PR #35480 完整报告

sgl-project/sglang

[Fix] Pass Anthropic thinking history as reasoning_content for custom chat encoders

合并时间：2026-08-22 02:22

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35480

执行摘要

- 一句话：修复自定义编码器下 Anthropic 历史思考标记泄漏
- 推荐动作：值得精读。三点设计值得借鉴：一是从渲染产物反推根因——用修复前后经真实 tokenizer 的渲染对照定位频道错位，而不是只看解析器行为；二是契约用 spec is not None 覆盖整个编码器家族并明确安全默认方向（丢弃优于泄漏），避免未来新增编码器踩同样的坑；三是用源码解析守卫测试把"新增编码器必须分类"变成 CI 硬约束并做 mutation-check。若后续增强，建议补 dsv4 / dsv32 / inkling 的端到端多轮回归，并把 reasoning_content 的双向验证收进契约测试。

功能与动机

按 PR body 的描述，用 Kimi-K3 跑 --reasoning-parser kimi_k3 --tool-call-parser kimi_k3 并在 /v1/messages 开启 thinking 时，agent 客户端按 Messages API 要求把 thinking 块回传历史，几轮之后原始 XTML 频道标记（如 <|openl>think<|sepl>）出现在可见文本中且永不恢复。作者通过去掉 thinking 块的对照实验确认根因在"历史推理的回灌方式"而非推理解析器：K3 没有 Jinja chat template，SGLang 走 checkpoint 自带的 XTML 编码器，该编码器自行框架 think / response / tools 频道并期望历史推理放在 assistant 的 reasoning_content 字段；而 Anthropic 适配器从未使用该字段，反而把历史思考用推理检测器标记包裹后拼进 content，等于每轮都在教模型把频道标记当正文输出。

实现拆解

变更入口

整个修复以 chat_encoding.py 新增的 spec_owns_reasoning_history 契约为入口，沿 serving_chat.py 的能力委托，最终在 anthropic/serving.py 的 Anthropic → OpenAI 消息转换中消费。

1. 契约定义（python/sglang/srt/entrypoints/openai/chat_encoding.py）：新增 spec_owns_reasoning_history(spec)，返回 spec is not None，即所有绕过 apply_chat_template 的自定义编码器（dsv4 / dsv32 / inkling / kimi_k3）都被认定"自行负责推理历史渲染"；同时更新 resolve_chat_encoding_spec 的 docstring，使契约与规格定义放在一起。选择按整个编码器家族作答而非枚举名单，让未来新增的第五种编码器自动获得安全默认：最坏情况是丢弃历史，而不是泄漏标记。

2. 能力暴露（python/sglang/srt/entrypoints/openai/serving_chat.py）：OpenAIServingChat 新增 supports_native_reasoning_history()，一行委托给 chat_encoding.spec_owns_reasoning_history(self.chat_encoding_spec)，供 Anthropic 适配器查询当前模型的编码器是否原生接收 reasoning_content。
3. 消费端改造（python/sglang/srt/entrypoints/anthropic/serving.py）：_convert_assistant_thinking_blocks 返回类型从 Optional[str] 扩展为 (Optional[str], Optional[str]) 二元组，分别表示 reasoning_content 与需 wrap 进 content 的文本，二者至多一个非 None。组装 assistant 消息时，命中原生能力则写入 openai_msg["reasoning_content"]，否则走原 wrap_reasoning_history 拼接路径；redacted_thinking 仍在入口抛错，无 detector 模型的尽力丢弃逻辑原样保留。无需 schema 变更：ChatCompletionMessageGenericParam 已声明 reasoning_content，消息序列化未排除 None，字段自然透传。
4. 测试配套：anthropic/test_serving.py 新增 2 个用例——历史思考必须到达 reasoning_content 且绝不经过 wrap 路径（用抛 AssertionError 的桩验证）、纯思考轮在空 content 占位分支下仍保留推理字段；openai/test_serving_chat.py 新增能力枚举用例（四个 spec 加 None）与源码解析守卫测试（正则解析 resolve_chat_encoding_spec 的 return 字面量并与 _ALL_CHAT_ENCODING_SPECS 比对，新增编码器未分类时 CI 失败）。作者声明新用例对修复前代码均红且经过 mutation-check，并非空洞断言；实机验证覆盖流式 / 非流式 / 并发会话 / 禁用 thinking / redacted_thinking 等多种场景。

