

PR #41181 完整报告

vllm-project/vllm

[Bugfix] Fix `RuntimeError: Already borrowed` by adding thread-safe Hugging Face fast-tokenizer wrappers

合并时间: 2026-05-05 22:04

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/41181>

执行摘要

- 一句话: 修复 HF fast-tokenizer 并发 Already borrowed 错误
- 推荐动作: 此 PR 是解决核心并发问题的关键修复, 设计思路 (队列 + 动态子类化) 值得学习。建议仔细阅读 vllm/tokenizers/hf.py 中 maybe_make_thread_pool 的实现, 特别是动态类创建和 __reduce__ 的处理。未来若扩展 tokenizer 接口, 需确保新方法也纳入包装范围。

功能与动机

Issue #40949 报告高并发下 HF fast-tokenizer 的 Rust 后端在可变借用 (如 encode) 与不可变借用 (decode) 并发时抛出 Already borrowed。原有 workaround (deepcopy tokenizer 给多模态处理器) 不够彻底。PR body 说明: "Uses a tokenizer pool that dispatches calls to borrowing methods to a free deepcopied tokenizer instance. Fix concurrency issues with bad_words sampling param. Removes the need to use deepcopy for multimodal processor."

实现拆解

1. 核心线程池包装(vllm/tokenizers/hf.py): 新增 maybe_make_thread_pool 函数, 使用 queue.Queue 维护一组 deep-copied tokenizer 实例; 定义 _borrow_from_pool 上下文管理器借出 / 归还; 动态创建 TokenizerPool 类继承原始 tokenizer 并混入 ThreadSafeHFTokenizerMixin, 重写 encode, decode, batch_encode, batch_decode, apply_chat_template, convert_tokens_to_ids, __call__ 等方法, 全部通过 with _borrow_from_pool 委托给池中副本。新增 ThreadSafeHFTokenizerMixin 标记基类防止重复包装。
2. 调用点集成(vllm/renderers/hf.py): 在 HfRenderer.__init__ 末尾 (多模态处理器可能突变 tokenizer 之后) 调用 maybe_make_thread_pool(self.tokenizer, renderer_num_workers+1), 将 tokenizer 替换为线程安全版本; 同时做防御性拷贝 copy.copy(tokenizer) 确保原始 tokenizer 不被意外修改。
3. 移除旧 workaround(vllm/renderers/base.py): 删除创建多模态处理器时对 tokenizer 的 copy.deepcopy, 直接传递 self.tokenizer, 依赖线程池保证安全。
4. 删除重试逻辑(vllm/multimodal/processing/context.py): 移除 call_hf_processor 中针对 RuntimeError: Already borrowed 的 num_tries 重试循环 (包括 time.sleep), 因为线程

池已杜绝此类错误。

5. 导出符号(vllm/tokenizers/__init__.py): 将 `maybe_make_thread_pool` 加入 `__all__` 以便外部模块使用。

关键文件:

- vllm/tokenizers/hf.py (模块 分词器; 类别 source; 类型 dependency-wiring; 符号 ThreadSafeHFTokenizerMixin, maybe_make_thread_pool, _borrow_from_pool, TokenizerPool) : 核心实现: 新增线程安全包装逻辑、TokenizerPool 动态类、maybe_make_thread_pool 函数。
- vllm/renderers/hf.py (模块 渲染器; 类别 source; 类型 dependency-wiring; 符号 HfRenderer.init) : 调用点集成: 在 HfRenderer.init 中调用 maybe_make_thread_pool, 并处理了 tokenizer 突变前置。
- vllm/renderers/base.py (模块 渲染器; 类别 source; 类型 dependency-wiring; 符号 BaseRenderer.init) : 移除 tokenizer deepcopy workaround, 简化多模态处理器创建。
- vllm/multimodal/processing/context.py (模块 多模态; 类别 source; 类型 core-logic; 符号 InputProcessingContext.call_hf_processor) : 删除 retry 逻辑, 因 tokenizer 池已避免 Already borrowed。
- vllm/tokenizers/__init__.py (模块 分词器; 类别 source; 类型 dependency-wiring; 符号 maybe_make_thread_pool) : 导出 maybe_make_thread_pool 供外部模块使用。

