PR #33025 完整报告

sgl-project/sglang

[Kimi K3] Add reasoning, tool-call, and OpenAI serving support

合并时间：2026-08-02 05:57

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33025

执行摘要

- 一句话：Kimi K3 推理与工具调用解析及 OpenAI 服务支持
- 推荐动作：值得精读。建议关注 4 个设计点：(1) kimik3_structural_tag.py 的 JSON Schema → XTML 双格式翻译与 OrFormat 变体构造；(2) KimiK3Detector（function_call 与 reasoning_parser 两个）的流式状态机与 buffer / 游标重置契约；(3) TokenSequenceMatcher（KMP 前缀函数）在 grammar 回滚与多 token marker 匹配中的用途；(4) #33192 的约束失败降级策略——"不强制 JSON-only schema" 的取舍。若要接入新的专用 token 格式模型，这是当前最完整的参考实现。

功能与动机

PR body 明确说明拆分动机："This extracts the parser and directly related OpenAI serving work from #32541 so the model-support PR has a smaller review surface. The parser, constraint, auto-detection, Chat Completions, and Responses pieces are presented as one pull request because they share the same Kimi K3 wire format and need end-to-end validation together." 需要解决的三个问题：(1) Kimi K3 使用 XTML 结构化通道（think / response / tools），每个 marker 都是多 token 特殊序列，现有 parser 无法处理；(2) xgrammar 结构约束必须贴合 K3 原生编码，否则强制 JSON-only schema 时模型输出无法被解析器读回；(3) 字面 <|kimi_image_placeholder|> 文本会被上游渲染器当图像槽位，text-only 场景直接 400（issue #32960），PR 同时并入 #33003 的占位符中和修复。

实现拆解

1. wire 格式常量层（新增 python/sglang/srt/function_call/kimik3_format.py）：定义 THINK_OPEN / THINK_CLOSE / RESPONSE_OPEN / RESPONSE_CLOSE / TOOLS_OPEN / TOOLS_CLOSE / MESSAGE_CLOSE 等 XTML marker 常量，并提供 3 个流式工具函数——partial_suffix_len（计算文本尾部与任一marker前缀的最大重叠长度）、strip_partial_marker_suffix（剥离因 max_tokens 截断残留的半截 marker）、strip_response_wrappers（剥掉 response / message 包装）。这是流式解析正确性的地基：任何 marker 都可能跨 chunk 截断。
2. 结构约束层（新增 kimik3_structural_tag.py，+598 行）：把 OpenAI Tool 的 JSON Schema 翻译成 xgrammar StructuralTag。关键路径是 _schema_types（推导 schema 可接受的 JSON 类型集合，处理 $ref / anyOf / oneOf / allOf / const / enum 及关键字启发）、_restrict_schema_type（按单个 JSON 类型收窄 schema，保证各类型变体互斥）、

_argument_value_variants 与 _known_argument_format（为每个类型生成 TagFormat，string 用 qwen_xml 风格、其余用 JSONSchemaFormat，多类型时包成 OrFormat）。对外暴露 get_kimik3_structural_tag 与 get_kimik3_auto_tool_call_structural_tag，支持 auto / required / named 三种 tool choice、parallel_tool_calls 与 SGLANG_TOOL_STRICT_LEVEL。

