

PR #38890 完整报告

sgl-project/sglang

Support non-strict GLM47 tool calls with EBNF constraints

合并时间: 2026-09-15 16:41

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38890>

执行摘要

- 一句话: 为 GLM47 非 strict 工具调用新增 EBNF 约束
- 推荐动作: 值得精读, 但重点不在体量而在几处设计取舍: ① glm4_moe_detector.py 的 `_glm_has_complete_properties` 递归完备性判定与 `_glm_cached_string_exclude` 的 Aho-Corasick → EBNF 状态编译, 是判断「哪些 schema 值得收紧」的通用范式; ② `function_call_parser.get_structure_constraint` 的新分支展示了「约束类型 + 优先级 + 原生解析声明」三件事如何在一次调用里同时决策; ③ `protocol.py` 把冲突从 `raise` 降级为 `warning` 的理由 (默认注入 vs 用户显式), 是 API 兼容性的典型处理方式。建议阅读时同步核对三件事: `_effective_tools` 与 `request.tools` 的差异是否会破坏 strict 判定、`enable_thinking` 默认值是否与 chat template 实际行为一致、`tool_choice="none"` 也注入文法是否符合产品预期。测试文件中的 `TestGlm47FullAssistantGrammar` 值得作为文法回归的参考实现。

功能与动机

GLM-4.7 / GLM-5 的 XML 风格工具调用在非 strict 场景下此前没有任何解码约束, 模型可能产生结构错误的 / 输出。PR body 明确目标: 「Add EBNF constrained decoding for GLM47 requests without strict tools. The grammar covers reasoning, text, and XML tool-call structure, including tool names, argument names, and basic value types.」同时给出三条边界: 含 `strict=true` 的请求继续走原有约束路径 (「preserving each tool's strict setting」); 显式输出格式约束优先于默认 EBNF; 生成的文法已包含推理段, 因此要绕过 `reasoner` 的推理文法包装。作者也诚实标注了能力上限: 「It does not enforce nested JSON Schema requirements or guarantee required argument presence.」

实现拆解

步骤 1: 入口层识别 GLM47 非 strict 请求 (`serving_chat.py`) `_process_messages` 中新增 `glm_constraint = self.tool_call_parser == "glm47" and not any(tool.function.strict for tool in effective_tools)`。命中时提前用 `FunctionCallParser(request.tools or [], ...)` 构造约束, `thinking_mode` 取 `chat_template_kwargs.enable_thinking` (缺省按 `True` 处理); 随后把 `required_parsed_natively` 直接置为 `glm_constraint`, 并用 `if self.tool_call_parser and not glm_constraint` 跳过原有 `parser` 分支, 从而避免同一请求叠加两套约束。这样改的原因是: 该 EBNF 覆盖整段 `assistant` 输出, 必须在 `chat template` 渲染之前就确定, 且不能与后续 `json_schema` 回退路径重复施加。

步骤 2: 检测器开关与约束构造 (glm47_moe_detector.py / function_call_parser.py)

Glm47MoeDetector.__init__ 新增 use_full_assistant_constraint = False, FunctionCallParser.__init__ 在拿到 detector 后按 not any(tool.function.strict for tool in tools) 赋值; 新增 parses_required_natively() 返回该标志, 作为「required 已由文法原生化处理」的信号。get_structure_constraint 在新分支中构造 functions 列表 (命名 ToolChoice 只保留被点名的 function, tool_choice == "none" 或 tools 为空时为 None), 并调用 generate_glm_grammar(enable_thinking=..., functions=..., special_tokens=GlmSpecialTokenConfig(), chat_template_version="glm47", accommodate_chat_template=True, allow_multiple_assistant_turns=False, required=is_required, parallel_tool_calls=parallel_tool_calls), 返回 ("full_assistant_ebnf", grammar)。

