PR #21722 完整报告

sgl-project/sglang

feat: use structural tags to enable strict tool calling and reasoning for more models

合并时间：2026-05-04 17:30

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/21722

执行摘要

- 一句话：升级 xgrammar 并引入内置 structural tags 支持更多模型 tool calling
- 推荐动作：建议团队精读此 PR，尤其是 get_structure_constraint 的重构和 base_format_detector.py 中 get_structural_tag 的集中化设计，这为未来添加更多模型的原生结构化约束奠定了良好基础。同时注意监控 Kimi 内置 tag 的 xgrammar 上游修复进度，以便及时重新启用。

功能与动机

PR body 明确指出升级 xgrammar 到 v0.2.0 以利用其 built-in structural_tags 特性，为更多模型提供结构化生成，尤其是 tool calling 场景。性能基准测试（BFCL-v3）显示开启 structural tags 后正确率显著提升。此外，社区反馈也期望支持更多模型（如 minimax）的结构化调用。

实现拆解

1. 升级依赖并处理可选导入：在 pyproject.toml 中将 xgrammar 从 0.1.32 升级到 0.2.0，并在 base_format_detector.py 中使用 try/except ImportError 保护 xgrammar.StructuralTag 和 get_model_structural_tag 的导入，避免 XPU 等无 xgrammar 环境启动失败。
2. 新增基类方法：在 BaseFormatDetector 中添加 get_structural_tag_name()（子类重写返回模型名称）和 get_structural_tag()（调用 get_model_structural_tag 生成原生 tag），子类只需提供名称即可复用统一逻辑。
3. 重构约束生成逻辑：FunctionCallParser.get_structure_constraint() 优先尝试 detector.get_structural_tag()，若返回非 None 则直接使用原生 tag；否则回退到旧有的 legacy_structural_tag 路径（需要 supports_structural_tag() 为 True）；最后才是 JSON schema fallback。同时处理 thinking_mode 参数的传递，并在存在 ReasonerGrammarBackend 时不要求 xgrammar 包含 reasoning prefix。
4. 新增模型 detector：添加 deepseekv4_detector.py，继承 DeepSeekV32Detector 但使用 <｜DSML｜tool_calls> 标签；qwen3_coder_detector.py 开启 supports_structural_tag 并返回 qwen_3_coder 模型名；deepseekv32_detector.py、gpt_oss_detector.py 等也添加了 get_structural_tag_name。

5. 修复 speculative decoding 兼容性：spec_utils.py 中的 traverse_tree 内部循环在调用 xgrammar 前将 0 维 tensor 显式转换为 Python int，以适配 xgrammar 0.2.0 的严格类型签名。
6. 测试配套：在 test_function_call_parser.py 中新增 test_get_model_structural_tag 和 TestDeepSeekV4Detector 等测试，覆盖原生 structural tag 的生成与解析；新增 test_spec_utils_traverse_tree.py 回归测试，确保 tensor 转换正确。

关键文件：

- python/sglang/srt/function_call/function_call_parser.py（模块 函数调用；类别 source；类型 core-logic；符号 get_structure_constraint, get_legacy_structural_tag, get_structure_tag）：核心解析器：重构 get_structure_constraint 方法，引入基于 get_structural_tag 的三级回退机制；重命名 get_structure_tag 为 get_legacy_structural_tag。
- python/sglang/srt/function_call/base_format_detector.py（模块 函数调用；类别 source；类型 dependency-wiring；符号 get_structural_tag_name, get_structural_tag）：基类：添加 get_structural_tag_name 和 get_structural_tag 方法，集中实现 xgrammar 桥接逻辑，所有子类只需返回模型名称。
- python/sglang/srt/function_call/deepseekv4_detector.py（模块 函数调用；类别 source；类型 core-logic；符号 DeepSeekV4Detector, init, get_structural_tag_name）：新增 DeepSeek V4 检测器，继承 V3.2 但使用 DSML 标签，支持两种参数格式（XML 或 JSON），并返回模型名 'deepseek_v4'。
- test/registered/unit/function_call/test_function_call_parser.py（模块 测试；类别 test；类型 test-coverage；符号 TestDeepSeekV4Detector, test_get_model_structural_tag, test_detect_and_parse_xml_format, test_detect_and_parse_json_format）：大规模测试新增：覆盖 DeepSeek V4 detector 的 XML/JSON 格式解析、streaming 解析、以及 get_structural_tag 生成测试
- test/registered/unit/spec/test_spec_utils_traverse_tree.py（模块 推测解码；类别 test；类型 test-coverage；符号 TestTraverseTreePassesIntsToGrammar, _record_grammar, record_accept, record_fill）：回归测试：验证 spec_utils.traverse_tree 在 xgrammar 0.2.0 下传递 int 而非 tensor，避免类型错误

