 一致, 导致 bug 很容易被忽略。

实现拆解

1. 根因定位 (`vllm/renderers/params.py`): `TokenizeParams._validate_tokens` 通过 validator 元组依次执行 `_token_padding` → `_token_truncation` → `_token_len_check`。错误发生在 padding 先于 truncation: 补位让序列达到 `max_input_tokens`, 左侧截断 " 保留最后 N 个 " 恰恰选中刚追加的 pad token。 `apply_post_tokenization` 是 pooling、embedding、scoring 三条路径的统一后处理入口, 因此这里是全局修复点。
2. 主修复: 将 `_token_truncation` 调整到 `_token_padding` 之前, 并保留 `_token_len_check` 最后执行。安全性有两点保证: `_token_padding` 最多补齐到 `max_input_tokens`, 而 `__post_init__` 已拒绝 `truncate_prompt_tokens > max_input_tokens`, 所以重排不会让序列越过长度上限。
3. 联动修复 (跟随提交 `ac5d07e1a`, `vllm/entrypoints/pooling/scoring/io_processor.py`): `_apply_post_tokenization_to_token_type_ids` 为 cross-encoder scoring 请求维护与

prompt_token_ids 平行的 token_type_ids 数组，原实现同样先补位后截断。claude[bot] 在 review 中指出主修复后两个数组按不同顺序处理，compress_token_type_ids 压缩出的 query/document 段边界与实际 token 位置错位。该函数被重排为 truncate-first，与 _validate_tokens 保持一致。

4. 测试配套：tests/renderers/test_completions.py 新增

test_padding_with_left_truncation_keeps_the_prompt，核心哨兵是 pad_token_id = 99999（故意落在 encode 返回范围外），用 set(token_ids) != {pad_id} 证明 "退化值到达了消费者" 而非断言抛异常（该场景本就无人抛异常）；同时新增

test_padding_without_truncation_is_unchanged 固定 padding-only 行为。

tests/entrypoints/pooling/scoring/test_io_processor_unit.py 为新增文件，断言长度一致与段边界值 10——其中边界断言是承重墙，单独的长度断言在 main 上也能通过（main 上两个数组以同样方式出错）。

5. CI 排障：Buildkite 86210 唯一失败是 test_openai_schema.py::test_openapi_stateless，由 schemathesis 生成的 bad_words 含 U+0085（NEL，对 str.lstrip() 是空白）触发 500；作者论证 chat completion 请求路径上 padding/truncation 均为恒等操作、顺序不可观察，与本 PR 无关。

关键文件：

- vllm/renderers/params.py（模块 参数校验；类别 source；类型 core-logic；符号 _validate_tokens, apply_post_tokenization）：根因所在：
TokenizeParams._validate_tokens 的 validator 顺序从 padding → truncation → len_check 调整为 truncation → padding → len_check，是本 PR 的核心修复点，影响所有 pooling/embedding 请求的 token 后处理。
- vllm/entrypoints/pooling/scoring/io_processor.py（模块 评分处理；类别 source；类型 core-logic；符号 _apply_post_tokenization_to_token_type_ids）：claude[bot] 指出的联动缺陷所在：_apply_post_tokenization_to_token_type_ids 原先按同样错误的 pad-first 顺序处理与 prompt_token_ids 平行的 token_type_ids，主修复后两数组顺序分叉，cross-encoder 段边界错位；跟随提交将其重排为 truncate-first。
- tests/renderers/test_completions.py（模块 渲染器测试；类别 test；类型 test-coverage；符号 test_padding_with_left_truncation_keeps_the_prompt, test_padding_without_truncation_is_unchanged）：新增
test_padding_with_left_truncation_keeps_the_prompt（回归哨兵，断言 prompt 不全为 pad token 且 Transformers 语义正确）与 test_padding_without_truncation_is_unchanged（固定 padding-only 行为），并给 DummyTokenizer 增加 pad_token_id = 99999 哨兵。
- tests/entrypoints/pooling/scoring/test_io_processor_unit.py（模块 评分测试；类别 test；类型 test-coverage；符号 _DummyTokenizer, test_token_type_ids_stay_aligned_with_a_truncated_padded_prompt）：新增文件，首个测试
test_token_type_ids_stay_aligned_with_a_truncated_padded_prompt 用 _DummyTokenizer 固定长度与段边界双重断言，边界断言是承重墙——单独长度断言在 main 上也能通过。

关键符号: `TokenizerParams._validate_tokens`, `TokenizerParams.apply_post_tokenization`, `_apply_post_tokenization_to_token_type_ids`, `test_padding_with_left_truncation_keeps_the_prompt`, `test_padding_without_truncation_is_unchanged`, `test_token_type_ids_stay_aligned_with_a_truncated_padded_prompt`

