PR #24204 完整报告

sgl-project/sglang

[Model] Laguna-XS.2 Model Support

合并时间：2026-05-09 05:43

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24204

执行摘要

- 一句话：支持 Laguna-XS.2 混合 SWA MoE 模型推理
- 推荐动作：该 PR 适合需要部署 Laguna-XS.2 模型的用户重点关注。对框架开发者，其 FSM 解析器设计和模型注册机制值得学习。代码组织清晰，测试覆盖全面，可快速集成。

功能与动机

根据 PR 描述，若不添加原生支持，checkpoint 只能通过 --trust-remote-code 加载并退化为 HF 参考模块，绕过 RadixAttention、FusedMoE 和 SGLang 的混合 SWA KV cache 池，无法利用框架优化。

实现拆解

1. 模型配置：新建 python/sglang/srt/configs/laguna.py，定义 LagunaConfig 继承 PretrainedConfig，将 HF 嵌套的 rope_parameters 扁平化为 SGLang 可识别的格式；通过 _to_sglang_rope_scaling 转换 RoPE 参数。
2. 模型实现：新建 python/sglang/srt/models/laguna.py，实现 LagunaForCausalLM，包含 LagunaMLP、LagunaMoEGate、LagunaMoE（支持 256 路由专家 + 共享专家）、LagunaAttention（每层可配置不同注意力头数和 RoPE 类型）、LagunaDecoderLayer；通过 get_moe_weights 加载权重。
3. 工具调用解析器：新建 python/sglang/srt/function_call/poolside_v1_detector.py，实现 PoolsideV1Detector，基于 5 状态 FSM (_ParseState) 流式解析 <tool_call>... 格式；支持 schema 类型强制转换和异常恢复。
4. 推理解析器：修改 python/sglang/srt/parser/reasoning_parser.py，添加 _PoolsideV1Detector（后直接映射到 Qwen3Detector），设置 reasoning_default = explicit_enable_thinking，默认关闭推理，需通过 chat_template_kwargs 开启。
5. 模型注册：修改 python/sglang/srt/configs/model_config.py 和 python/sglang/srt/function_call/function_call_parser.py，将 laguna 架构注册到 _CONFIG_REGISTRY，将 poolside_v1 解析器注册到工具 / 推理解析器映射表。
6. 测试覆盖：新增 test/registered/unit/function_call/test_poolside_v1_detector.py（38 个用例，覆盖正常、流式、异常输入）；修改 test/registered/unit/entrypoints/openai/test_serving_chat.py 和 test/registered/unit/parser/test_reasoning_parser.py，验证推理分发和不会重复添加 <think>。

关键文件：

- python/sglang/srt/models/laguna.py（模块 模型实现；类别 source；类型 data-contract；符号 LagunaMLP, init, forward, LagunaMoEGate）：核心模型文件，实现 LagunaForCausalLM、LagunaMLP、LagunaMoE、LagunaAttention 等关键模块。
- python/sglang/srt/function_call/poolside_v1_detector.py（模块 工具解析；类别 source；类型 dependency-wiring；符号 _ParseState, PoolsideV1Detector, init, _reset_call_state）：工具调用解析器，实现基于 FSM 的流式解析，核心设计亮点。
- python/sglang/srt/configs/laguna.py（模块 模型配置；类别 source；类型 dependency-wiring；符号 _first_not_none, _to_sglang_rope_scaling, LagunaConfig, init）：模型配置处理，负责从 HF 配置转换 rope 参数并标准化。
- python/sglang/srt/parser/reasoning_parser.py（模块 推理解析；类别 source；类型 core-logic；符号 _PoolsideV1Detector, init）：注册 poolside_v1 推理解析器，关键设计是将默认行为设为 explicit_enable_thinking。
- test/registered/unit/function_call/test_poolside_v1_detector.py（模块 工具解析测试；类别 test；类型 test-coverage；符号 TestPoolsideV1Detector, setUp, test_has_tool_call_true, test_has_tool_call_false）：解析器单元测试，38 个用例覆盖正常、流式、异常输入，确保 FSM 正确性。
- test/registered/unit/entrypoints/openai/test_serving_chat.py（模块 聊天服务测试；类别 test；类型 test-coverage；符号 test_poolside_v1_enable_thinking_dispatch, test_poolside_v1_does_not_double_prepend_think）：测试推理 mode 的分发逻辑，确保 enable_thinking 正确传递且不重复添加 。
- python/sglang/srt/configs/model_config.py（模块 模型配置；类别 source；类型 data-contract）：注册模型架构到 _CONFIG_REGISTRY，使 SGLang 能加载 laguna 模型。
- python/sglang/srt/function_call/function_call_parser.py（模块 工具解析；类别 source；类型 dependency-wiring）：注册 poolside_v1 工具调用解析器到解析器映射表。
- python/sglang/srt/configs/__init__.py（模块 配置导出；类别 source；类型 dependency-wiring）：导出 LagunaConfig 以支持自动发现。
- python/sglang/srt/utils/hf_transformers/common.py（模块 工具库；类别 source；类型 core-logic）：确保依赖库版本兼容性，辅助模型加载。
- docs_new/docs/supported-models/generative_models.mdx（模块 文档；类别 other；类型 data-contract）：更新支持模型列表，使用户可查到 Lagana-XS.2。