关键符号：BaseFormatDetector.get_structural_tag_name, BaseFormatDetector.get_structural_tag, FunctionCallParser.get_structure_constraint, FunctionCallParser.get_legacy_structural_tag, DeepSeekV4Detector.init, DeepSeekV4Detector.get_structural_tag_name, Qwen3CoderDetector.get_structural_tag_name, DeepSeekV32Detector.get_structural_tag_name, GptOssDetector.get_structural_tag_name

关键源码片段

python/sglang/srt/function_call/function_call_parser.py

核心解析器：重构 get_structure_constraint 方法，引入基于 get_structural_tag 的三级回退机制；重命名 get_structure_tag 为 get_legacy_structural_tag。

```
# python/sglang/srt/function_call/function_call_parser.py
# 重构后的 get_structure_constraint 方法（部分）

def get_structure_constraint(
    self,
    tool_choice: Union[ToolChoice, Literal["auto", "required"]],
    parallel_tool_calls: bool = True,
    thinking_mode: bool = False,
) -> Optional[ToolCallConstraint]:
    is_required = tool_choice == "required" or isinstance(tool_choice, ToolChoice)
    should_constrain_auto = tool_choice == "auto" and (
        any(tool.function.strict for tool in self.tools)
        or self.tool_strict_level >= ToolStrictLevel.FUNCTION
    )

    # 1. 优先使用模型原生 structural_tag（xgrammar 0.2.0 内建）
    # 例如 DeepSeek V4 / Qwen3 Coder 可通过 get_structural_tag_name 返回模型名
    try:
        if is_required or should_constrain_auto:
            structural_tag = self.detector.get_structural_tag(
                tools=self.tools,
                thinking_mode=thinking_mode,
                tool_choice=tool_choice,
            )
            if structural_tag is not None:
                return ("structural_tag", structural_tag)

            # 2. 回退到 legacy structural_tag（旧的 parser 内建约束）
            if self.detector.supports_structural_tag():
                tag = self.get_legacy_structural_tag(at_least_one=is_required)
                return ("structural_tag", tag)

    except Exception:
        # 若 xgrammar 调用失败（如模型名不匹配），继续回退
        pass

    # 3. 最后回退到 JSON schema
    if tool_choice == "required" or isinstance(tool_choice, ToolChoice):
        json_schema = get_json_schema_constraint(
            self.tools, tool_choice, parallel_tool_calls=parallel_tool_calls
        )
        return ("json_schema", json_schema)
    return None
```