关键符号: maybe_make_thread_pool, _borrow_from_pool, TokenizerPool.init, TokenizerPool.encode, TokenizerPool.decode, TokenizerPool.batch_encode, TokenizerPool.batch_decode, TokenizerPool.apply_chat_template, TokenizerPool.convert_tokens_to_ids, TokenizerPool.call, TokenizerPool.reduce, ThreadSafeHFTokenizerMixin

关键源码片段

vllm/tokenizers/hf.py

核心实现: 新增线程安全包装逻辑、TokenizerPool 动态类、maybe_make_thread_pool 函数。

```
import contextlib
import copy
import queue
from typing import TypeVar

from transformers import PreTrainedTokenizerFast
from .protocol import TokenizerLike

_T = TypeVar("_T", bound=TokenizerLike)

class ThreadSafeHFTokenizerMixin:
    """Mixin 标记类, 用于标识已包装的 tokenizer 对象。"""
    pass

def maybe_make_thread_pool(tokenizer: _T, copies: int = 1) -> _T:
```

```
"""
```

若 tokenizer 是 PreTrainedTokenizerFast, 则原地修改其类型为 TokenizerPool, 将所有公共方法调用通过深度复制池路由, 实现线程安全。

注意: 不覆盖 _tokenizer 属性及 add_special_tokens 等突变方法。

```
"""
```

```
# 只对 fast tokenizer 且尚未包装的进行包装
```

```
if not isinstance(tokenizer, PreTrainedTokenizerFast) or isinstance(
    tokenizer, ThreadSafeHFTokenizerMixin
):
    return tokenizer
```

```
og_tokenizer = copy.copy(tokenizer) # 浅拷贝作为原始模板
```

```
# 初始化深度复制池
```

```
tokenizer_pool: queue.Queue[PreTrainedTokenizerFast] = queue.Queue()
for _ in range(copies):
    tokenizer_pool.put(copy.deepcopy(og_tokenizer))
```

```
@contextlib.contextmanager
```

```
def _borrow_from_pool():
```

```
    """从池中借出一个 tokenizer, 用后归还。池空则临时 deepcopy 一个。"""
```

```
    try:
```

```
        tok = tokenizer_pool.get_nowait()
```

```
        yield tok
```

```
    except queue.Empty:
```

```
        tok = copy.deepcopy(og_tokenizer)
```

```
        yield tok
```

```
    finally:
```

```
        tokenizer_pool.put(tok)
```

```
# 动态创建 TokenizerPool 类, 继承原始 tokenizer 类并混入标记
```

```
class TokenizerPool(tokenizer.__class__, ThreadSafeHFTokenizerMixin):
```

```
    def apply_chat_template(self, *args, **kwargs):
```

```
        with _borrow_from_pool() as tok:
```

```
            return tok.apply_chat_template(*args, **kwargs)
```

```
    def batch_decode(self, *args, **kwargs):
```

```
        with _borrow_from_pool() as tok:
```

```
            return tok.batch_decode(*args, **kwargs)
```

```
    def batch_encode(self, *args, **kwargs):
```

```
        with _borrow_from_pool() as tok:
```

```
            return tok.batch_encode(*args, **kwargs)
```

```
    def convert_tokens_to_ids(self, *args, **kwargs):
```

```
        with _borrow_from_pool() as tok:
```

```
            return tok.convert_tokens_to_ids(*args, **kwargs)
```

```
    def convert_ids_to_tokens(self, *args, **kwargs):
```

```

    with _borrow_from_pool() as tok:
        return tok.convert_ids_to_tokens(*args, **kwargs)

def convert_tokens_to_string(self, *args, **kwargs):
    with _borrow_from_pool() as tok:
        return tok.convert_tokens_to_string(*args, **kwargs)

def decode(self, *args, **kwargs):
    with _borrow_from_pool() as tok:
        return tok.decode(*args, **kwargs)

def encode(self, *args, **kwargs):
    with _borrow_from_pool() as tok:
        return tok.encode(*args, **kwargs)

def __call__(self, *args, **kwargs):
    with _borrow_from_pool() as tok:
        return tok(*args, **kwargs)

def __reduce__(self):
    # 支持 pickle 序列化, 反序列化时重新创建池
    return maybe_make_thread_pool, (og_tokenizer, copies)

TokenizerPool.__name__ = f"TokenizerPool{og_tokenizer.__class__.__name__}"
tokenizer.__class__ = TokenizerPool
return tokenizer

