

PR #34187 完整报告

sgl-project/sglang

[Kimi K3] Rework skipped-think fix as opt-in force_nonempty_content with streaming coverage

合并时间: 2026-09-03 17:37

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/34187>

执行摘要

- 一句话: K3 跳过 think 修复改为 opt-in 开关并覆盖流式
- 推荐动作: 值得精读, 尤其是负责流式解析、reasoning 输出后处理与前端 SSE 适配的工程师。建议重点关注四处设计决策: 一是为什么用「先实时发 reasoning、结束后补 content」而不是 hold-back (30 万字符级长思考下的客户端 idle 超时是硬约束); 二是把 `_in_reasoning` 当作「流中是否已做通道决策」的状态位来守卫补发, 避免内容重复; 三是 `stream_reasoning=False` 下必须在子类自行冲刷 reasoning, 因为基类在 fnc 语义下会把它当 content; 四是 `_is_skipped_think_answer` 为什么要求关闭标记作为正向证据——这正是与 #33995 的分水岭。另外, 评估该 PR 时不要只看空内容率的下降, 还要确认下游对「同一段文本既出现在 reasoning_content 又出现在 content」的容忍度, 以及是否愿意把它写成 K3 的默认服务端配置。

功能与动机

Kimi K3 有时在 think 通道内直接 EOS, 回答以 reasoning_content 返回而 content 为空 (BEAM 1M 上 110-127/700 条)。#33995 已做出一版修复, 但 PR body 指出两个泄漏: 「截断的思考被当成答案: max_tokens 截断同样产生零 think 标记, 于是被误判为 content」, 以及「流式仍然是坏的: 该分支只存在于 detect_and_parse, parse_streaming_increment 以 force_reasoning=True 起步, 把原始 payload 当作 reasoning_text 流出去, stream=True 依旧返回 content=""」。此外 K3 之前不接受 force_nonempty_content 参数, 通过 chat_template_kwargs 传入会直接 TypeError。

实现拆解

1. 开关入口与默认值: KimiK3Detector.__init__ 新增 force_nonempty_content: bool = False, 并保存为 self._force_nonempty_content, 同时新增 self._stream_text 旁路缓冲。默认关闭意味着所有既有 K3 部署行为不变, 开关通过 --default-chat-template-kwarg ' {"force_nonempty_content": true}' 或请求级 chat_template_kwargs 打开。
2. 非流式改判改为「正向证据」驱动: detect_and_parse 在原有的 force_reasoning 分支之后插入一段判断, 调用新方法 _is_skipped_think_answer; 该方法要求四种 think 标记形态 (含 removesuffix("<lsep|>") 的裸名变体) 全部缺席, 且 RESPONSE_CLOSE 或 MESSAGE_CLOSE 至少出现一个, 因此无关闭标记的 max_tokens 截断文本仍留在 reasoning 侧。

3. 流式路径选择「不扣留 chunk」：parse_streaming_increment 只在开关打开时把 new_text 累加到 _stream_text，其余逐 chunk 逻辑与原来完全一致。PR body 明确说明 hold-back 会让长思考期间 SSE 没有 delta，实测触发约 300 s 的客户端 idle 超时；代价是同一段文本会先以 reasoning 实时下发，通道判定成立后再由 finish() 补发为 content。
4. finish() 的门控与冲刷：新增的 finish() 未开启开关时直接 super().finish()；开启时先取出 _stream_text，只有 self._in_reasoning 为真（说明流中从未发生过通道切换）才补发 content，避免对已经流出的答案重复下发；对 stream_reasoning=False 的情况，子类显式把缓冲按 reasoning 冲刷，而不是交给基类（基类在 fnc 语义下会把它误判成 content）。这是 commit 5dbba68 专门修的第二个回归。
5. 测试配套：test/registered/unit/parser/test_kimik3_reasoning_parser.py 新增 helper _stream_with_finish，把「逐 chunk 流式」与「finish() 补发」合并统计，并新增 7 个用例覆盖：非流式 skipped-think 与截断思考的区分、流式补发（chunk 1/5/13）、无关闭证据仍留在 reasoning、20k 字符长思考逐 chunk 实时流出、RESPONSE_OPEN 与 force_reasoning=False 不重复补发、stream_reasoning=False 下的截断冲刷与 skipped-think 补发。没有配置、schema 或部署文件改动。