步骤 3: EBNF 生成内核 (glm4_moe_detector.py, 新增 495 行) 这是本 PR 的技术主体, 包含四组能力: ① JSON 值文法 _GLM_XML_GRAMMAR_RULES (basic_string / basic_number / basic_array / basic_object / basic_boolean / basic_null) 与 _GLM_TYPE_MAPPING; ② schema 到文法的映射 _glm_get_value_rule / _glm_handle_enum / _glm_handle_type, 非字符串 enum 按 JSON 类型序列化, 布尔 schema (True/False) 单独容错; ③ 完备性判定 _glm_has_complete_properties, 遇到 \$ref、\$dynamicRef、patternProperties、dependentSchemas、if/then/else、非 False 的 additionalProperties 等关键字即判定不可完整表达, 从而回退为宽松参数文法; ④ 负向串排除工具链 _GlmTrieNode + _glm_build_trie_with_failure_links + _glm_get_transition + _glm_cached_string_exclude, 用 Aho-Corasick 状态机生成「不含 specialtoken/禁用子串」的字符类规则, 并用 lru_cache(maxsize=32) 缓存; 配套 _glm_escape_char_class / _glm_escape_string / _glm_escape_ebnf_string / _glm_hash_name 处理客户端可控的 property 名、function 名与字面量转义。

步骤 4: 约束落地、优先级与缓存隔离 (protocol.py / sampling_params.py / grammar_manager.py / reasoner_grammar_backend.py / sampling.rs)

ToolCallConstraint 联合类型新增 ("ebnf", str) 与 ("full_assistant_ebnf", str); to_sampling_params 对 full_assistant_ebnf 与既有输出约束冲突时从 raise ValueError 降级为 warning 并保留显式约束 (因为该 EBNF 是服务端默认注入而非用户显式要求), 否则写入 sampling_params["ebnf"] 并置 ebnf_full_assistant = True。SamplingParams 增加同名字段, grammar_manager.process_req_with_grammar 用 "full_assistant_ebnf" 作为缓存 key 类型, 避免与普通 ebnf 文法互相命中; reasoner_grammar_backend._init_value_dispatch 对 ("full_assistant_ebnf", ...) 直接返回内层 grammar, 不再包 ReasonerGrammarObject。Rust 侧 rust/sglang-server/src/message/sampling.rs 以 #[serde(skip_deserializing)] 镜像该字段, 保持与 Python schema 的字段顺序 lockstep, 并注明「客户端可设置就会剥掉自己的推理包装」。

步骤 5: 测试配套 test_function_call_parser.py 新增 TestGlm47FullAssistantGrammar, 用 RAW 词表的 xgrammar GrammarCompiler + GrammarMatcher 做编译与接受性断言, 覆盖 tool_choice/parallel/thinking 组合、enum 与非字符串类型、组合 schema、转义 property 名, 并把原 test_required_tool_choice_falls_back_when_native_tag_is_unavailable 的期望从 json_schema 改为 full_assistant_ebnf (strict tools 仍为 json_schema); test_protocol.py 新增 test_full_assistant_ebnf_preserves_explicit_output_constraints;

test_reasoner_grammar_backend.py 参数化验证 json / ebnf / full_assistant_ebnf 三种 key 的差异化包装行为；test_grammar_manager.py 同步调整缓存 key 断言。

关键文件：

- python/sglang/srt/function_call/glm4_moe_detector.py (模块 函数调用；类别 source；类型 core-logic；符号 _GlmTrieNode, _glm_build_trie_with_failure_links, _glm_get_transition, _glm_cached_string_exclude)：本次改动的技术主体，新增 495 行 EBNF 生成工具链：JSON 值文法、schema 映射、递归完备性判定、Aho-Corasick 负向串排除与转义工具，是 full_assistant_ebnf 的产线所在。
- python/sglang/srt/function_call/function_call_parser.py (模块 函数调用；类别 source；类型 core-logic；符号 FunctionCallParser.init, FunctionCallParser.get_structure_constraint)：约束选择的中枢：新增 Glm47MoeDetector 的 use_full_assistant_constraint 标志赋值，并在 get_structure_constraint 最前面插入 full_assistant_ebnf 分支，决定 functions 过滤、required 与 parallel_tool_calls 如何进入文法。
- python/sglang/srt/entrypoints/openai/protocol.py (模块 接口协议；类别 source；类型 core-logic；符号 ToolCallConstraint, to_sampling_params)：约束类型与落地规则的扩展点：ToolCallConstraint 新增 ebnf / full_assistant_ebnf, to_sampling_params 定义「默认 EBNF 让位于显式输出约束」的优先级，并写入 ebnf_full_assistant 标记位。
- python/sglang/srt/entrypoints/openai/serving_chat.py (模块 对话入口；类别 source；类型 core-logic；符号 _process_messages)：入口层决策点：在 chat template 渲染前识别 glm47 非 strict 请求并提前生成约束，同时短路 required_parsed_natively 与原有 parser 分支，是行为变更最直接的落点。
- python/sglang/srt/function_call/glm47_moe_detector.py (模块 函数调用；类别 source；类型 core-logic；符号 Glm47MoeDetector.init, Glm47MoeDetector.parses_required_natively)：提供 GLM47 侧的开关与原生 required 语义：新增 use_full_assistant_constraint 字段和 parses_required_natively()，是入口层与解析器之间协议的一部分。
- python/sglang/srt/constrained/grammar_manager.py (模块 语法缓存；类别 source；类型 core-logic；符号 process_req_with_grammar)：缓存键隔离点：ebnf 缓存 key 类型按 ebnf_full_assistant 变为 full_assistant_ebnf，避免语义不同的文法互相命中缓存。
- python/sglang/srt/constrained/reasoner_grammar_backend.py (模块 语法后端；类别 source；类型 core-logic；符号 _init_value_dispatch)：推理包装例外的实现处：对 full_assistant_ebnf key 直接返回内层 grammar，避免重复包裹推理状态机。
- python/sglang/srt/constrained/base_grammar_backend.py (模块 语法后端；类别 source；类型 core-logic)：基础语法后端的关键类型分发逻辑随之调整（仅 1 行），与 reasoner backend 的新 key 保持一致。
- python/sglang/srt/sampling/sampling_params.py (模块 采样参数；类别 source；类型 data-contract；符号 ebnf_full_assistant)：新增 ebnf_full_assistant 布尔字段，是从协议层传递到 grammar manager / reasoner backend 的载体。
- rust/sglang-server/src/message/sampling.rs (模块 采样协议；类别 source；类型 data-contract；符号 SamplingParams.ebnf_full_assistant, Default for