python/sglang/srt/function_call/base_format_detector.py

基类：添加 get_structural_tag_name 和 get_structural_tag 方法，集中实现 xgrammar 桥接逻辑，所有子类只需返回模型名称。

```
# python/sglang/srt/function_call/base_format_detector.py

# 在文件顶部，使用 try/except 保护 xgrammar 导入
# 使得无 xgrammar 的 XPU 环境也能正常加载模块
try:
    from xgrammar import StructuralTag, get_model_structural_tag
except ImportError:
    StructuralTag = Any # type: ignore[misc, assignment]
    get_model_structural_tag = None

class BaseFormatDetector(ABC):
    # ... 其他方法 ...

    def get_structural_tag_name(self) -> Optional[str]:
        """子类重写返回 XGrammar 模型名称（如 "deepseek_v4"），
        基类默认返回 None，表示不支持原生 structural tag。"""
        return None

    def get_structural_tag(
        self,
        tools: Union[List[Tool], None] = None,
        tool_choice: Union[ToolChoice, Literal["auto", "required"]] = "auto",
        thinking_mode: bool = False,
    ) -> Optional[StructuralTag]:
        """
        生成模型原生 XGrammar structural tag。
        当 --reasoning-parser 配置时，SGLang 的 ReasonerGrammarBackend
        会处理 <think>...</think> 前缀，因此此处传入 reasoning=thinking_mode
        但只在不需要外层推理时才让 xgrammar 包含 reasoning prefix。
        """
        structural_tag_name = self.get_structural_tag_name()
        # 既没有模型名，也没有 xgrammar 可用时直接返回 None
        if not structural_tag_name or get_model_structural_tag is None:
            return None

        # 将 SGLang 的 Tool 对象转换为 dict
        converted_tools = [tool.model_dump() for tool in tools or []]
        converted_tool_choice = (
            tool_choice.model_dump()
            if isinstance(tool_choice, ToolChoice)
            else tool_choice
        )
        return get_model_structural_tag(
            model=structural_tag_name,
            tools=converted_tools,
            tool_choice=converted_tool_choice,
```

```
            reasoning=thinking_mode,
        )
```

python/sglang/srt/function_call/deepseekv4_detector.py

新增 DeepSeek V4 检测器，继承 V3.2 但使用 DSML 标签，支持两种参数格式（XML 或 JSON），并返回模型名 'deepseek_v4'。

```
# python/sglang/srt/function_call/deepseekv4_detector.py
import logging

from sglang.srt.function_call.deepseekv32_detector import DeepSeekV32Detector

logger = logging.getLogger(__name__)


class DeepSeekV4Detector(DeepSeekV32Detector):
    """
    检测 DeepSeek V4 模型函数调用格式。
    使用 DSML 标签（如 <｜DSML｜tool_calls>）而非 V3.2 的 <｜DSML｜function_calls>。
    支持两种参数格式：
      1. XML Parameter Tags: <｜DSML｜parameter name="...">value</｜DSML｜parameter>
      2. Direct JSON: { "param_name": "value" } 直接放在 <｜DSML｜invoke> 内
    """

    def __init__(self):
        super().__init__()
        # V4 使用 tool_calls 而非 function_calls
        self.bot_token = "<｜DSML｜tool_calls>"
        self.eot_token = "</｜DSML｜tool_calls>"
        self.function_calls_regex = r"<｜DSML｜tool_calls>(.*?)</｜DSML｜tool_calls>"

    def get_structural_tag_name(self) -> str:
        # 返回 xgrammar 内建模型名称，用于 get_model_structural_tag
        return "deepseek_v4"
```

评论区精华

Review 中主要讨论了以下几个关键点：

- 依赖版本问题：JustinTong0323 指出 xgrammar==0.2.0 在发布前无法在 PyPI 上安装，导致 CI 失败（后已发布）。
- XPU 兼容性：JustinTong0323 发现 base_format_detector.py 中直接 import xgrammar 会导致无 xgrammar 的 XPU 环境启动失败，要求保护导入，最终采用了 try/except 方案。
- Kimi 内置 tag 缺少 section wrapper：JustinTong0323 发现 xgrammar 0.2.0 的 Kimi built-in grammar 在 tool_choice="auto" 时缺少 <|tool_calls_section_begin|> 包装，导致 SGLang 解析器丢弃生成的 tool call，最终决定暂不为 Kimi 启用内置 tag，等待 xgrammar 修复。