关键文件：

- python/sglang/srt/parser/reasoning_parser.py（模块 推理解析器；类别 source；类型 core-logic；符号 KimiK3Detector.init, KimiK3Detector.detect_and_parse, KimiK3Detector._is_skipped_think_answer, KimiK3Detector.parse_streaming_increment）：唯一源码改动文件，集中承载 K3 跳过 think 的判定（_is_skipped_think_answer）、流式原文旁路累积（_stream_text）与收尾补发（finish）；默认关闭保证零回归。
- test/registered/unit/parser/test_kimik3_reasoning_parser.py（模块 解析器测试；类别 test；类型 test-coverage；符号 _stream_with_finish, test_fnc_non_stream_skipped_think_vs_truncated_reasoning, test_fnc_streaming_skipped_think_answer, test_fnc_streaming_truncated_reasoning_stays_reasoning）：新增 7 个用例覆盖非流式改判边界、流式补发、长思考不扣留、重复下发门控与 stream_reasoning=False 冲刷路径，并引入 _stream_with_finish helper 统一「流式 + 收尾」断言。

关键符号：KimiK3Detector.init, KimiK3Detector.detect_and_parse, KimiK3Detector._is_skipped_think_answer, KimiK3Detector.parse_streaming_increment, KimiK3Detector.finish, _stream_with_finish

关键源码片段

python/sglang/srt/parser/reasoning_parser.py

唯一源码改动文件，集中承载 K3 跳过 think 的判定（_is_skipped_think_answer）、流式原文旁路累积（_stream_text）与收尾补发（finish）；默认关闭保证零回归。

python/sglang/srt/parser/reasoning_parser.py 中三处核心新增：改判判定、流式旁路累积、以及收尾补发。

```
def detect_and_parse(self, text: str) -> StreamingParseResult:
    in_reasoning = self._in_reasoning or self.think_start_token in text
```

```

if not in_reasoning and self.think_end_token not in text:
    return StreamingParseResult(normal_text=self._clean_content(text))
# 只有开启开关、且拿到「正向证据」时才改判：
# 文本里完全没有 think 标记，却出现了 response / message 的关闭标记。
# 原先「无 think 标记即 content」的写法会把 max_tokens 截断的纯思考误判成答案。
if self._force_nonempty_content and self._is_skipped_think_answer(text):
    return StreamingParseResult(normal_text=self._clean_content(text))
# 以下与基线完全一致：仍按 think 区间切分 reasoning / content。
...

def _is_skipped_think_answer(self, text: str) -> bool:
    # 四种 think 标记形态都必须缺席，包含 `removesuffix("<lsepl>")` 得到的裸名变体
    # （strict-thinking 会把完整标记压平成单个 token，裸名同样不可出现）。
    # 后半段的关闭标记要求才是关键：没有它，截断思考与跳过 think 无法区分。
    return (
        self.think_start_token not in text
        and self.think_end_token not in text
        and self.think_start_token.removesuffix("<lsepl>") not in text
        and self.think_end_token.removesuffix("<lsepl>") not in text
        and (RESPONSE_CLOSE in text or MESSAGE_CLOSE in text)
    )

def parse_streaming_increment(self, new_text: str) -> StreamingParseResult:
    self._buffer += new_text
    if self._force_nonempty_content:
        # 只做旁路累积，不改变 chunk 的吐出时机：
        # 任何 hold-back 都会让长思考期间 SSE 长时间没有 delta，触发客户端空闲超时。
        self._stream_text += new_text
    ... # 其余逐 chunk 逻辑与基线一致，实时把未判定的文本当作 reasoning 流出

def finish(self) -> StreamingParseResult:
    # 未开启开关时完全走基类逻辑，保证既有部署零行为变更。
    if not self._force_nonempty_content:
        return super().finish()

    text, self._stream_text = self._stream_text, ""

    if self._in_reasoning and self._is_skipped_think_answer(text):
        # `_in_reasoning` 表示流式过程中从未发生过通道切换，答案此前是以 reasoning 形式
        # 实时下发的，这里补发一次清洗后的 content。缺少这道门控，已经按 content 流出过的
        # 答案（出现 RESPONSE_OPEN，或 force_reasoning=False）会被重复下发。
        self._buffer = ""
        return StreamingParseResult(normal_text=self._clean_content(text))

    if self._in_reasoning and not self._stream_reasoning and self._buffer:
        # 这里不能交给 `super().finish()`：在 force_nonempty_content 语义下基类会把这段
        # 缓冲当作 content 发出，正是本开关要堵住的泄漏，因此显式按 reasoning 冲刷。
        buffer, self._buffer = self._buffer, ""
        return StreamingParseResult(reasoning_text=buffer)