SamplingParams)：跨语言 schema 同步：以 skip_deserializing 镜像 Python 新字段，保持 Rust 前端与 Python 采样参数 lockstep。

- test/registered/unit/function_call/test_function_call_parser.py (模块 语法测试；类别 test；类型 test-coverage；符号 TestGlm47FullAssistantGrammar, setUpClass, _compile, _accepts)：核心测试：新增 TestGlm47FullAssistantGrammar，用 xgrammar 编译并断言接受性，覆盖 tool_choice/parallel/thinking 组合、enum 与布尔 schema、组合 schema 回退与转义 property 名。
- test/registered/unit/entrypoints/openai/test_protocol.py (模块 协议测试；类别 test；类型 test-coverage；符号 test_full_assistant_ebnf_preserves_explicit_output_constraints)：覆盖优先级规则：验证 full_assistant_ebnf 与显式 ebnf / response_format 同时出现时，显式约束保留且 ebnf_full_assistant 不被置位。
- test/registered/unit/constrained/test_reasoner_grammar_backend.py (模块 后端测试；类别 test；类型 test-coverage；符号 test_wraps_inner_grammar_with_reasoning_state_machine)：验证 reasoner 包装的差异化行为：json 与普通 ebnf 仍被 ReasonerGrammarObject 包裹，full_assistant_ebnf 直接返回内层 grammar。
- test/registered/unit/constrained/test_grammar_manager.py (模块 缓存测试；类别 test；类型 test-coverage)：同步缓存 key 类型变更的断言，保障 full_assistant_ebnf 与普通 ebnf 不会共用缓存条目。

关键符号：generate_glm_grammar, FunctionCallParser.get_structure_constraint, FunctionCallParser.init, Glm47MoeDetector.parses_required_natively, _glm_cached_string_exclude, _glm_build_trie_with_failure_links, _glm_has_complete_properties, ChatCompletionRequest.to_sampling_params, GrammarManager.process_req_with_grammar, ReasonerGrammarBackend._init_value_dispatch, ServingChat._process_messages