关键符号：LagunaMLP.forward, LagunaMoEGate.forward, LagunaMoE.get_moe_weights, LagunaDecoderLayer.forward, PoolsideV1Detector.detect_and_parse, PoolsideV1Detector.parse_streaming_increment, _to_sglang_rope_scaling, _PoolsideV1Detector.init

关键源码片段

python/sglang/srt/models/laguna.py

核心模型文件，实现 LagunaForCausalLM、LagunaMLP、LagunaMoE、LagunaAttention 等关键模块。

```
# 核心前馈网络：合并门控 + 上投影，Silu 激活，下投影
class LagunaMLP(nn.Module):
    def __init__(
        self,
        hidden_size: int,
        intermediate_size: int,
        hidden_act: str,
        quant_config: Optional[QuantizationConfig] = None,
        reduce_results: bool = True,
        prefix: str = "",
    ) -> None:
        super().__init__()
        if hidden_act != "silu":
            raise ValueError(f"Unsupported activation: {hidden_act}. Only silu is supported.")
        # 合并列并行：输出 intermediate_size * 2，分别作为 gate 和 up
        self.gate_up_proj = MergedColumnParallelLinear(
            hidden_size,
            [intermediate_size] * 2,
            bias=False,
            quant_config=quant_config,
            prefix=add_prefix("gate_up_proj", prefix),
        )
        # 行并行降维
        self.down_proj = RowParallelLinear(
            intermediate_size,
            hidden_size,
            bias=False,
            quant_config=quant_config,
            reduce_results=reduce_results,
            prefix=add_prefix("down_proj", prefix),
        )
        self.act_fn = SiluAndMul()

    def forward(
        self,
        x: torch.Tensor,
        forward_batch: Optional[ForwardBatch] = None,
        should_allreduce_fusion: bool = False,
        use_reduce_scatter: bool = False,
    ) -> torch.Tensor:
        gate_up, _ = self.gate_up_proj(x)
        x = self.act_fn(gate_up)
        # 跳过块内 all-reduce 如果后续有融合或下一层期望 reduce-scatter
        x, _ = self.down_proj(
            x,
            skip_all_reduce=should_allreduce_fusion or use_reduce_scatter,
        )
        return x
```

python/sglang/srt/function_call/poolside_v1_detector.py

工具调用解析器，实现基于 FSM 的流式解析，核心设计亮点。

```
class _ParseState(Enum):
    """FSM 的 5 个状态。
    READING_VALUE 只能从 READING_KEY 进入，结构上杜绝"值出现在调用之前"。
    退出守卫：READING_KEY 和 READING_VALUE 均可通过 `</tool_call>` 关闭调用；
    READING_VALUE 还可在 `<arg_key>` 时替换孤儿 pending key。
    """
    OUTSIDE = auto()
    READING_NAME = auto()
    READING_KEY = auto()
    READING_VALUE = auto()
    DRAINING = auto()

class PoolsideV1Detector(BaseFormatDetector):
    # 线格式标记常量
    tool_call_start_token = "<tool_call>"
    tool_call_end_token = "</tool_call>"
    arg_key_start = "<arg_key>"
    arg_key_end = "</arg_key>"
    arg_value_start = "<arg_value>"
    arg_value_end = "</arg_value>"

    # 正则：键使用 [^<]*? 防止跨边界回溯；值保留 .*? 以允许包含 <
    tool_call_regex = re.compile(r"<tool_call>(.*?)</tool_call>", re.DOTALL)
    arg_pair_regex = re.compile(
        r"<arg_key>([^<]*?)</arg_key>\s*<arg_value>(.*?)</arg_value>", re.DOTALL
    )

    def __init__(self):
        super().__init__()
        self.parsed_pos: int = 0
        self._state: _ParseState = _ParseState.OUTSIDE
        self.current_func_name: Optional[str] = None
        self.current_pending_key: Optional[str] = None
        self.json_started: bool = False

    def _consume_arg_key(self, slice_: str) -> bool:
        """消耗 `<arg_key>K</arg_key>`，设置 current_pending_key。
        如果 `</arg_key>` 未到则返回 False。
        READING_KEY 中调用：正常转移到 READING_VALUE；
        READING_VALUE 中调用：替换孤儿键，仍留在 READING_VALUE。
        """
        end = slice_.find(self.arg_key_end)
        if end == -1:
            return False
        self.current_pending_key = slice_[len(self.arg_key_start):end].strip()
        self.parsed_pos += end + len(self.arg_key_end)
```

```
        return True

    def _close_current_call(self, calls: List[ToolCallItem]) -> None:
        """关闭当前调用：追加结尾 `}`（或 `{}` 零参数），
        跳过 `</tool_call>`，回到 OUTSIDE，重置状态。
        处理正常路径和异常关闭（如孤键后直接关闭）。
        """
        fragment = "}" if self.json_started else "{}"
        # ... 省略中间处理
        self._reset_call_state()
        self._state = _ParseState.OUTSIDE
        self.parsed_pos += len(self.tool_call_end_token)
```