```

```
return StreamingParseResult()
```

test/registered/unit/parser/test_kimik3_reasoning_parser.py

新增 7 个用例覆盖非流式改判边界、流式补发、长思考不扣留、重复下发门控与 stream_reasoning=False 冲刷路径，并引入 _stream_with_finish helper 统一「流式 + 收尾」断言。

测试新增的合并断言 helper 与三类关键用例：

```
def _stream_with_finish(detector: KimiK3Detector, chunks: list[str]) -> tuple[str, str]:
    # 把「逐 chunk 流式」与「流结束后的 finish() 补发」拼在一起统计，
    # 才能覆盖 skipped-think 场景下「reasoning 实时流出 + content 由 finish() 补发」的组合语义。
    reasoning, content = _stream(detector, chunks)
    result = detector.finish()
    return reasoning + result.reasoning_text, content + result.normal_text
```

```
@pytest.mark.parametrize("chunk_size", [1, 5, 13])
def test_fnc_streaming_skipped_think_answer(chunk_size: int) -> None:
    detector = KimiK3Detector(force_reasoning=True, force_nonempty_content=True)
    text = f"bare answer{RESPONSE_CLOSE}{MESSAGE_CLOSE}"
    reasoning, content = _stream_with_finish(detector, _chunks(text, chunk_size))
    # 实时阶段整段文本按 reasoning 下发；关闭标记出现后由 finish() 补发清洗过的 content。
    assert reasoning == text
    assert content == "bare answer"
```

```
@pytest.mark.parametrize("chunk_size", [5, 13])
def test_fnc_streaming_long_think_streams_without_close(chunk_size: int) -> None:
    detector = KimiK3Detector(force_reasoning=True, force_nonempty_content=True)
    text = "x" * 20000
    reasoning = ""
    for chunk in _chunks(text, chunk_size):
        # 逐 chunk 立即校验，确保长思考期间没有 hold-back，否则 SSE 会长时间无 delta
        # 触发客户端超时。
        reasoning += detector.parse_streaming_increment(chunk).reasoning_text
    assert reasoning == text
    result = detector.finish()
    # 没有任何关闭证据：既不补发 content，也不把已流出的思考再吐一遍。
    assert result.reasoning_text == ""
    assert result.normal_text == ""
```

```
def test_fnc_streaming_response_open_not_reemitted() -> None:
    detector = KimiK3Detector(force_reasoning=True, force_nonempty_content=True)
    text = f"{RESPONSE_OPEN}bare answer{RESPONSE_CLOSE}{MESSAGE_CLOSE}"
    reasoning, content = _stream_with_finish(detector, _chunks(text, 5))
    # 中途已发生通道切换、答案早已按 content 流出，finish() 不得再补发一次。
    assert reasoning == ""
```

```
assert content == "bare answer"
```

评论区精华

讨论量很小，但 PR body 本身承载了主要的设计交锋：

- 关于原修复的判定口径，PR body 写道：#33995 fixed the non-stream path by treating marker-less output as content, which has two leaks —— 截断思考被误判为答案，以及流式路径完全没修。结论是放弃「无标记即 content」，改为要求关闭标记作为正向证据的 opt-in 开关。
- 关于流式实现，PR body 明确拒绝 hold-back: A hold-back would starve SSE during long thinking runs (no delta until the channel decision) and trip client idle timeouts (observed: ~300 s client-side terminated while the server is still decoding), 改为「实时发 reasoning、finish() 再补 content」。
- commit 5dbba68 记录了两处自审发现的回归并修复：finish() 补发在已做过通道决策的流上会重复下发内容，因此以 _in_reasoning 为门控；同时 stream_reasoning=False 下基类会把缓冲当 content 发出，子类改为显式按 reasoning 冲刷。PR body 称这两个回归用例在打补丁前会失败。
- 评审侧：原作者 yhyang201 以 APPROVED 通过且未留评论，whybeyoung 在 issue 评论中给出 LGTM；JustinTong0323 仅留下一条 /rerun-failed-ci，PR body 的 CI 状态显示 PR Test (Extra) 与 AMD ROCm 7.2 为失败，未见到失败归因结论。
- 废弃「无 think 标记即 content」，改为要求关闭标记作为正向证据 (design): 改为 opt-in 的 force_nonempty_content，判定收窄为「零 think 标记 + 出现 RESPONSE_CLOSE/MESSAGE_CLOSE」，默认关闭。
- 流式路径拒绝 hold-back，改为 finish() 阶段补发 content (performance): 实时 reasoning_content + finish() 补发 content；代价是同一段文本会出现两次（不同字段），由使用方自行适配。
- finish() 重复下发与 stream_reasoning=False 冲刷回归 (correctness): 以 _in_reasoning 门控补发，仅在流中从未发生通道决策时触发；非 fnc 路径走 super().finish()，fnc 路径显式按 reasoning 冲刷。PR body 称相关用例在打补丁前会失败。
- 评审与 CI: APPROVED + LGTM，但 Extra/AMD 流水线为失败 (testing): 代码层面无阻塞意见并已合入，但失败流水线缺少归因记录，无法从材料确认是否与本次改动相关。