关键源码片段

python/sglang/srt/function_call/glm4_moe_detector.py

本次改动的技术主体，新增 495 行 EBNF 生成工具链：JSON 值文法、schema 映射、递归完备性判定、Aho-Corasick 负向串排除与转义工具，是 full_assistant_ebnf 的产线所在。

```
# python/sglang/srt/function_call/glm4_moe_detector.py
```

```
class _GlmTrieNode:
    """带 Aho-Corasick fail 指针的 Trie 节点。"""

    def __init__(self, node_id: int):
        self.id = node_id
        self.children: Dict[str, _GlmTrieNode] = {}
        self.is_end = False # root 到本节点是否已构成一条禁用串
        self.fail: _GlmTrieNode = None

    def _glm_get_transition(node, char, root):
```

```
"""沿 fail 指针找下一个状态；找不到就回 root。"""
```

```
current = node
```

```
while True:
```

```
    if char in current.children:
```

```
        return current.children[char]
```

```
    if current == root:
```

```
        return root
```

```
    current = current.fail
```

```
@lru_cache(maxsize=32)
```

```
def _glm_cached_string_exclude(rule_name, negative_strings):
```

```
    """生成「不含任何 negative_strings 子串」的 EBNF 规则组。
```

```
    核心思路：把禁用串建成 Aho-Corasick 自动机，再让每个自动机状态
```

```
    对应一个 EBNF 非终结符——命中禁用串的状态被剔除，字符类分支允许的
```

```
    字符在文法里显式回退到自动机的目标状态，其余落在模式字符集之外的
```

```
    字符一律回到 root。规则名带 schema 摘要前缀，避免不同请求之间符号冲突。
```

```
    """
```

```
    if not negative_strings:
```

```
        return (f"{rule_name} ::= [^]*",) # 无禁用串：任意字符串
```

```
    sorted_strings = sorted(set(s for s in negative_strings if s))
```

```
    if not sorted_strings:
```

```
        return (f"{rule_name} ::= [^]*",)
```

```
    # 用禁用串集合的哈希做状态名前缀，保证同名规则在同一个文法内可复用
```

```
    hash_prefix = hashlib.sha256(
```

```
        "\x00".join(sorted_strings).encode("utf-8")
```

```
    ).hexdigest()[:16]
```

```
    root, all_nodes = _glm_build_trie_with_failure_links(sorted_strings)
```

```
    all_pattern_chars = {c for pattern in sorted_strings for c in pattern}
```

```
    def state_name(node):
```

```
        return f"s_{hash_prefix}_{node.id}"
```

```
    rules = [f"{rule_name} ::= {state_name(root)}"]
```

```
    for node in all_nodes:
```

```
        if node.is_end:
```

```
            continue # 命中禁用串的状态直接不可达
```

```
    excluded_chars, transitions_by_target = [], {}
```

```
    for char in all_pattern_chars:
```

```
        target = _glm_get_transition(node, char, root)
```

```
        if target.is_end:
```

```
            excluded_chars.append(char) # 走这一步就命中禁用串
```

```
        else:
```

```
            transitions_by_target.setdefault(target.id, []).append(char)
```

```
    # 第一种分支：除「危险字符 + 显式转移字符」外的任意字符，回到 root
```

```

all_explicit_chars = set(excluded_chars)
for chars in transitions_by_target.values():
    all_explicit_chars.update(chars)
if all_explicit_chars:
    escaped = _glm_escape_char_class(''.join(sorted(all_explicit_chars)))
    alternatives = [f"^[{escaped}] {state_name(root)}"]
else:
    alternatives = [f"^[ ] {state_name(root)}"]

# 第二种分支：为每个可安全转移的字符生成显式字符串分支
target_by_id = {n.id: n for n in all_nodes}
for target_id in sorted(transitions_by_target):
    target_node = target_by_id[target_id]
    for char in sorted(transitions_by_target[target_id]):
        alternatives.append(
            f"_{glm_escape_string(char)}" {state_name(target_node)}'
        )
    alternatives.append('') # 允许在此结束字符串

rules.append(f"{state_name(node)} ::= {' | '.join(alternatives)}")

return tuple(rules)

```

python/sclang/srt/function_call/function_call_parser.py

约束选择的中枢：新增 Glm47MoeDetector 的 use_full_assistant_constraint 标志赋值，并在 get_structure_constraint 最前面插入 full_assistant_ebnf 分支，决定 functions 过滤、required 与 parallel_tool_calls 如何进入文法。

```

# python/sclang/srt/function_call/function_call_parser.py
from sclang.srt.function_call.glm4_moe_detector import (
    Glm4MoeDetector,
    GlmSpecialTokenConfig,
    generate_glm_grammar,
)

```

```

class FunctionCallParser:

```

```

    def __init__(self, tools: List[Tool], tool_call_parser: str, tokenizer=None):
        detector_class = self.ToolCallParserEnum.get(tool_call_parser)
        if detector_class:
            ...
            detector = detector_class(**kwargs)
        else:
            raise ValueError(f"Unsupported tool_call_parser: {tool_call_parser}")