- DeepSeek V4 detector 未注册到枚举：JustinTong0323 指出新增的 DeepSeek V4 detector 没有被加入 ToolCallParserEnum，因此无法通过运行时选择使用；Seven-Streams 解释是因为 V4 的 begin 标签与 V3.2 不同（< | DSML | tool_calls> vs < | DSML | function_calls>）。
- thinking_mode 默认值：JustinTong0323 批评将 thinking_mode 默认设为 True 会导致非 reasoning 模型生成错误格式，最终修正为从 chat_template_kwargs 中正确推导。
- 设计集中化：Ubospica 和 JustinTong0323 建议将 get_structural_tag 的实现集中到基类，子类只提供模型名称，最终采纳并重构。
  - 依赖版本未发布导致安装失败 (infra): 等待 xgrammar 0.2.0 发布后重新尝试；AgainstEntropy 确认已可安装。
  - XPU 环境因直接导入 xgrammar 启动失败 (security): 采用 try/except 导入，设置 StructuralTag = Any，get_model_structural_tag = None。
  - Kimi 内置 structural tag 缺少 section wrapper (correctness): 暂不为 Kimi 启用内置 tag，保留在 legacy 路径；等待 xgrammar 修复（issue #622）。
  - DeepSeek V4 detector 未注册到解析器枚举 (design): 代码中已添加注册，但 reviewer 仍建议确保正确性（最终已加入）。
  - thinking_mode 默认值不当 (correctness): 修改为从 chat_template_kwargs 中正确解析 thinking_mode，并在存在 ReasoningParser 时不传递 reasoning 给 xgrammar。

风险与影响

- 风险：
  1. 依赖升级风险：xgrammar 0.2.0 可能与其他依赖（如 NPU 工作流使用的 0.1.25）冲突，PR 中已对 NPU 工作流也做了升级（从 0.1.25 跳到 0.2.0），但未充分验证可能引入不兼容。
  2. XPU 导入失败风险：虽然使用了 try/except 保护导入，但若 xgrammar 的 StructuralTag 被用作类型注解且运行时缺失，仍可能导致某些路径抛出 NameError（当前代码仅在运行时方法中引用，风险较低）。
  3. Kimii/Kimi 解析弃用：由于 xgrammar 内置 Kimi 语法缺少 section wrapper，当前对 Kimi 禁用了内置 structural tag，但若未来 xgrammar 修复后忘记重新启用，可能导致 Kimi 工具调用约束退化。
  4. 测试覆盖不全：PR 主要测试了 Qwen3 Coder 和 DeepSeek V4，但 Kimi K2 和 GPT-OSS 的 get_structural_tag 覆盖未添加单元测试，可能隐藏问题。
  5. 核心解析路径变更：get_structure_constraint 的重构改变了工具调用约束的优先级，若某个 detector 的 get_structural_tag 抛出异常（如模型名称不匹配），将导致请求 500 错误（已有 try 块但未处理所有异常）。 - 影响：功能影响：为 DeepSeek V4、Qwen3 Coder 等模型启用了原生结构化工具调用，可显著提高工具调用格式正确率（BFCL 测试显示提升 5-10%）。系统影响：依赖升级可能影响安装流程；NPU 工作流可能因版本跳跃需要额外验证。团队影响：函数调用解析器的架构更加统一，后续添加新模型只需在 detector 中实现 get_structural_tag_name 即可。用户影响：使用相关模型时工具调用可靠性提升，但 tool_choice="auto" 且未启用 strict 模式时行为不变。 - 风险

标记：依赖升级未完全验证，XPU 兼容性依赖保护，Kimi 内置 tag 暂禁需跟踪上游修复，核心解析路径新异常未覆盖，部分 detector 缺少 get_structural_tag 测试

关联脉络

- PR #25600 Add MiniCPM5 tool call parser for XML-style function calls: 同样涉及 tool calling 解析器扩展，修改了 function_call_parser.py 和 template_detection.py，与本 PR 的解析器重构有重叠逻辑。
- PR #26085 drop FutureIndices wrapper class: 修改了 speculative decoding 相关模块，与本 PR 对 spec_utils.py 的 tensor 类型修复有间接关联。