风险与影响

- 风险：
 1. 同一段文本下发两次（语义风险，有意为之）：开启开关后，skipped-think 的回答会先以 reasoning_content 实时推送，再由 finish() 以 content 补发。对按 delta 累加、去重或计费的客户端 / 网关，需要显式处理这段重复文本。
 2. 截断场景仍有空洞：_is_skipped_think_answer 依赖 RESPONSE_CLOSE/MESSAGE_CLOSE 作为正向证据；若模型跳过 think 后答案本身被 max_tokens 截断在关闭标记之前，content 仍会为空。PR body 中剩余的 4/700 被归因为 finish=length 的思考中截断，但「答案区截断」这一形态未被该判据覆盖。

3. 每请求额外保留全量输出：parse_streaming_increment 里 self._stream_text += new_text 会为开启开关的请求保留一份完整流式原文（K3 评测场景接近 32k token 上限），在 Python 层逐 chunk 拼接存在重复拷贝开销，属于以内存 /CPU 换取「不扣留 chunk」的取舍，仅在开关打开时生效。
4. 开关透传语义：force_nonempty_content 经由 chat_template_kwargs 进入解析器，同一份 kwargs 也会流向 chat template；其他 detector（含基类）对该参数的处理口径与 K3 不同，跨模型复用同名 kwarg 时需注意语义差异。
5. 回归面已收敛但仍靠开关自律：默认 False 时 finish() 首行即 super().finish() 早返回，未开启部署与基线一致；只要有人把该开关作为 K3 默认配置写进服务端，风险 1 与风险 2 就直接暴露给所有用户。
6. CI 未完全绿灯：PR body 显示 PR Test (Extra)、AMD ROCm 7.2 两条流水线为失败，仅以 /rerun-failed-ci 处理，缺少失败原因记录，无法从材料确认其与本次改动的因果关系。
 - 影响：用户侧：对 Kimi K3 端点开启该开关后，BEAM 1M（700 题、近 1M 上下文）空内容率由 17.57%（123/700）降到 0.71%（5/700），其中典型的 skipped-think 模式由 16.6%（116/700）降到 0.14%（1/700），LLM 评审整体分 0.288 → 0.333，且两名独立评审方向一致。默认关闭，未配置的部署零感知。系统侧：影响面严格限定在 KimiK3Detector（即 ReasoningParser("kimi_k3")）内；finish() 属于解析器公共生命周期钩子，所有流式收尾都会经过，但新逻辑全部由 _force_nonempty_content 与 _in_reasoning 双重门控，不影响其他模型解析器。团队侧：该 PR 在基类已有 force_nonempty_content 的前提下，示范了「模型特有判定无法复用基类启发式时，如何在子类内重写判定 + 收尾」的范式（_is_skipped_think_answer 的正向证据判据、_in_reasoning 作为「流中是否已做通道决策」的状态位），对 K2、GLM 等 detector 的同类问题具备参考价值；_stream_with_finish 这种「流式 + finish 合并断言」的测试写法也可以被其他解析器测试复用。
- 风险标记：同一文本重复下发（reasoning 实时 + content 补发），答案区 max_tokens 截断仍可能 content 为空，开启后每请求保留全量流式原文，CI Extra/AMD ROCm 流水线失败未归因，开关语义依赖使用方显式配置

关联脉络

- PR #33995 [Kimi K3] Fix reasoning parser returning empty content when model skips think channel: 本 PR 明确 stacked on top of 它，保留原作者 commit d11868d7d9 并做 rework：把「无标记即 content」改成 opt-in 的 force_nonempty_content，并补上它缺失的流式路径。
- PR #39227 Force reasoning mode for GLM-5.3 chat templates: 同样落在 python/sglang/srt/parser 下的推理模式 / 模板解析语义修复，说明「推理通道判定」在这段时间是一条持续演进的线，本 PR 的 opt-in 开关范式可能被同类 detector 借鉴。
- PR #39318 Scope prefetch cache state to the request attempt: 同一时期对「跨请求状态污染」的重构类修复，与本次为 K3 引入的 _stream_text 请求内状态累积形成对照，可参考其状态生命周期治理思路。