PR #35963 完整报告

sgl-project/sglang

Add Spark3 Model

合并时间：2026-08-26 07:24

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35963

执行摘要

- 一句话：新增 Spark3 模型架构支持与工具调用解析器
- 推荐动作：值得精读。面向模型接入开发者，重点看三处：Spark3Config 的 layer_types 与两套 rope_parameters 默认策略；models/spark3.py 的滑动窗口语义换算与 head-wise 输出门控实现；Spark3Detector 的流式原子输出与类型转换回退。建议在合并后跟进模型 E2E 精度测试与 1M context 压测。

功能与动机

PR body 明确目标是提供 "native support for the Spark 3 model architecture in SGLang"，并列出多项卖点：效率优先的架构（"Sliding Window Attention (SWA) and other structural optimizations"）、"Native 1M Context"、更强的 Coding 与 Agent 能力（"Strong Coding & Agent Capabilities"），以及可控思考预算（"configurable reasoning levels — none, low, medium, and high"）。核心动机是让 Spark3 1.7B/4B 模型的长上下文编码与工具调用场景能在 SGLang 上直接运行，与近期多个模型接入 PR 一脉相承。

实现拆解

1. 配置层：新增 python/sglang/srt/configs/spark3.py，定义 Spark3Config（继承 PretrainedConfig，model_type = \"spark3\"）。关键默认策略：layer_types 默认按 (i + 1) % 4 每 4 层插入 1 个 full_attention，其余为 sliding_attention；rope_parameters 提供两套默认值，full attention 用 rope_theta = 5000000 与 partial_rotary_factor = 0.25，sliding attention 用 rope_theta = 10000 与全量旋转。同时在 python/sglang/srt/configs/__init__.py 导出，并在 python/sglang/srt/utils/hf_transformers/common.py 注册到 HF 配置注册表，保证从 checkpoint 加载时能自动识别 spark3 类型。
2. 模型层：新增 python/sglang/srt/models/spark3.py，实现 Spark3ForCausalLM。核心设计包括：_get_attention_sliding_window_size 将 HF 的包含式滑动窗口换算为 SGLang 的排除式语义（sliding_window - 1）；Spark3Attention 使用 QKVParallelLinear 支持 GQA，按 attn_tp_size 切分 Q/KV head，并实现 headwise_attn_output_gate（逐 head 门控注意力输出）与 partial_rotary_factor 部分旋转；Spark3MLP 采用 MergedColumnParallelLinear + GeluAndMul 的 gated MLP 结构。模型层通过 make_layers 与 get_layer_id 适配 TP/PP 切分，接入 RadixAttention 与 split prefill（is_dp_attention_enabled）路径，并通过 default_weight_loader 加载权重。

3. 工具调用层：新增 python/sglang/srt/function_call/spark3_detector.py，实现继承 BaseFormatDetector 的 Spark3Detector。解析器通过 ARG_PAIR_PATTERN 正则提取 <arg_key>/<arg_value> 键值对，_convert_value 依据工具声明的 JSON Schema 类型将文本转为 int、float、bool、list、dict，失败时回退原文；流式模式下完整块原子输出，避免 XML 片段被暴露为 JSON 增量。随后在 python/sglang/srt/function_call/function_call_parser.py 将 Spark3Detector 注册到枚举的 \"spark\" 名称下。
4. 测试配套：新增 test/registered/unit/function_call/test_spark3_detector.py（202 行），覆盖注册关系、多调用非流式解析、类型转换与 \"null\" 语义、未知工具策略（SGLANG_FORWARD_UNKNOWN_TOOLS）、流式部分标记与终止刷新，并注册为 CPU CI（base-a-test-cpu）。注意该 PR 没有模型侧的 E2E 测试，PR body 的 MMLU-Pro 数据来自内部 benchmark。