关键源码片段

vllm/renderers/params.py

根因所在: `TokenizerParams._validate_tokens` 的 validator 顺序从 padding → truncation → len_check 调整为 truncation → padding → len_check, 是本 PR 的核心修复点, 影响所有 pooling/embedding 请求的 token 后处理。

```
def _validate_tokens(self, tokenizer: TokenizerLike | None, tokens: _S) -> _S:
    """Apply all validators to a token sequence."""
    # 先截断、后补齐, 与这些参数所命名的 Transformers 流水线保持一致。
    # 若先补齐, 随后的左侧截断只会保留刚追加的 pad token,
    # 真实内容被全部丢弃, 而请求依然能成功返回 200。
    for validator in (
        self._token_truncation,
        self._token_padding,
        self._token_len_check,
    ):
        tokens = validator(tokenizer, tokens)

    return tokens


def apply_post_tokenization(
    self,
    tokenizer: TokenizerLike | None,
    prompt: TokensPrompt | EmbedsPrompt,
) -> TokensPrompt | EmbedsPrompt:
    """Ensure that the prompt meets the requirements set out by this config.

    If that is not possible, raise a `VLLMValidationError`.
    """
    # pooling、embedding、scoring 三条前端路径都会经由本方法进入
    # `_validate_tokens`, 因此这里的顺序调整是统一的 token 后处理行为变更。
    if "prompt_token_ids" in prompt:
        prompt["prompt_token_ids"] = self._validate_tokens( # type: ignore[typeddict-unknown-key]
            tokenizer,
            prompt["prompt_token_ids"], # type: ignore[typeddict-item]
        )
    if "prompt_embeds" in prompt:
        prompt["prompt_embeds"] = self._validate_tokens( # type: ignore[typeddict-unknown-key]
            tokenizer,
            prompt["prompt_embeds"], # type: ignore[typeddict-item]
        )
```

```
return prompt
```

vllm/entrypoints/pooling/scoring/io_processor.py

claude[bot] 指出的联动缺陷所在：_apply_post_tokenization_to_token_type_ids 原先按同样错误的 pad-first 顺序处理与 prompt_token_ids 平行的 token_type_ids，主修复后两数组顺序分叉，cross-encoder 段边界错位；跟随提交将其重排为 truncate-first。

```
def _apply_post_tokenization_to_token_type_ids(
    tokenizer: Any,
    tok_params: TokenizeParams,
    token_type_ids: list[int],
) -> list[int]:
    # 顺序必须与 `TokenizeParams._validate_tokens` 完全一致：先截断、后补齐。
    # `token_type_ids` 与 `prompt_token_ids` 是平行数组，用于定位
    # cross-encoder 的 query/document 段边界；两个数组若按不同顺序处理，
    # 压缩后的边界索引会静默错位，模型会误以为 query 段为空。
    max_length = tok_params.truncate_prompt_tokens
    if max_length is not None and max_length < 0:
        max_length = tok_params.max_input_tokens

    if max_length is not None and max_length < len(token_type_ids):
        if max_length == 0:
            token_type_ids = token_type_ids[:0]
        else:
            side = tok_params.truncation_side or (
                tokenizer.truncation_side if tokenizer is not None else None
            )
            token_type_ids = (
                token_type_ids[-max_length:]
                if side == "left"
                else token_type_ids[:max_length]
            )

    pad_length = tok_params.pad_prompt_tokens
    if pad_length is not None and pad_length < 0:
        pad_length = tok_params.max_input_tokens

    if pad_length is not None and pad_length > len(token_type_ids):
        # 沿用数组最后一个 token type id 作为补齐值：query 段为 0、document 段为 1。
        pad_token_type_id = token_type_ids[-1] if token_type_ids else 0
        token_type_ids = token_type_ids + [pad_token_type_id] * (
            pad_length - len(token_type_ids)
        )

    return token_type_ids
```

tests/entrypoints/pooling/scoring/test_io_processor_unit.py

新增文件，首个测试 `test_token_type_ids_stay_aligned_with_a_truncated_padded_prompt` 用 `_DummyTokenizer` 固定长度与段边界双重断言，边界断言是承重墙——单独长度断言在 `main` 上也能通过。

```
def test_token_type_ids_stay_aligned_with_a_truncated_padded_prompt():
    """cross-encoder 段边界必须能穿过 truncate + pad 保持正确。"""
    tokenizer = _DummyTokenizer()
    num_query, num_doc = 20, 30
    prompt = TokensPrompt(prompt_token_ids=list(range(num_query + num_doc)))
    token_type_ids = [0] * num_query + [1] * num_doc

    tok_params = TokenizeParams(
        max_total_tokens=100,
        pad_prompt_tokens=-1,
        truncate_prompt_tokens=40,
        truncation_side="left",
    )

    # 两条路径使用同一份 `tok_params` 与 tokenizer,
    # 任何一方截断 / 补齐顺序不一致都会让下面的断言失败。
    prompt = tok_params.apply_post_tokenization(tokenizer, prompt)
    token_type_ids = _apply_post_tokenization_to_token_type_ids(
        tokenizer, tok_params, token_type_ids
    )

    prompt_token_ids = prompt["prompt_token_ids"]
    assert len(token_type_ids) == len(prompt_token_ids)

    # 左侧截断到 40 会丢掉前 10 个 query token, 剩余 10 个 query token,
    # document 段从索引 10 开始; main 上该值为 0, 等于告诉模型 query 段为空。
    first_doc = compress_token_type_ids(token_type_ids)
    assert first_doc == 10
    assert prompt_token_ids[:first_doc] == list(range(10, num_query))
    assert prompt_token_ids[first_doc:40] == list(range(num_query, 50))
```