```

评论区精华

核心方案选择：作者最初实现了锁版本和 `threading.local` 版本，reviewers 建议改用更简洁的队列池。nooooo 说 "I think using Queue() might be a bit simpler"，作者采纳。

方法覆盖范围：DarkLight1337 问 "How to determine which methods we need to wrap?"，nooooo 建议 "always use `_borrow_from_pool` for all method calls"，最终覆盖了所有公共入口。

FastIncrementalDetokenizer：作者担心 `detokenizer` 直接访问 `_tokenizer` 属性绕过池，nooooo 确认它运行在单线程核心，暂无需处理。

pool 放置时机：almayne 质疑在 `HfRenderer.__init__` 末尾才调用 `maybe_make_thread_pool` 是否安全，作者解释因为多模态处理器会突变 `tokenizer`（如添加特殊 token），必须在处理器初始化之后、但在使用之前包装。

回归风险：almayne 报告 Gemma3 多模态推理出现约 5% 性能下降，作者尚未确认原因；atalman 提出 PyTorch 2.12 上的测试失败，但作者无法稳定复现，怀疑与 torch 版本相关。

- 锁 vs 深度拷贝 vs 队列 (design): 采用基于 `queue.Queue` 的池化方案，兼顾简洁性和性能。
- 确定需要包装的方法 (design): 最终覆盖了所有公共调用入口（`call`, `encode`, `decode`, `batch_encode`, `batch_decode`, `apply_chat_template`, `convert_tokens_to_ids` 等）。

- FastIncrementalDetokenizer 绕过问题 (correctness): 当前安全, 未来如需支持多线程再迁移。
- maybe_make_thread_pool 放置位置 (design): 当前顺序正确: 先让 mm_processor 突变 tokenizer (添加特殊 token), 再调用 maybe_make_thread_pool 包装。
- Gemma3 多模态性能回归 (performance): pending: 回归是否由本 PR 引起仍不确定, 作者建议进一步调查。

风险与影响

- 风险:
 - 方法覆盖不足: 若未来新增 tokenizer 公共方法未纳入池包装, 仍可能触发 Already borrowed。当前已覆盖主要可疑方法 (encode, decode, batch_*, apply_chat_template 等), review 确认充分。
 - 突变不传播: 初始化后对 tokenizer 的运行时突变 (如 add_special_tokens) 不会同步到池中副本, 可能导致不一致。当前所有 mutation 发生在初始化阶段 (如 prompt_embeds 占位符), 安全。
 - FastIncrementalDetokenizer 绕过: detokenizer 直接访问 _tokenizer 属性不经过池, 但运行在单线程核心, 目前无并发风险; 未来若改为多线程则需适配。
 - 性能风险: 引入 deepcopy 和队列操作, benchmark 未发现退化, 但 almayne 报告 Gemma3 多模态 5% 下降, 需进一步确认是否由本 PR 导致。
 - 类型兼容性: 动态改变 tokenizer 的 __class__ 可能影响依赖 isinstance 或 type() 的代码, 但 ThreadSafeHFTokenizerMixin 标记可缓解。
- 影响:
 - 用户: 消除高并发下 RuntimeError: Already borrowed 错误, 尤其是使用 bad_words 参数和多模态请求时更加稳定。
 - 系统: 移除旧的 deepcopy 和重试逻辑, 减少不必要的内存拷贝和延迟; 线程池引入少量额外内存 (每个 worker 一个 deep-copied 副本), 通常可接受。
 - 团队: 简化多模态处理器的 tokenizer 传递, 不再需要手动 deepcopy, 降低后续开发的心智负担。
 - 风险标记: 方法覆盖不足, 突变不传播, 性能回归, 类型兼容性

关联脉络

- PR #40720 [Prompt] Tokenizer mutation in prompt_embeds handling: 作者在评论中指出 #40720 中的 tokenizer 突变操作导致 CI 测试失败, 因此本 PR 的 tokenizer 池需要特别处理 mutation。