

PR #41800 完整报告

vllm-project/vllm

[Bugfix] Account for truncate_prompt_tokens when computing max_tokens

合并时间: 2026-05-07 00:10

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/41800>

执行摘要

- 一句话: 修复 truncate_prompt_tokens 导致 max_tokens 预算低估
- 推荐动作: 值得精读, 尤其关注如何将分散的重复逻辑收拢到核心公共函数中。测试用例对三种场景的覆盖设计清晰, 可作为类似边界测试的参考。

功能与动机

PR 描述指出: "When a request sets truncate_prompt_tokens, the prompt is truncated before inference but get_max_tokens() was receiving the original (untruncated) prompt length. This caused the output token budget to be underestimated by (original_len - truncated_len) tokens."

实现拆解

修复分三步完成:

1. 核心函数增强: 在 vllm/entrypoints/utils.py 的 get_max_tokens() 中增加可选参数 truncate_prompt_tokens, 在计算 model_max_tokens 之前对 input_length 进行缩窄处理: 若参数为 -1 则截断至 max_model_len, 否则截断至指定值, 然后才执行原有逻辑。
2. 调用点适配: 在所有调用 get_max_tokens() 的 serving 入口 (chat_completion/serving.py、completion/serving.py、responses/serving.py 两处、serve/render/serving.py 两处) 传递对应的 truncate_prompt_tokens 参数, 消除各入口的重复逻辑。
3. 回归测试: 在 tests/entrypoints/openai/chat_completion/test_serving_chat.py 中新增 test_serving_chat_truncate_prompt_tokens_max_token_accounting 测试, 覆盖无截断、截断值为 5、截断值为 -1 三种场景, 断言最终 max_tokens 正确。

关键文件:

- vllm/entrypoints/utils.py (模块 公共函数; 类别 source; 类型 core-logic; 符号 get_max_tokens): 核心修改: 在 get_max_tokens 函数中新增 truncate_prompt_tokens 参数, 内部根据该参数缩窄 input_length, 是修复的核心逻辑所在。
- tests/entrypoints/openai/chat_completion/test_serving_chat.py (模块 服务测试; 类别 test; 类型 test-coverage; 符号 test_serving_chat_truncate_prompt_tokens_max_token_accounting): 新增回归测试, 覆盖三种场景, 确保修复正确且不退化已有行为。

- `vllm/entrypoints/openai/responses/serving.py` (模块 响应服务; 类别 source; 类型 core-logic; 符号 `create_responses`, `_generate_with_built_in_tools`) : Responses API 的两处 `get_max_tokens` 调用都增加了 `truncate_prompt_tokens` 参数, 确保该入口也受益于修复。
- `vllm/entrypoints/serve/render/serving.py` (模块 渲染服务; 类别 source; 类型 core-logic; 符号 `render_chat_request`, `render_completion_request`) : Render serving 的两处 `get_max_tokens` 调用增加了 `truncate_prompt_tokens` 参数, 覆盖渲染端点。
- `vllm/entrypoints/openai/chat_completion/serving.py` (模块 聊天服务; 类别 source; 类型 core-logic; 符号 `create_chat_completion`) : Chat completion 入口调用 `get_max_tokens` 时增加 `truncate_prompt_tokens` 参数, 是该修复的主要应用场景之一。
- `vllm/entrypoints/openai/completion/serving.py` (模块 补全服务; 类别 source; 类型 core-logic; 符号 `create_completion`) : Completion 入口调用 `get_max_tokens` 时增加 `truncate_prompt_tokens` 参数, 是该修复的主要应用场景之一。

关键符号: `get_max_tokens`, `test_serving_chat_truncate_prompt_tokens_max_token_accounting`, `create_chat_completion`, `create_completion`, `create_responses`, `_generate_with_built_in_tools`, `render_chat_request`, `render_completion_request`

关键源码片段

`vllm/entrypoints/utils.py`

核心修改: 在 `get_max_tokens` 函数中新增 `truncate_prompt_tokens` 参数, 内部根据该参数缩窄 `input_length`, 是修复的核心逻辑所在。

```
def get_max_tokens(
    max_model_len: int,
    max_tokens: int | None,
    input_length: int,
    default_sampling_params: dict,
    override_max_tokens: int | None = None,
    truncate_prompt_tokens: int | None = None, # 新增参数
) -> int:
    # 若设置了 truncate_prompt_tokens, 则按规则缩窄 input_length
    if truncate_prompt_tokens is not None:
        limit = truncate_prompt_tokens
        # 值为 -1 表示使用 max_model_len 作为上限, 否则使用指定值
        input_length = min(
            input_length,
            max_model_len if limit == -1 else limit,
        )
    # 后续逻辑不变
    if max_model_len < input_length:
        raise ValueError(...)
    model_max_tokens = max_model_len - input_length
    platform_max_tokens = current_platform.get_max_output_tokens(input_length)
    fallback_max_tokens = (
        max_tokens
```

```

        if max_tokens is not None
        else default_sampling_params.get("max_tokens")
    )
    return min(
        val
        for val in (
            model_max_tokens,
            fallback_max_tokens,
            override_max_tokens,
            platform_max_tokens,
        )
        if val is not None
    )