关键文件：

- python/sglang/srt/entrypoints/anthropic/serving.py（模块 协议适配；类别 source；类型 core-logic；符号 _convert_assistant_thinking_blocks）：核心修复点：_convert_assistant_thinking_blocks 改为返回 (reasoning_content, text) 二元组，并依据 supports_native_reasoning_history() 分支决定历史思考走字段还是走 wrap 拼接；所有经 Anthropic /v1/messages 携带 thinking 历史的请求都经过这条路径。
- python/sglang/srt/entrypoints/openai/chat_encoding.py（模块 聊天编码；类别 source；类型 core-logic；符号 spec_owns_reasoning_history, resolve_chat_encoding_spec）：新增 spec_owns_reasoning_history 契约，定义“非 None spec 即拥有推理历史渲染权”，是整个修复的判定依据；契约与 resolve_chat_encoding_spec 放在一起，让规格定义与契约绑定，未来新增编码器自动获得安全默认。
- python/sglang/srt/entrypoints/openai/serving_chat.py（模块 聊天服务；类别 source；类型 core-logic；符号 supports_native_reasoning_history）：新增 supports_native_reasoning_history() 一行委托，把 chat_encoding 契约暴露给 Anthropic 适配器查询；与 wrap_reasoning_history 并列放置，明确两条历史渲染路径的边界。
- test/registered/unit/entrypoints/anthropic/test_serving.py（模块 协议测试；类别 test；类型 test-coverage；符号 supports_native_reasoning_history, test_assistant_thinking_history_uses_native_reasoning_content, _NativeOpenAI, test_thinking_only_turn_keeps_native_reasoning_content）：新增两个单测钉死新行为：历史思考必须到达 reasoning_content 且绝不经过 wrap 路径（wrap 被替换为抛 AssertionError 的桩），纯思考轮在空 content 占位分支下仍保留推理字段；两个用例对

修复前代码均红。

- test/registered/unit/entrypoints/openai/test_serving_chat.py（模块 聊天测试；类别 test；类型 test-coverage；符号 test_custom_encoders_own_reasoning_history, test_all_chat_encoding_specs_are_enumerated, _ALL_CHAT_ENCODING_SPECS）：test_custom_encoders_own_reasoning_history 对全部四个 spec 加 None 断言能力枚举；test_all_chat_encoding_specs_are_enumerated 用正则解析源码 return 字面量并钉住 spec 名单，新增编码器未分类时 CI 立即失败，且经 mutation-check。

关键符号：spec_owns_reasoning_history, supports_native_reasoning_history, _convert_assistant_thinking_blocks, test_assistant_thinking_history_uses_native_reasoning_content, test_thinking_only_turn_keeps_native_reasoning_content, test_custom_encoders_own_reasoning_history, test_all_chat_encoding_specs_are_enumerated