3. 工具调用解析层（新增 kimik3_detector.py，+234 行）：KimiK3Detector 用正则提取 tools 通道中的 call / argument 块，_parse_attrs 反转义属性（& / "），string 参数保留原文、其余类型 JSON 解码（失败回退字符串）。提供一次性 detect_and_parse 与流式 parse_streaming_increment 两套接口；流式实现依赖 buffer 与 _sent_normal_idx 游标，异常路径按"重置 buffer 必须同步重置游标"的契约处理，否则后续输出会被静默丢弃。supports_structural_tag() 返回 True，parses_required_natively() 返回 False（配合 #33192 降级策略）。
4. 推理解析层（修改 reasoning_parser.py，+188 行）：新增同名 KimiK3Detector（职责为解析 think 通道）并注册进 ReasoningParser.DetectorMap 的 kimi_k3 项。处理 3 个边界：缺省 <|sepl|> 的结束 marker 恢复、response / message 包装剥离、tools 通道原样透传给 tool-call 检测器。配套新增 utils/token_sequence_matcher.py（KMP 前缀函数）并改造 constrained/reasoner_grammar_backend.py：支持预算耗尽时逐步行走完整多 token think-end 序列、回滚后 marker 存活；schedule_batch.py::update_reasoning_tokens 改为等待完整 end 序列出现后才计入 reasoning token。
5. OpenAI 服务层（修改 serving_chat.py，+272/-42）：_prepare_kimi_k3_messages 统一做消息预处理——文本与结构化字段（含 reasoning_content、tool_calls arguments）递归中和 <|kimi_image_placeholder|>、developer 消息改写为 system、system / developer 消息级 tools 透传；_effective_tools 汇总请求级与消息级工具，使校验、约束构建与流式工具处理口径一致；image_prompts 仅在确有图片时设置，避免 [] 触发上游渲染器 scan-and-consume 模式；normalize_assistant_tool_call_arguments 增加 strict 参数，K3 渲染路径允许畸形历史参数原样通过；结构约束构建失败时不强制 JSON-only schema，保留 K3 原生 stop 格式（TOOLS_CLOSE，即 #33192）。serving_responses.py 同步传递 reasoning 状态与 chat encoder 字段，protocol.py 增加 _has_message_level_tools 逻辑。
6. 测试与验证：新增 test_kimik3_structural_tag.py（810 行，用 xgrammar 的 accept-string 校验 grammar 接受 / 拒绝）、test_kimik3_detector.py（221 行，覆盖单 / 多 call、未闭合 tools、流式分块 1 / 7 / 23）、test_kimik3_reasoning_parser.py（162 行，覆盖缺 sep 恢复、部分 marker 剥离、流式 split、tools 透传）；扩展 serving_chat / serving_responses / protocol / reasoner_grammar_backend 测试。PR body 报告 200 + 35 + 436 + 73 用例全绿；message-level tools 校验通过 default / dsv4 / dsv32 / kimi_k3 四种编码的 subtest 覆盖；另含 Python 3.10 兼容修复（f12f3ac）。

关键文件：

- python/sglang/srt/function_call/kimik3_structural_tag.py（模块 结构约束；类别 source；类型 core-logic；符号 _escape_attr, _json_type, _matches_json_type, _resolve_local_ref）：本 PR 核心新增：把 OpenAI Tool 的 JSON Schema 翻译成 Kimi K3 原生 XTML 的 xgrammar StructuralTag，支撑 auto / required / named 三种 tool