关键文件：

- python/sglang/srt/models/spark3.py（模块 模型实现；类别 source；类型 core-logic；符号 _get_attention_sliding_window_size, Spark3MLP, Spark3Attention, Spark3DecoderLayer）：Spark3 模型核心实现（564 行），包含 SWA/full attention 交错、head-wise 输出门控、TP/PP 并行与 checkpoint 权重加载，是本 PR 的主体。
- python/sglang/srt/function_call/spark3_detector.py（模块 工具解析；类别 source；类型 core-logic；符号 Spark3Detector, _parse_tool_call_xml, _convert_value, _get_param_type）：新增 Spark3 工具调用解析器，解析 XML-KV 协议并处理流式边界，是 agent 场景的关键配套。
- python/sglang/srt/configs/spark3.py（模块 模型配置；类别 source；类型 core-logic；符号 Spark3Config, init）：定义 Spark3Config，承载模型维度、layer_types 默认布局与两套 RoPE 参数，是模型可加载的前提。
- test/registered/unit/function_call/test_spark3_detector.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestSpark3DetectorDetectAndParse, test_spark3_parser_is_registered, test_nonstream_parses_multiple_calls_and_preserves_normal_text, test_null_and_conversion_fallbacks_match_spark3_protocol）：唯一测试配套，覆盖解析器注册、类型转换、未知工具与流式边界，但未覆盖模型加载 / 精度。
- python/sglang/srt/function_call/function_call_parser.py（模块 解析器注册；类别 source；类型 dependency-wiring）：将 Spark3Detector 注册到全局 ToolCallParserEnum 的 "spark" 名称，供用户通过 --tool-call-parser spark 启用。
- python/sglang/srt/configs/__init__.py（模块 配置导出；类别 source；类型 dependency-wiring）：导出 Spark3Config，保证 from sglang.srt.configs import Spark3Config 可用。
- python/sglang/srt/utils/hf_transformers/common.py（模块 模型注册；类别 source；类型 core-logic）：将 Spark3Config 注册进 HF 配置注册表，使 SGLang 能从 checkpoint 自动识别 spark3 模型类型。

关键符号：_get_attention_sliding_window_size, Spark3MLP.forward, Spark3Attention.init, Spark3Attention.forward, Spark3DecoderLayer.forward, Spark3Model.forward, Spark3ForCausalLM.forward, Spark3Config.init, Spark3Detector.detect_and_parse, Spark3Detector.parse_streaming_increment,

_parse_tool_call_xml, _convert_value, _get_param_type

关键源码片段

python/sglang/srt/models/spark3.py

Spark3 模型核心实现（564 行），包含 SWA/full attention 交错、head-wise 输出门控、TP/PP 并行与 checkpoint 权重加载，是本 PR 的主体。

```
# 与 HF 实现保持一致：HF 的 sliding window 语义是包含最后一个 token 的，
# 而 SGLang 假设为排除式窗口，因此这里减 1，避免注意力范围多出一个位置。
def _get_attention_sliding_window_size(config):
    return config.sliding_window - 1


class Spark3MLP(nn.Module):
    def __init__(
        self,
        hidden_size: int,
        intermediate_size: int,
        quant_config: Optional[QuantizationConfig] = None,
        prefix: str = "",
        reduce_results: bool = True,
    ) -> None:
        super().__init__()
        # gated MLP：gate 与 up 合并为一路 MergedColumnParallelLinear，
        # down 投影按需 reduce，支持 TP 下跨 rank 归约。
        self.gate_up_proj = MergedColumnParallelLinear(
            hidden_size,
            [intermediate_size] * 2,
            bias=False,
            quant_config=quant_config,
            prefix=add_prefix("gate_up_proj", prefix),
        )
        self.down_proj = RowParallelLinear(
            intermediate_size,
            hidden_size,
            bias=False,
            quant_config=quant_config,
            prefix=add_prefix("down_proj", prefix),
            reduce_results=reduce_results,
        )
        self.act_fn = GeluAndMul()

    def forward(self, x, forward_batch=None):
        gate_up, _ = self.gate_up_proj(x)
        # GeluAndMul 在激活的同时完成 gate 与 up 的逐元素乘法，
        # 省去一次中间 tensor 的物化。
        x = self.act_fn(gate_up)
        x, _ = self.down_proj(x)
```

```
        return x
```