关键源码片段

python/sglang/srt/entrypoints/anthropic/serving.py

核心修复点：_convert_assistant_thinking_blocks 改为返回 (reasoning_content, text) 二元组，并依据 supports_native_reasoning_history() 分支决定历史思考走字段还是走 wrap 拼接；所有经 Anthropic /v1/messages 携带 thinking 历史的请求都经过这条路径。

```
def _convert_assistant_thinking_blocks(
    blocks: list[AnthropicContentBlock],
) -> tuple[Optional[str], Optional[str]]:
    """把上一轮 thinking 块还原为 (reasoning_content, text) 二元组，至多一个非 None。

    自带频道框架的编码器（custom encoder，如 dsv4 / dsv32 / kimi_k3 / inkling）
    通过 reasoning_content 接收历史思考；其余模型沿用 wrap 拼接进 content。
    """
    # redacted_thinking 携带本地解析器无法解读的加密字节，宁可抛错也不静默丢弃
    if any(block.type == "redacted_thinking" for block in blocks):
        raise ValueError("Anthropic redacted_thinking history is not supported")

    thinking_parts = [
        block.thinking
        for block in blocks
        if block.type == "thinking" and block.thinking
    ]
    if not thinking_parts:
        return None, None

    reasoning_text = "\n".join(thinking_parts)
    # 关键分支：编码器自己框架 think 频道，历史思考必须走 reasoning_content，
    # 否则 think 块会嵌套进 response 频道而真频道为空，等于每轮教模型输出原始标记
    if self.openai_serving_chat.supports_native_reasoning_history():
        return reasoning_text, None

    # 默认路径（Jinja chat template）：用检测器标记包裹后拼进 content，
```

```
        # 标记本来就活在内容流里，能忠实还原模型此前的输出
        try:
            return None, self.openai_serving_chat.wrap_reasoning_history(reasoning_text)
        except ValueError as e:
            # 没有配置 reasoning detector 的模型：尽力而为，丢弃历史思考，
            # 避免历史回显把整个请求打成 400
            logger.warning(
                "Dropping prior-turn thinking history (%d blocks): %s",
                len(thinking_parts),
                e,
            )
            return None, None


    # 调用侧：组装 assistant 消息时按返回值分流
    if msg.role == "assistant":
        reasoning_content, reasoning_history = _convert_assistant_thinking_blocks(msg.content)
        # 自定义编码器：历史思考直接进 reasoning_content 字段，随消息透传
        if reasoning_content is not None:
            openai_msg["reasoning_content"] = reasoning_content
        # Jinja 路径：wrap 后的文本作为 content 片段，放在正文最前
        if reasoning_history is not None:
            content_parts.append({"type": "text", "text": reasoning_history})
```

python/sglang/srt/entrypoints/openai/chat_encoding.py

新增 spec_owns_reasoning_history 契约，定义“非 None spec 即拥有推理历史渲染权”，是整个修复的判定依据；契约与 resolve_chat_encoding_spec 放在一起，让规格定义与契约绑定，未来新增编码器自动获得安全默认。

```
def resolve_chat_encoding_spec(
    *, hf_config: Any, tokenizer: Any, tool_call_parser: Optional[str] = None
) -> Optional[str]:
    """返回模型对应的聊天编码 spec。

    None 表示默认路径（HF chat template）；非 None 的 spec 同时拥有
    推理历史渲染权（见 spec_owns_reasoning_history）。
    """
    # 工具解析器优先：显式指定 deepseekv4 / deepseekv32 / kimi_k3 时直接命中
    if tool_call_parser == "deepseekv4":
        return "dsv4"
    if tool_call_parser == "deepseekv32":
        return "dsv32"
    if tool_call_parser == "kimi_k3":
        return "kimi_k3"

    architectures = hf_config.architectures
    arch = architectures[0] if architectures else ""

    # 架构兜底：DeepSeek-V4、Kimi-K3 等模型没有可用 Jinja 模板
```

```
    if "DeepseekV4" in arch:
        return "dsv4"
    if "KimiK3" in arch:
        return "kimi_k3"

    # Inkling 同样没有 Jinja chat_template，只能走专用渲染器
    if "InklingForConditionalGeneration" in arch:
        return "inkling"

    has_chat_template = tokenizer is not None and tokenizer.chat_template is not None
    if "DeepseekV3" in arch and not has_chat_template:
        return "dsv32"
    return None


def spec_owns_reasoning_history(spec: Optional[str]) -> bool:
    """判断 spec 对应的编码器是否自行渲染助手推理历史。

    自定义编码器自己框架 think / response 频道并读取 reasoning_content，
    历史思考必须走该字段传入；若沿用旧逻辑把检测器标记拼进 content，
    会在 response 频道内部嵌套一个 think 块而真正的 think 频道为空，
    等于每轮都在教模型把原始标记当可见文本输出。

    按整个编码器家族（spec is not None）作答而非枚举名单：
    新增编码器自动拿到安全默认 —— 最坏情况是丢弃历史，而不是泄漏标记。
    """
    return spec is not None
```