评论区精华

Review 中最有价值的交锋来自 `claude[bot]` 的补充发现：主修复重排了 `params.py` 的 `_validate_tokens`，但 `vllm/entrypoints/pooling/scoring/io_processor.py` 中针对平行数组 `token_type_ids` 的重复 `pad/truncate` 逻辑仍保持先补位后截断。`claude[bot]` 指出：此前 `prompt_token_ids` 与 `token_type_ids` 以同样方式出错、保持对齐，现在两者静默分叉，`compress_token_type_ids` 传给 cross-encoder 的段边界将与实际 token 位置错位。Hotragn 确认后提交 `ac5d07e1a` 重排该函数，并用具体数字论证：20 个 query token + 30 个 document token、补齐到 100、左侧截断到 40 时，正确边界应为 10（丢弃前 10 个 query token），而 `main` 上边界为 0（query 段被判空）。DarkLight1337 询问新测试是否能在旧代码上失败，Hotragn 给出两种失败模式：对第一提交 `48d3b41a2` 失败于长度断言 40 vs 100，对 `main` 失败于边界断言 0 vs 10，证明测试分别守卫不同层级的回归。

- `io_processor.py` 中 `token_type_ids` 路径仍是 `pad-first`, 与 `prompt_token_ids` 顺序不一致 (correctness): Hotragn 确认发现正确, 提交 `ac5d07e1a` 将该函数重排为先截断后补齐, 并新增单元测试固定长度与边界 (边界 10 vs main 的 0)。
- 新测试是否能在旧代码上失败 (testing): 测试被验证有效且能分别守卫不同层级的回归, DarkLight1337 认可 (Thanks for checking)。
- CI 唯一失败与本 diff 无关 (question): 作者完成排障并说明该失败来自既有测试数据边界; PR 最终获批。

风险与影响

- 风险:
 1. 通用路径变更: `_validate_tokens` 是 pooling/embedding 所有 token 后处理的统一入口, 顺序调整影响整个前端; 不过仅 padding + left truncation 组合的行为发生变化, 右侧截断与纯 padding/ 纯截断均不受影响 (有测试固定)。
 2. 平行数组一致性成为隐性约束: `params.py` 与 `io_processor.py` 两处顺序逻辑分处两文件, 虽加注释提示, 但缺少机制性保障, 未来改动可能再次分叉。
 3. `_token_padding` 无保护使用 `tokenizer.pad_token_id`, 若某 tokenizer 的 `pad_token_id` 为 None, 会在 truncate 之后把 None 拼进 `prompt_token_ids`; PR 作者明确未处理该路径, 属于已知缺口。
 4. CI 中 `test_openapi_stateless` 的 500 由 schemathesis 生成的 `bad_words` 含 `U+0085` 触发, 与本 PR 无关, 但暴露了 `sampling_params.py` 对坏词空串化的边界用例缺口。
 - 影响: 用户侧: 使用 `/v1/embeddings` 或 `scoring` API 且同时开启 padding 与左侧截断的用户, 将从静默拿到纯 pad embedding 变为获得正确内容 + 补位结果; 右侧截断与不同时使用两者的人群无感知。系统侧: `scoring/rerank` 请求的 `cross-encoder query/document` 段边界计算依赖两个平行数组同步处理, 直接影响排序质量。团队侧: 确立了前端 token 后处理 " 先截断后补齐 " 的语义约定, 并沉淀了可复用的 sentinel 测试模式 (`pad_token_id` 故意落在 `encode` 范围之外, 用 `set(token_ids) != {pad_id}` 证明退化值到达消费者)。
 - 风险标记: 核心路径变更, 平行数组一致性约束, `pad_token_id` 缺失未防护, 行为语义对齐外部约定

关联脉络

- PR #54407 [Bugfix][Frontend] Truncate `prompt_token_offsets` with the prompt: 同属 `renderer` 前端参数后处理一致性主题, 同样修复平行数组未随截断同步的问题, 体现 " 截断必须同步所有派生并行结构 " 的维护原则。
- PR #48584 [Rust Frontend] Add support for `truncate_prompt_tokens` and `truncation_side`: 为 Rust 前端引入同一组截断参数, 与本 PR 处理的 `truncate_prompt_tokens` / `truncation_side` 参数语义同源, 均在收敛截断行为。