```

```

# 只要没有任何一个 tool 声明 strict=true, 就为 GLM47 打开
# 「整段 assistant EBNF」路径：由服务端兜底约束，而不是依赖模型原生 structural_tag。
if isinstance(detector, Glm47MoeDetector):
    detector.use_full_assistant_constraint = not any(

```

```

        tool.function.strict for tool in tools
    )
    self.detector = detector
    self.tools = tools
    self.tool_strict_level = envs.SGLANG_TOOL_STRICT_LEVEL.get()

def get_structure_constraint(
    self,
    tool_choice,
    parallel_tool_calls: bool = True,
    thinking_mode: bool = False,
):
    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
    )

    try:
        if (
            isinstance(self.detector, Glm47MoeDetector)
            and self.detector.use_full_assistant_constraint
        ):
            # tool_choice 为 none 或没有 tool 时不注入任何 function,
            # 但文法仍然约束「推理段 + 文本段」, 避免模型自由发散;
            # 命名 ToolChoice 只保留被点名的那一个 function。
            functions = (
                [
                    tool.function
                    for tool in self.tools
                    if not isinstance(tool_choice, ToolChoice)
                    or tool.function.name == tool_choice.function.name
                ]
                if self.tools and tool_choice != "none"
                else None
            )
        return (
            "full_assistant_ebnf",
            generate_glm_grammar(
                enable_thinking=thinking_mode,
                functions=functions,
                special_tokens=GlmSpecialTokenConfig(),
                chat_template_version="glm47",
                accommodate_chat_template=True,
                allow_multiple_assistant_turns=False,
                required=is_required,
                parallel_tool_calls=parallel_tool_calls,
            ),
        )

```

... (未命中时沿用原有 structural_tag / json_schema 回退路径)

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

约束类型与落地规则的扩展点：ToolCallConstraint 新增 ebnf / full_assistant_ebnf, to_sampling_params 定义「默认 EBNF 让位于显式输出约束」的优先级，并写入 ebnf_full_assistant 标记位。

```
# python/sglang/srt/entrypoints/openai/protocol.py
```

```
# 约束类型新增普通 ebnf 与「整段 assistant」 ebnf 两种
```

```
ToolCallConstraint: TypeAlias = Union[
    Tuple[Literal["structural_tag"], StructuralTagResponseFormat],
    Tuple[Literal["json_schema"], Any], # json_schema 允许 dict / str / None
    Tuple[Literal["ebnf"], str],
    Tuple[Literal["full_assistant_ebnf"], str],
]
```

```
class ChatCompletionRequest(BaseModel):
```

```
    def to_sampling_params(self, ...):
```

```
        ...
```

```
        # 收集已经存在的显式输出约束 (regex / ebnf / structural_tag / json_schema)
```

```
        has_existing_constraints = (
            sampling_params.get("regex")
            or sampling_params.get("ebnf")
            or sampling_params.get("structural_tag")
            or sampling_params.get("json_schema")
        )
```

```
        if tool_call_constraint and has_existing_constraints:
```

```
            # full_assistant_ebnf 是服务端为非 strict GLM47 请求「默认注入」的约束，
```

```
            # 因此用户显式给出的响应格式 / regex / ebnf 优先；
```

```
            # 其余情况 (required 或指定 tool) 依旧直接报错，避免两套约束互相打架。
```

```
            if tool_call_constraint[0] != "full_assistant_ebnf" and (
                self.tool_choice == "required"
                or isinstance(self.tool_choice, ToolChoice)
            ):
                raise ValueError(
                    "tool_choice 'required' or a named tool cannot be combined with "
                    "response_format, regex, or ebnf: the tool-call constraint and the "
                    "output constraint cannot both be honored."
                )
```

```
            logger.warning("Constrained decoding is not compatible with tool calls.")
```

```
        elif tool_call_constraint:
```

```
            constraint_type, constraint_value = tool_call_constraint
```

```
            if constraint_type == "structural_tag":
```

```
                sampling_params[constraint_type] = convert_json_schema_to_str(
                    constraint_value.model_dump(by_alias=True)
                )
```

```

elif constraint_type == "json_schema":
    sampling_params[constraint_type] = convert_json_schema_to_str(
        constraint_value
    )
elif constraint_type == "full_assistant_ebnf":
    # 落地为 ebnf，并置内部标记位：
    # grammar_manager 用它隔离缓存键，reasoner backend
    # 用它决定不再包一层推理文法。
    sampling_params["ebnf"] = constraint_value
    sampling_params["ebnf_full_assistant"] = True
else:
    sampling_params[constraint_type] = constraint_value

return sampling_params