```

tests/entrypoints/openai/chat_completion/test_serving_chat.py

新增回归测试，覆盖三种场景，确保修复正确且不退化已有行为。

```

@pytest.mark.asyncio
async def test_serving_chat_truncate_prompt_tokens_max_token_accounting():
    """当 truncate_prompt_tokens 设置后，max_tokens 必须使用截断后的
    prompt 长度计算，而非原始 prompt 长度。
    """
    mock_engine = MagicMock(spec=AsyncLLM)
    mock_engine.errorred = False
    mock_engine.model_config = MockModelConfig()
    mock_engine.input_processor = MagicMock()
    mock_engine.renderer = _build_renderer(mock_engine.model_config)
    serving_chat = _build_serving_chat(mock_engine)

    messages = [{"role": "user", "content": "what is 1+1?"}] # 7 tokens

    # Baseline: 无 truncation, max_tokens = 100 - 7 = 93
    req = ChatCompletionRequest(model=MODEL_NAME, messages=messages)
    with suppress(Exception):
        await serving_chat.create_chat_completion(req)
    assert mock_engine.generate.call_args.args[1].max_tokens == 93

    # truncate_prompt_tokens=5 时，有效 prompt 长度为 5,
    # max_tokens = 100 - 5 = 95
    req = ChatCompletionRequest(
        model=MODEL_NAME,
        messages=messages,
        truncate_prompt_tokens=5,
    )
    with suppress(Exception):
        await serving_chat.create_chat_completion(req)
    assert mock_engine.generate.call_args.args[1].max_tokens == 95

    # truncate_prompt_tokens=-1 表示用 max_model_len 做上限,

```

```
# 实际 prompt 只有 7，所以仍为 100 - 7 = 93
req = ChatCompletionRequest(
    model=MODEL_NAME,
    messages=messages,
    truncate_prompt_tokens=-1,
)
with suppress(Exception):
    await serving_chat.create_chat_completion(req)
assert mock_engine.generate.call_args.args[1].max_tokens == 93
```

评论区精华

Review 中 `gemini-code-assist[bot]` 指出 `chat_completion/serving.py` 和 `completion/serving.py` 两处存在相同的截断逻辑重复，建议集中到共享工具类。
`DarkLight1337` 回应 "Perhaps we should factor this out into `get_max_tokens` function", 并提醒 "`get_max_tokens` is called in a couple other places so you should fix those as well"。作者采纳建议，将逻辑折叠进 `get_max_tokens`，并在第三轮提交中修复了 Responses API 等其他调用点。最终实现所有入口统一传入参数，没有重复代码。

- 逻辑重复与集中化建议 (design): 作者采纳建议，在第二轮提交中将重复逻辑折叠进 `get_max_tokens`，第三轮提交修复了 Responses API 等其他调用点。
- Responses API 的适配 (correctness): 作者在第三轮提交中为 `responses/serving.py` 的两处调用添加了 `truncate_prompt_tokens` 参数。

风险与影响

- 风险：回归风险：`get_max_tokens` 是多个入口共用的公共函数，新增参数默认为 `None`，对未设置 `truncate_prompt_tokens` 的请求行为不变，回归风险低。遗漏风险：若未来新增调用 `get_max_tokens` 的入口，可能忘记传入 `truncate_prompt_tokens` 参数，但参数默认为 `None` 可保证向后兼容。性能风险：仅新增一次 `min` 计算和条件分支，无性能影响。兼容风险：函数签名向后兼容，外部调用无需修改。
- 影响：用户影响：设置 `truncate_prompt_tokens` 的请求现在能获得正确的输出 token 预算，不再因预算低估而被异常截断或报错。系统影响：所有聊天、补全、Responses API 和渲染端点均受益，覆盖范围完整。团队影响：统一逻辑到 `get_max_tokens` 降低了未来维护成本和逻辑分歧风险。
- 风险标记：核心函数变更，外部 API 兼容性

关联脉络

- 暂无明显关联 PR