choice、per-tool 严格约束与 parallel_tool_calls。

- python/sglang/srt/function_call/kimik3_detector.py（模块 工具解析；类别 source；类型 core-logic；符号 _unescape_attr, _parse_attrs, KimiK3Detector, has_tool_call）：Kimi K3 XTML 工具调用检测与流式解析，承担 tools 通道提取、属性反转义与非 string 参数 JSON 解码，是 serving 层 tool-call 解析的落点。
- python/sglang/srt/entrypoints/openai/serving_chat.py（模块 服务层；类别 source；类型 core-logic；符号 normalize_assistant_tool_call_arguments, neutralize_kimi_k3_image_placeholder, neutralize_kimi_k3_image_placeholder_value, _effective_tools）：OpenAI Chat 服务层集成：消息预处理占位符中和、message-level tools 合并、image_prompts 守卫、约束失败降级与 wire 字段透传，是全模型核心入口。
- python/sglang/srt/parser/reasoning_parser.py（模块 推理解析；类别 source；类型 core-logic；符号 KimiK3Detector, _clean_content, _next_channel_idx, detect_and_parse）：新增推理侧 KimiK3Detector 解析 think 通道并注册 kimi_k3，处理多 token marker 流式 holdback、缺 sep 恢复与工具通道透传。
- python/sglang/srt/constrained/reasoner_grammar_backend.py（模块 语法约束；类别 source；类型 dependency-wiring）：多 token think-end 标记的预算耗尽行走与回滚存活改造，影响所有使用 reasoner grammar 的模型，而非仅 Kimi K3。
- python/sglang/srt/function_call/kimik3_format.py（模块 格式常量；类别 source；类型 core-logic；符号 partial_suffix_len, strip_partial_marker_suffix, strip_response_wrappers）：XTML marker 常量与 partial marker 处理工具，是流式解析正确性的地基，被 detector 与 reasoning parser 共同依赖。
- python/sglang/srt/utils/token_sequence_matcher.py（模块 匹配工具；类别 source；类型 dependency-wiring；符号 TokenSequenceMatcher, _build_prefix_lengths, advance）：KMP 前缀函数实现的多 token 序列匹配器，支撑跨 chunk 识别完整 think-end 标记并在 grammar 回滚中存活。
- test/registered/unit/function_call/test_kimik3_structural_tag.py（模块 约束测试；类别 test；类型 test-coverage；符号 _tool, _argument, _call, _tools_section）：810 行 grammar 接受 / 拒绝测试，覆盖 strict schema、named tool choice、$ref / anyOf / allOf 组合、非严格工具与 loose string。
- test/registered/unit/entrypoints/openai/test_serving_chat.py（模块 服务层测试；类别 test；类型 test-coverage；符号 test_process_messages_records_template_reasoning_state, test_kimi_k3_constraint_failure_keeps_native_stop_format, test_kimi_k3_tool_call_stop_is_scoped_to_active_tools, test_kimi_k3_encoder_receives_wire_request_fields）：服务层行为测试：reasoning 状态记录、约束失败保留原生 stop 格式、wire 字段透传、占位符中和与跨编码 message-level tools 校验。

关键符号：KimiK3Detector(function_call).parse_streaming_increment, KimiK3Detector(function_call).detect_and_parse, KimiK3Detector(function_call)._decode_call, get_kimik3_structural_tag, get_kimik3_auto_tool_call_structural_tag, _schema_types, _restrict_schema_type, _argument_value_variants, _known_argument_format, partial_suffix_len,

strip_partial_marker_suffix, strip_response_wrappers,KimiK3Detector(reasoning_parser).parse_streaming_increment,KimiK3Detector(reasoning_parser)._drain_content, TokenSequenceMatcher.advance,_prepare_kimi_k3_messages, _effective_tools,neutralize_kimi_k3_image_placeholder_value, normalize_assistant_tool_call_arguments,update_reasoning_tokens