评论区精华

本 PR 无 review 评论线程（review_comments = 0），唯一审核为 JustinTong0323 的 APPROVED，附言 "LGTM"。核心设计论证全部集中在 PR body：作者给出了修复前后经 K3 tokenizer 实际渲染的对照（修复前 think 频道为空、思考块嵌套在 response 频道内；修复后 think 频道正确承载历史思考），并论证“所有四个绕过 apply_chat_template 的 Python 聊天编码器都自行框架推理频道并读取 reasoning_content，因此旧的拼接逻辑对它们是普遍性错误而非 K3 特例”。同时对两处边界做了明确取舍：Jinja 模板路径故意保留拼接（其标记本来就活在内容流里，拼接能忠实还原模型输出）；忽略该字段的编码器最坏丢弃历史而不泄漏。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 行为面广：spec is not None 一票覆盖 dsv4 / dsv32 / inkling / kimi_k3 四个编码器，多轮历史渲染路径全部从 wrap 切换为 reasoning_content，虽然符合 dsv4 / dsv32 的官方多轮契约，但存量部署若依赖旧拼接行为需要回归；实机验证仅覆盖 Kimi-K3，其余编码器只有单测保障。

2. 静默丢弃：若未来某编码器不读 reasoning_content，历史思考会被静默丢弃而非报错，用户难以察觉——这是设计上有意取舍（丢弃优于泄漏），但在可观测性上留了缺口。
3. 守卫测试脆弱：test_all_chat_encoding_specs_are_enumerated 用正则解析源码 return 字面量，若未来重构改变返回写法（如改为变量间接返回）会漏检或误检。
4. 序列化依赖：字段透传依赖 ChatCompletionMessageGenericParam 已声明 reasoning_content 且消息 dump 不排除 None，未来序列化策略变更会导致字段静默丢失。
5. CI 状态：PR body 显示 Extra 与 AMD ROCm 7.2 两个 CI 任务失败，材料中未见失败原因说明，合并前未解决；建议关注后续 nightly 是否覆盖 Anthropic 多轮路径。
6. 回归面整体可控：_convert_assistant_thinking_blocks 返回类型变更影响所有经 /v1/messages 携带 thinking 历史的请求，但非自定义编码器分支逻辑等价，OpenAI 端点与 Jinja 模板模型行为不变。 - 影响：对用户：Kimi-K3 经 Anthropic /v1/messages 的多轮工具对话不再出现 XTML 标记泄漏，工具循环能够收敛（12 轮从每轮泄漏变为第 9 轮收敛，并发 6 会话全部转好）；DeepSeek-V4 / V3.2 的多轮思考历史也切换到官方 reasoning_content 契约。对系统：无 API 或 schema 破坏性变更，OpenAI /v1/chat/completions 逻辑不变，Jinja 模板模型行为保持原样，回归面集中在自定义编码器 + Anthropic 适配器这一条路径。对团队：确立了"自定义编码器拥有推理历史渲染权"的契约与安全默认，守卫测试把"新增编码器必须分类"变成 CI 硬约束，该测试手法（源码字面量解析 + mutation-check）可复用到同类能力探测。 - 风险标记：核心服务路径变更，影响全部自定义编码器，静默丢弃历史思考的取舍，正则守卫测试较脆弱，Extra / AMD CI 失败未说明

关联脉络

- PR #35508 [NPU] [DOC] Add Ascend NPU (A3) recipe to the Kimi-K3 cookbook: 同类模型线（Kimi-K3）：该 PR 完善了 Kimi-K3 的部署 recipe，而本 PR 修复了 Kimi-K3 在 Anthropic 协议下多轮思考历史泄漏，两者共同构成 Kimi-K3 端到端可用性支撑。
- PR #35854 [AMD] Update amd deepseek v4 cookbook 0822: DeepSeek-V4 同样走 dsv4 自定义聊天编码器路径，本 PR 使 dsv4 / dsv32 的多轮 Anthropic 思考历史从 wrap 拼接切换到官方 reasoning_content 契约，与 V4 部署演进相关。