python/sglang/srt/function_call/spark3_detector.py

新增 Spark3 工具调用解析器，解析 XML-KV 协议并处理流式边界，是 agent 场景的关键配套。

```
def _convert_value(value: str, param_type: str) -> Any:
    # 按工具声明的 JSON Schema 类型转换 XML 文本；
    # integer、number、boolean 等显式收窄，未声明或复合类型走 json.loads，失败则回退原文。
    if value.lower() == "null":
        return None

    normalized_type = param_type.lower()
    try:
        if normalized_type in {"string", "str", "text"}:
            return value
        if normalized_type in {"integer", "int"}:
            return int(value)
        if normalized_type in {"number", "float"}:
            number = float(value)
            return int(number) if number.is_integer() else number
        if normalized_type in {"boolean", "bool"}:
            normalized_value = value.strip().lower()
            if normalized_value not in {"true", "1", "false", "0"}:
                raise ValueError(f"invalid boolean: {value}")
            return normalized_value in {"true", "1"}
        return json.loads(value)
    except (TypeError, ValueError, json.JSONDecodeError):
        try:
            return json.loads(value)
        except (TypeError, ValueError, json.JSONDecodeError):
            return value


def _parse_tool_call_xml(tool_xml: str, tools: list[Tool]) -> _Spark3ToolCall | None:
    # 仅接受完整包裹的 <tool_call>...</tool_call> 块；
    # 函数名取块内第一个 <arg_key> 之前的文本，参数对按正则全部取出。
    if not tool_xml.startswith(TOOL_CALL_BEGIN) or not tool_xml.endswith(TOOL_CALL_END):
        return None

    body = tool_xml[len(TOOL_CALL_BEGIN) : -len(TOOL_CALL_END)]
    first_arg = body.find(ARG_KEY_BEGIN)
    function_name = (body if first_arg < 0 else body[:first_arg]).strip()
    if not function_name:
        return None

    arguments: dict[str, Any] = {}
    for match in ARG_PAIR_PATTERN.finditer(body):
        key, raw_value = match.group(1), match.group(2)
        if not key:
```

```
            continue
        arguments[key] = _convert_value(
            raw_value,
            _get_param_type(tools, function_name, key),
        )
    return _Spark3ToolCall(name=function_name, arguments=arguments)
```

python/sglang/srt/configs/spark3.py

定义 Spark3Config，承载模型维度、layer_types 默认布局与两套 RoPE 参数，是模型可加载的前提。

```
from typing import Any, Optional

from transformers.configuration_utils import PretrainedConfig


class Spark3Config(PretrainedConfig):
    model_type = "spark3"
    architectures = ["Spark3ForCausalLM"]

    def __init__(
        self,
        hidden_size: int = 2048,
        intermediate_size: int = 6656,
        num_attention_heads: int = 8,
        num_key_value_heads: int = 2,
        num_hidden_layers: int = 28,
        head_dim: int = 256,
        headwise_attn_output_gate: bool = True,
        sliding_window: int = 512,
        vocab_size: int = 133120,
        rms_norm_eps: float = 1e-6,
        max_position_embeddings: int = 8192,
        rope_parameters: Optional[dict[str, Any]] = None,
        layer_types: list[str] = None,
        tie_word_embeddings: Optional[bool] = None,
        **kwargs,
    ) -> None:
        self.hidden_size = hidden_size
        self.intermediate_size = intermediate_size
        self.num_attention_heads = num_attention_heads
        self.num_key_value_heads = num_key_value_heads
        self.num_hidden_layers = num_hidden_layers
        self.head_dim = head_dim
        self.headwise_attn_output_gate = headwise_attn_output_gate
        self.sliding_window = sliding_window
        self.vocab_size = vocab_size
        self.rms_norm_eps = rms_norm_eps
        self.max_position_embeddings = max_position_embeddings
```

```
        # 默认层级布局：每 4 层插入 1 个 full attention，其余为 sliding attention，
        # 这是 Spark3 平衡长上下文成本与全局信息获取的关键设计。
        if layer_types is not None:
            layer_types = layer_types[: self.num_hidden_layers]
        else:
            layer_types = [
                "sliding_attention" if bool((i + 1) % 4) else "full_attention"
                for i in range(self.num_hidden_layers)
            ]
        self.layer_types = layer_types

        # 两套 RoPE 参数：full attention 走大 theta + 部分旋转，
        # sliding attention 走小 theta + 全量旋转，保证局部窗口位置编码稳定。
        if rope_parameters is not None:
            self.rope_parameters = rope_parameters
        else:
            self.rope_parameters = {
                "full_attention": {
                    "rope_theta": 5000000,
                    "partial_rotary_factor": 0.25,
                },
                "sliding_attention": {
                    "rope_theta": 10000,
                    "partial_rotary_factor": 1.0,
                },
            }

        super().__init__(**kwargs, tie_word_embeddings=tie_word_embeddings)
```