```

评论区精华

本 PR 的 review 记录极薄：只有 JustinTong0323 的一条 APPROVED 「LGTM」和 Issue 区的一条 [/tag-and-rerun-ci](#)，inline review comment 为 0。真正的技术交锋体现在合并者直接推送到 head 分支的三个 follow-up commit 上，可以视为「以 commit 代替评论」的评审：

351d18d: thread tool_choice and parallel_tool_calls into grammar construction; serialize non-string enum values by JSON type; tolerate boolean schemas; reuse get_schema_properties for composed schemas; permissive fallback; scope reasoner bypass to generated full_assistant_ebnf constraints only; escape property and function names in EBNF literals; report native required parsing only for the full-assistant path.

378cbb8: The root-only incompleteness check missed \$ref / patternProperties / mixed declarations inside allOf/anyOf/oneOf branches, producing a restrictive argument grammar that rejected valid arguments. Check completeness recursively and fall back to permissive arguments whenever any branch is not fully representable.

q95a238: The Python SamplingParams field added with the full-assistant EBNF work broke test_rust_sampling_schema_stays_in_lockstep; mirror it as an internal skip_deserializing bool after is_normalized.

可以读出三条关键决策：一是 reasoner 绕过必须最小化授权，只对服务端生成的 full_assistant_ebnf 生效，防止客户端自带 ebnf 借道剥离推理约束；二是「宁可放宽、不可收紧」，凡是 schema 不可完整表达就回退宽松文法；三是跨语言采样参数 schema 必须严格同步，用 skip_deserializing 把该字段钉死为流水线内部产物。

- 递归 schema 完备性检查，避免生成过严文法 (correctness): 已修复，并由 test_incomplete_composition_branches_allow_arguments 与 test_complete_compositions_restrict_argument_names 构成双向护栏。
- reasoner 推理文法包装的绕过范围 (design): 按最小授权实现，测试对比 json / ebnf / full_assistant_ebnf 三种 key 的包装差异。

- 显式输出约束优先于默认注入的 EBNF (design): 区分「默认注入」与「用户显式要求」两类约束, 已由 `test_full_assistant_ebnf_preserves_explicit_output_constraints` 覆盖三种组合。
- Python 与 Rust 采样参数 schema 必须 lockstep (design): 已同步并通过序列化测试; 后续任何采样参数字段变更都需双写。
- 非字符串 enum 与布尔 schema 的序列化容错 (correctness): 已修复, 由 `test_enum_json_types_and_boolean_schemas` 的多组用例 (含 null 联合类型与对象 / 数组枚举) 覆盖。
- CI 存在失败项但缺少根因讨论 (testing): 以 Base 通过加单人 LGTM 合并, 额外 CI 失败根因未定位, 需要后续跟踪确认是否与新约束路径或 xgrammar 测试相关。

风险与影响

- 风险: 1) 默认行为变更, 触发条件比看上去宽: `serving_chat` 的 `glm_constraint` 只看 `tool_call_parser == "glm47"` 与 `strict` 声明, 不看 `tool_choice`。对携带 tools 但 `tool_choice="none"` 的请求, 现在会注入「不含 function」的 full_assistant EBNF (测试 `test_tool_choice_and_parallel_calls` 中 `choice="none"` 分支确认这是有意设计), 输出被限制在推理段 + 文本段, 且 `ebnf_full_assistant=True` 会绕过 `reasoner` 包装——与旧行为 (不注入约束) 不同, 需要在使用文档中显式提示。
- 2) 两处 tool 集合口径不一致: `strict` 判定用 `effective_tools`, 而重建 parser 时用 `request.tools`。若两者在 `strict` 语义上出现差异, 可能出现 `glm_constraint=True` 但 `detector.use_full_assistant_constraint=False`, `get_structure_constraint` 落到原路径并可能返回 None; 此时 `required_parsed_natively` 已被置为 True, 会吞掉 `required` 场景的 `get_json_schema_constraint` 回退, 造成约束静默丢失。需确认 `_effective_tools` 的过滤语义 (例如是否受 tool 白名单影响)。
- 3) `thinking_mode` 推导来源单一: 入口层用 `thinking_mode=True if enable_thinking is None else bool(enable_thinking)`, 其中 `enable_thinking` 仅取 `chat_template_kwargs`; 若模板默认值来自 tokenizer 侧或服务端默认 kwargs, 实际不输出推理段而文法要求推理前缀时, 可能被文法直接拒绝或提前截断。
- 4) schema 严格性回归风险 (已修一轮但仍是高危面): `_glm_has_complete_properties` 一旦漏判组合 schema, 表现是「文法过严、拒绝合法参数」而非报错。最后两个 commit 正是为此返工, `test_incomplete_composition_branches_allow_arguments` 与 `test_complete_compositions_restrict_argument_names` 构成双向护栏, 但覆盖面向的是构造出的 schema 形态, 真实客户端的复杂 schema (`$defs` 跨引、深层嵌套 `allOf`) 仍需灰度验证。
- 5) 客户端可控输入进入文法 + 性能: property 名、function 名、enum 值都会编入 EBNF (已通过 `_glm_escape_*` 与 `_glm_hash_name` 做转义和命名空间隔离), 但 Aho-Corasick 状态展开与递归完备性检查均由请求内容驱动, 超大 / 超深 schema 可能带来文法编译开销; `_glm_cached_string_exclude` 只有 32 项缓存, 且不同 tools 集合会产生不同 EBNF, `grammar_manager` 的缓存命中率取决于工具定义稳定性。