关键源码片段

python/sglang/srt/function_call/kimik3_structural_tag.py

本 PR 核心新增：把 OpenAI Tool 的 JSON Schema 翻译成 Kimi K3 原生 XTML 的 xgrammar StructuralTag，支撑 auto / required / named 三种 tool choice、per-tool 严格约束与 parallel_tool_calls。

```
def _argument_value_variants(
    schema: Union[bool, Dict[str, Any]],
    root_schema: Dict[str, Any],
    loose_strings: bool = False,
) -> List[Tuple[str, Format]]:
    # 对 schema 允许的每个 JSON 类型生成一个"类型标签 + 取值格式"变体：
    # K3 的 argument 头里 type 属性是显式的，因此不同 JSON 类型必须拆成
    # 互斥分支，最终由调用方包成 OrFormat。
    return [
        (
            json_type,
            _value_format(restricted, json_type, loose_string=loose_strings),
        )
        for json_type in _schema_types(schema, root_schema)
        if (restricted := _restrict_schema_type(schema, json_type, root_schema))
        is not False
    ]


def _known_argument_format(
    key: str,
    schema: Union[bool, Dict[str, Any]],
    root_schema: Dict[str, Any],
) -> Optional[Format]:
    # 已知参数：为每个 JSON 类型构造一个 TagFormat。begin 直接是 XTML 的
    # <|open|>argument key="..." type="..."<|sep|>，内容格式取决于类型：
    # string 用 qwen_xml 风格（原生 XTML 文本），其余类型用 JSON 风格。
    escaped_key = _escape_attr(key)
    variants = [
        TagFormat(
            begin=(
                f'<|open|>argument key="{escaped_key}" '
                f'type="{_JSON_TO_XTML_TYPE[json_type]}"<|sep|>'
            ),
            content=value_format,
```

```
            end=ARGUMENT_CLOSE,
        )
        for json_type, value_format in _argument_value_variants(schema, root_schema)
    ]
    if not variants:
        return None
    if len(variants) == 1:
        return variants[0]
    return OrFormat(elements=variants)
```

python/sglang/srt/function_call/kimik3_detector.py

Kimi K3 XTML 工具调用检测与流式解析，承担 tools 通道提取、属性反转义与非 string 参数 JSON 解码，是 serving 层 tool-call 解析的落点。

```
def parse_streaming_increment(self, new_text: str, tools: List[Tool]) -> StreamingParseResult:
    # 流式解析入口：先把新 chunk 追加进内部 buffer，再尝试从中提取完整
    # tool call。XTML 的 marker 是多 token 特殊序列，可能跨 chunk 截断，
    # 因此未闭合部分必须留在 buffer 里继续累积。
    self._buffer += new_text
    try:
        open_idx = self._buffer.find(self.bot_token)
        if open_idx == -1:
            # 尚未出现 tools 通道开头，按普通文本输出；_emit_normal_text
            # 内部用 partial_suffix_len 做 holdback，避免把 marker 前缀
            # 拦腰吐给客户端。
            return StreamingParseResult(normal_text=self._emit_normal_text())

        normal_text = self._emit_normal_text(limit=open_idx)
        section = self._buffer[open_idx + len(self.bot_token) :]
        calls = []
        parsed = self._parse_calls(section)
        for call in parsed[self.current_tool_id + 1 :]:
            self.current_tool_id += 1
            while len(self.prev_tool_call_arr) <= self.current_tool_id:
                self.prev_tool_call_arr.append({})
            while len(self.streamed_args_for_tool) <= self.current_tool_id:
                self.streamed_args_for_tool.append('')
            self.prev_tool_call_arr[self.current_tool_id] = {
                'name': call['name'],
                'arguments': json.loads(call['arguments']),
            }
            self.streamed_args_for_tool[self.current_tool_id] = call['arguments']
            calls.append(
                ToolCallItem(
                    tool_index=self.current_tool_id,
                    name=call['name'],
                    parameters=call['arguments'],
                )
            )
```

```
            return StreamingParseResult(normal_text=normal_text, calls=calls)
        except Exception as e:
            logger.error(
                'Error in Kimi K3 parse_streaming_increment: %s', e, exc_info=True
            )
            # _sent_normal_idx 索引指向 _buffer，因此必须与 buffer 一起重置；
            # 否则后续每次 _emit_normal_text 都会因 limit <= _sent_normal_idx
            # 而静默丢弃响应剩余部分。
            self._buffer = ''
            self._sent_normal_idx = 0
            return StreamingParseResult()
```