评论区精华

该 PR 没有产生实质性的技术讨论。唯一 review 来自 Fridge003，直接 APPROVED（评语 \"Nice\"）。Issue 评论区主要是 CI 流程相关：whybeyoung 多次触发 /tag-and-rerun-ci，并提醒 \"lint first, then will be triggered\"，作者执行 pre-commit 修复后回复 \"done\"；dongjiang1989 也触发了一次 CI 重跑。整体呈现 \" 新模型快速合并 \" 的节奏，滑窗语义、head-wise 门控等设计点均未在公开评论中展开权衡讨论。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 模型侧零 E2E 测试：python/sglang/srt/models/spark3.py 新增 564 行，但没有加载、精度或长上下文测试；PR body 中的 MMLU-Pro 数据来自内部 benchmark，CI 无法复现，后续改动容易回归。
  2. 滑动窗口语义易错：_get_attention_sliding_window_size 依赖 HF 与 SGLang 对 inclusive/exclusive 的理解一致，一旦 checkpoint 层顺序或窗口配置不符，会产生静默的注意力截断错误。

3. 新机制组合风险：headwise_attn_output_gate、partial_rotary_factor 与量化、TP/PP 并存路径没有测试覆盖，数值一致性问题难以被发现。
4. 全局解析器注册：Spark3Detector 进入 FunctionCallParser.ToolCallParserEnum 后，相关进程都会加载该模块；流式边界逻辑（如 _partial_marker_suffix_length）若出错会波及所有 spark parser 用户。
5. 长上下文承诺缺乏验证：PR body 声称原生 1M context，但默认 max_position_embeddings 仅为 8192，长文本依赖 RoPE 外推，缺少实测数据。 - 影响：用户侧：Spark3 1.7B/4B 用户可获得开箱即用的部署路径，并能通过 --tool-call-parser spark 启用工具调用解析，长上下文与 agent 场景是主要受益方向。系统侧：改动集中在增量注册（configs 导出、HF 注册表、parser 枚举），对既有模型路径无侵入。团队侧：本 PR 确立了 \" 配置类 + 模型实现 + 工具解析器 + 单测 \" 的新模型接入范式，与 Nemotron 3.5 等近期模型支持一致，但模型侧测试口径尚未统一，后续需要补齐 E2E 精度覆盖。 - 风险标记：新模型无 E2E 测试，滑窗语义换算易错，全局解析器注册，1M 长上下文未实测

关联脉络

- PR #36186 [Model] Support Nemotron 3.5 Lightning speculative decoding: 同为新增模型架构支持（dflash/dspark 配置与模型文件），体现 SGLang 新模型接入的通用模式。
- PR #36284 [CI] Add Kimi-K3 MMMU-Pro accuracy coverage: 展示新模型接入后应配套的精度 CI 覆盖，对照可见 Spark3 目前仅缺模型侧 E2E 验证。