PR #25265 完整报告

sgl-project/sglang

[perf] fix kimi tokenizer to improve ttft

合并时间：2026-05-15 10:11

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25265

执行摘要

- 一句话：优化 Kimi 等 tiktoken 衍生 tokenizer 的 TTFT
- 推荐动作：建议精读该 PR，尤其是其中条件分支的设计思路：通过检测 is_fast 属性，在慢速 tokenizer 上切换为 .encode() 循环，同时保留 fast tokenizer 的批量优化。这是一个典型的最小改动、最大收益的案例，值得在类似性能优化中参考。

功能与动机

SGLang 的 TokenizerManager._tokenize_texts 统一使用 tokenizer.call() 进行编码，但对于 tiktoken 衍生 tokenizer（如 Kimi-K2.5），该路径强制经过 PreTrainedTokenizer 的慢速 Python 管道，无法利用原生的 tiktoken fast path（仅能通过 .encode() 无 kwargs 到达）。PR body 中的微基准测试显示，在 80k tokens 单文本下，__call__耗时 308ms 而 encode 仅需 47ms，固定开销约 260ms/ 请求。E2E 基准测试（input len 80000, output len 1）显示 TTFT 从 3151ms 降至 2932ms，节省约 5-7%。

实现拆解

在 TokenizerManager._tokenize_texts 的 Step 3 分支中，对非 cross-encoder 且非 fast tokenizer 的场景，将统一的 encoded = self.tokenizer(tokenizer_input, **tokenizer_kwargs) 替换为逐条 self.tokenizer.encode(t) 的列表推导。

具体变更步骤：

1. 文件：python/sglang/srt/managers/tokenizer_manager.py，函数 _tokenize_texts。
2. 入口条件：在 else 分支（不使用 async_dynamic_batch_tokenizer）内部，新增针对非 cross-encoder 且 non-fast tokenizer 的判断。
3. 慢路径分支：当 not is_cross_encoder 且 not getattr(self.tokenizer, 'is_fast', False) 时，使用 [self.tokenizer.encode(t) for t in tokenizer_input]，绕过 __call__ 的 Python 包装层，直接调用 tiktoken 原生编解码。同时将 token_type_ids 置为 None。
4. 快路径 /fallback：其他情况（cross-encoder 或 fast tokenizer）保留原有的 self.tokenizer(tokenizer_input, **tokenizer_kwargs) 调用，确保批量并行化和 token_type_ids 传递不受影响。
5. 返回值：两种路径的输出格式均与步骤 4 的 _extract_tokenizer_results 兼容，无需额外适配。

变更仅修改 1 个文件，+10/-3 行，无测试 / 配置 / 部署配套改动。

关键文件：

- python/sglang/srt/managers/tokenizer_manager.py（模块 调度器；类别 source；类型 core-logic）：核心变更文件，修改了 _tokenize_texts 方法的 tokenization 策略，新增条件分支以优化 tiktoken 衍生 tokenizer 的 TTFT。

关键符号：_tokenize_texts

关键源码片段

python/sglang/srt/managers/tokenizer_manager.py

核心变更文件，修改了 _tokenize_texts 方法的 tokenization 策略，新增条件分支以优化 tiktoken 衍生 tokenizer 的 TTFT。

```
def _tokenize_texts(...): # 位于 tokenizer_manager.py
    ...
    # Step 3: Choose tokenization strategy
    ...
    else:
        logger.debug(f"Using regular tokenizer for {len(tokenizer_input)} inputs")

        # 对非 cross-encoder 且非 fast tokenizer（如 Kimi K2.5 的 tiktoken 衍生）
        # 使用 .encode() 循环绕开 __call__ 的慢速 Python 包装层
        # 参考：https://huggingface.co/moonshotai/Kimi-K2.5
        if not is_cross_encoder and (not getattr(self.tokenizer, "is_fast", False)):
            input_ids = [self.tokenizer.encode(t) for t in tokenizer_input]
            token_type_ids = None
        else:
            # 对 cross-encoder 或 fast tokenizer（Llama、Qwen 等）
            # 保留批量路径以利用 Rust 并行加速和 token_type_ids 支持
            encoded = self.tokenizer(tokenizer_input, **tokenizer_kwargs)
            input_ids = encoded["input_ids"]
            token_type_ids = (
                encoded.get("token_type_ids") if is_cross_encoder else None
            )

    # Step 4: Extract results based on input format
    return self._extract_tokenizer_results(
        input_ids, token_type_ids, input_format, original_batch_size
    )
```

评论区精华

主要讨论来自 gemini-code-assist[bot] 的 review 评论，指出使用 self.tokenizer.encode 循环在批量请求下会丢失 HuggingFace fast tokenizer 的 Rust/C++ 并行加速能力，建议当 len(tokenizer_input) > 1 时仍使用批量方法。PR 作者通过提交中的分支条件（not getattr(self.tokenizer, 'is_fast', False)）做出回应：仅对 non-fast tokenizer 使用循环，

fast tokenizer 仍走批量路径，从而规避了此风险。其他 reviewer（ByronHsu, Fridge003, kpham-sgl）均表示认可。

- 批量请求下循环 .encode() 的性能回归风险 (performance): PR 作者通过 not getattr(self.tokenizer, 'is_fast', False) 分支条件将循环限制在 non-fast tokenizer 上，fast tokenizer 仍走批量路径，从而消除了回归风险。

风险与影响

- 风险：
  1. 回归风险：对于 fast tokenizer（如 Llama、Qwen），新增的条件判断不会改变其原有批量路径，回归风险低。但对于某些自定义 fast tokenizer 其 is_fast 属性可能为 False，会不必要地走慢速循环，不过这类 tokenizer 较少且 .encode() 在单次调用时通常不会比 __call__ 更慢。
  2. 正确性风险：非 cross-encoder 场景下 token_type_ids 被直接置为 None，与原路径行为一致，无差异。
  3. 性能风险：当 batch size 较大（如 >16）时，循环 .encode() 的 Python 开销可能累积，但由于已限定仅在 non-fast tokenizer 上执行，而 non-fast tokenizer 本身无批量加速，因此该路径是净收益。 - 影响：影响范围：tiktoken 衍生 tokenizer（Kimi-K2.5、部分 OpenAI 系列）在非 cross-encoder、非 fast tokenizer 场景下，TTFT 减少 5-7%，尤其长输入场景节省显著（~260ms/ 请求）。其他 tokenizer 无行为变化。影响程度：中等——优化集中在特定 tokenizer 类型，但收益明确，且通过分支条件确保了 fast tokenizer 的批量性能不受损。团队无需额外配置或迁移。 - 风险标记：非 fast tokenizer 误判可能导致性能回退

关联脉络

- PR #24185 [Fix] load_audio: fall back to soundfile when torchcodec fails on WAV with trailing metadata: 同属 'sglang/srt' 模块下的 bugfix/ 性能改进，展示了对 tokenizer_manager.py 及相关工具函数的持续关注。