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

OpenAI Chat 服务层集成：消息预处理占位符中和、message-level tools 合并、image_prompts 守卫、约束失败降级与 wire 字段透传，是全模型核心入口。

```
    def neutralize_kimi_k3_image_placeholder_value(value: Any) -> Any:
        # 递归中和字符串、列表、字典中的字面占位符。上游 encoding_k3.py 渲染器
        # 会对文本执行占位符扫描（scan-and-consume），用户消息里出现
        # "<|kimi_image_placeholder|>" 会被误当成图片槽位（issue #32960）；
        # 替换成带空格的变体后，渲染器把它当作惰性纯文本处理。
        if isinstance(value, str):
            return neutralize_kimi_k3_image_placeholder(value)
        if isinstance(value, list):
            return [neutralize_kimi_k3_image_placeholder_value(item) for item in value]
        if isinstance(value, dict):
            return {
                key: neutralize_kimi_k3_image_placeholder_value(item)
                for key, item in value.items()
            }
        return value


    def _effective_tools(self, request: ChatCompletionRequest) -> List[Tool]:
        # 汇总请求级 tools 与 system / developer 消息上的级联 tools，让消息级
        # 工具参与校验、约束构建与流式工具调用处理；这改变了默认编码的校验
        # 口径，测试覆盖 default / dsv4 / dsv32 / kimi_k3 四种编码。
        tools = list(request.tools or [])
        for message in request.messages:
            if (
                isinstance(message, ChatCompletionMessageGenericParam)
                and message.role in ('system', 'developer')
                and message.tools
            ):
                tools.extend(message.tools)
        return tools
```

评论区精华

核心交锋集中在 issue #32960 的占位符 400 排查线程：

- tancheng33 定位到根因：serving_chat.py 无条件执行 template_kwargs['image_prompts'] = ['<|media_pad|>'] * image_count，当 image_count == 0 时传入 []，使上游 encoding_k3.py 渲染器进入 scan-and-consume 模式并立即抛 ValueError；而 image_prompts=None 会把占位符渲染为惰性纯文本。
- hnyls2002 确认该修复来自 #33003 并已落地本 PR。
- tancheng33 进一步指出残留路径：上游渲染器还会对 assistant tool_calls 参数（json_block 与 _xml_value 分支）以及 reasoning_content 做同样的占位符扫描，这两条路径未经过 neutralize，text-only agentic 对话一旦出现引用字面占位符的工具参数，后续每一轮请求都会 400；建议补 if image_count: 守卫。
- tancheng33 最终验证 #33183 覆盖了全部残留路径：渲染前统一中和 reasoning / 嵌套工具参数，image_prompts 重建时加 if image_count: 守卫并丢弃过期用户值。

另一项设计决策来自 #33192（作为本 PR 最后一个提交并入）：KimiK3Detector 的 parses_required_natively() 返回 False，但最终策略是让模型原生 parser 处理 required tool choice；当结构约束构建失败时，保持 K3 输出不受限（原生格式）而不是退回 JSON-only schema——因为 K3 的解析器读不回 JSON-only 输出。

- image_prompts=[] 触发上游渲染器 scan-and-consume 导致 400（issue #32960）(correctness): hnyls2002 确认 #33003 的修复已落地；tancheng33 验证 #33183 以 if image_count: 守卫 + 丢弃过期用户值方式闭合全部路径。
- 占位符中和的残留路径：assistant tool_calls 参数与 reasoning_content (correctness): 提交 5f4f76c 先尝试修复后被 revert，最终经 #33183 以“渲染前统一中和 reasoning / 嵌套工具参数”方式落地。
- 约束失败时的降级策略（#33192） (design): 以提交 af3e2d6（#33192）并入，配套测试 test_kimi_k3_constraint_failure_keeps_native_stop_format 覆盖。
- 流式解析错误路径的 buffer 与游标重置契约 (correctness): 以提交与注释形式固化，流式分块测试 test_streaming_split_markers 覆盖 1 / 7 / 23 字节分块。
- strict-thinking 下 think_excluded_tokens 的选词约束 (design): 以注释形式固化在 KimiK3Detector(reasoning_parser) 的实现中，测试 test_kimi_k3_excluded_tokens_spare_the_xtml_control_tokens 覆盖。