6) 跨语言契约脆弱: `sampling.rs` 依赖 `#[serde(skip_deserializing)]` 与字段位置约定, `test_rust_sampling_schema_stays_in_lockstep` 一旦失败即代表两侧 schema 漂移, 后续任何 `SamplingParams` 字段调整都必须双写。

7) CI 未全绿: PR body 显示 PR Test (Extra) 与 AMD ROCm 10 均为失败状态, 仅 Base 通过, 且过程只有一次 `/tag-and-rerun-ci`, 没有失败根因分析记录; 新增的 `xgrammar` 依赖单测与 ROCm 失败是否相关无法从现有材料判断。

- 影响: 用户侧: GLM-4.7 / GLM-5 用户在不声明 `strict` 的情况下调用工具, 将默认获得结构合规性更高的输出 (工具名、参数名、基础类型受文法约束), 代价是 `tool_choice="none"` 场景的输出空间也被收窄; 非 `strict` 语义仍是「浅约束」, 不保证 `required` 参数出现、不强制嵌套 JSON Schema, 用户不应据此假设参数完备。

系统侧: 新增 `ebnf_full_assistant` 采样参数进入跨语言 wire 协议 (Python ↔ Rust 前端), `grammar` 缓存的 key 空间扩大一个类型, `reasoner` 语法后端新增一条「不包装」的例外分支; GLM47 非 `strict` 请求首次进入约束解码路径, 对首 token 延迟与文法编译耗时有新增开销, 但被 `lru_cache` 和 `grammar manager` 缓存部分抵消。

团队侧: `function_call` 与 `constrained` 两个方向的维护者需要共同维护这条新路径 (检测器标志位、约束类型、缓存 key、`reasoner` 例外四处耦合); Rust 前端维护者被动跟随 Python 字段变更; 测试写法 (`xgrammar` 编译 + `accept_string` 断言) 可作为后续文法类改动的模板。

- 风险标记: 默认约束行为变更, `tool` 集合口径不一致, 客户端 schema 编入文法, CI 额外任务未通过, 跨语言字段需 `lockstep`, 递归完备性漏判风险

关联脉络

- PR #39227 Force reasoning mode for GLM-5.3 chat templates: 同属 GLM 系列推理模式与模板行为控制; 本 PR 在入口层用 `chat_template_kwargs.enable_thinking` 推导 `thinking_mode`, 与该 PR 建立的模板侧推理规则直接衔接 (本 PR 分支名也是 `glm53`)。
- PR #38939 [Rust] Use Dynamo native renderers when chat templates are missing: 同属 Rust 前端的 chat 渲染与推理处理链路; 本 PR 继续在 `rust/sglang-server` 的采样参数 schema 上做跨语言字段同步, 属于同一条 Rust 服务演进线的延续。
- PR #39534 [Fix] Aggregate all IPC weight update responses: 同在入口 / 协议层做「多响应或多约束必须统一判定」的收敛式修复思路 (该 PR 归并 IPC 响应, 本 PR 归并约束优先级), 可作为同类 API 行为一致性问题处理方式的对照。