风险与影响

- 风险：
  - 全模型 serving 入口变更：serving_chat.py 是 /v1/chat/completions 核心路径，normalize_assistant_tool_call_arguments 新增 strict 参数、_effective_tools 把消息级工具并入校验与流式处理，默认编码路径的校验口径变化会影响所有模型的服务行为（测试覆盖了 4 种编码，但真实模型组合仍有暴露差异的可能）。
  - 语法后端跨模型影响：reasoner_grammar_backend.py 的多 token marker 行走与回滚改造服务于所有 reasoner grammar 模型，不只是 Kimi K3；配套测试覆盖了 multi-token end marker 用例，但推理预算耗尽、自重叠 marker 等极端路径仍需生产验证。

- 流式解析状态机风险：KimiK3Detector.parse_streaming_increment 的异常路径重置 buffer 与游标，语义是“丢弃未吐出的内容”，调用方必须依赖该契约；正则 _CALL_RE / _ARG_RE 对参数正文中出现 <|tool_call_sep|> 原始序列的场景没有显式防护。
- 复杂 JSON Schema 翻译边界：_schema_types / _restrict_schema_type 对 allOf / anyOf / $ref 组合使用集合运算收窄（含 number / integer 互通的特殊处理），逻辑自洽但极端嵌套 schema 仍可能有类型枚举与关键字启发不一致的边界。
- reasoning token 计数口径：schedule_batch.py::update_reasoning_tokens 改为等待完整多 token think-end 序列后计数，影响 usage 统计口径，需与前端计费逻辑核对。

- 影响：
  - 用户 / 模型侧：Kimi K3 用户可开箱使用推理内容（reasoning_content）与原生工具调用，支持 parallel_tool_calls、per-tool 严格 schema、SGLANG_TOOL_STRICT_LEVEL、消息级工具与 named tool choice；字面 <|kimi_image_placeholder|> 不再被误判为图像输入（#32960）。
  - 系统侧：新增 4 个模块（kimik3_format / kimik3_structural_tag / kimik3_detector / token_sequence_matcher），改动 5 个高复用入口（serving_chat、serving_responses、reasoning_parser、reasoner_grammar_backend、schedule_batch）；ReasoningParser.DetectorMap 与 chat_encoding_spec 的自动检测机制降低了后续模型接入成本。
  - 团队侧：形成“wire 常量 + schema 翻译 + 检测器 + 推理解析 + serving 接线”的完整接入范式，后续新专用 token 格式模型可复制该结构。
  - 风险标记：核心服务路径变更，跨模型语法后端影响，流式解析状态机风险，正则解析边界，复杂 schema 翻译边界，reasoning token 计数口径变化

关联脉络

- PR #33192 [Kimi K3] Keep native tool-call format when constraints fail: 作为本 PR 最后一个提交（af3e2d6）并入，解决结构约束构建失败时误强制 JSON-only schema 导致解析器读不回的问题。
- PR #33183 Harden Kimi K3 parser edge cases: 作为提交 6658b91 并入（co-authored 与 A-transformer），覆盖占位符中和残留路径与 image_prompts 守卫，闭合 issue #32960。
- PR #33003 Neutralize literal kimi_image_placeholder text（标题据 PR body 描述补全）：PR body 明确说明其占位符中和修复已并入本 PR，是 #32960 的根因修复。
- PR #32541 Kimi K3 model support（本 PR 从中拆分 parser 与 serving，标题据 body 描述）：PR body 说明本 PR 从中提取 parser 与 OpenAI serving 工作，以缩小模型支持 PR 的审查面。
- PR #32768 Kimi K3 parser / serving draft（被本 PR 取代，标题据 body 描述）：PR body 声明 Supersedes #32768 及临时堆叠草稿 #33017、#33018、#33019、#33020。
- PR #32828 [Kimi] Support DCP + DSpark (ported from kimi-k3 branch): 同属 Kimi 模型族功能线，且都修改了 speculative / serving 相关链路，可视为 Kimi 系列支持的延续。