python/sglang/srt/configs/laguna.py

模型配置处理，负责从 HF 配置转换 rope 参数并标准化。

```
# 将 HF 的嵌套 rope 参数转换为 SGLang `get_rope` 所需的 `rope_scaling` 字典
def _to_sglang_rope_scaling(rope_params: Dict[str, Any]) -> Optional[Dict[str, Any]]:
    if not rope_params:
        return None
    rope_type = rope_params.get("rope_type") or rope_params.get("type")
    if rope_type in (None, "default"):
        return None

    out: Dict[str, Any] = {"rope_type": rope_type}
    # 直接透传的字段
    pass_through = (
        "factor", "original_max_position_embeddings", "beta_fast", "beta_slow",
        "extrapolation_factor", "truncate", "low_freq_factor", "high_freq_factor",
        "mscale", "mscale_all_dim", "short_factor", "long_factor",
        "short_mscale", "long_mscale",
    )
    for key in pass_through:
        if key in rope_params:
            out[key] = rope_params[key]
    # HF 用 attention_factor，SGLang 用 attn_factor
    if "attention_factor" in rope_params:
        out["attn_factor"] = rope_params["attention_factor"]
    return out

class LagunaConfig(PretrainedConfig):
    model_type = "laguna"
    keys_to_ignore_at_inference = ["past_key_values"]

    def __init__(
        self,
        vocab_size: int = 100352,
        hidden_size: int = 2048,
        intermediate_size: int = 8192,
```

```
        num_hidden_layers: int = 40,
        num_attention_heads: int = 48,
        num_key_value_heads: int = 8,
        head_dim: int = 128,
        hidden_act: str = "silu",
        max_position_embeddings: int = 131072,
        initializer_range: float = 0.02,
        rms_norm_eps: float = 1e-6,
        use_cache: bool = True,
        tie_word_embeddings: bool = False,
        attention_bias: bool = False,
        attention_dropout: float = 0.0,
        sliding_window: int = 512,
        layer_types: Optional[List[str]] = None,
        mlp_layer_types: Optional[List[str]] = None,
        num_attention_heads_per_layer: Optional[List[int]] = None,
        num_experts: int = 256,
        num_experts_per_tok: int = 8,
        moe_intermediate_size: int = 512,
        shared_expert_intermediate_size: int = 512,
        moe_routed_scaling_factor: float = 1.0,
        moe_router_logit_softcapping: float = 0.0,
        moe_apply_router_weight_on_input: bool = False,
        # HF 的嵌套 rope 参数，键如 "full_attention" / "sliding_attention"
        rope_parameters: Optional[Dict[str, Any]] = None,
        partial_rotary_factor: Optional[float] = None,
        rope_theta: Optional[float] = None,
        rope_scaling: Optional[Dict[str, Any]] = None,
        bos_token_id: Optional[int] = 2,
        eos_token_id: Optional[Any] = None,
        pad_token_id: Optional[int] = 9,
        **kwargs,
    ):
        super().__init__(...)
        self.sliding_window = sliding_window
        # ... 设置 MoE 和注意力相关字段
```

评论区精华

1. 推理解析器子类设计：Reviewer @kpham-sgl 指出 PoolsideV1Detector 继承 Qwen3Detector 但未重写任何方法，质疑必要性。作者 @Jiminator 接受建议，后续提交将映射直接改为 Qwen3Detector，简化代码。
2. TP > 1 与 DP Attention：Reviewer 询问 TP>1 准确性和 DP attention 支持。作者回复 DP attention 开箱即用，表明多卡推理已得到初步验证。

- PoolsideV1Detector 子类设计 (design): 作者接受建议，将映射改为直接使用 Qwen3Detector，避免无意义的子类。

- TP > 1 准确性与 DP Attention (question): 作者回复 DP attention 开箱即用，并在 PR body 补充了 TP=4 的准确率和性能数据。

风险与影响

- 风险：
  1. 模型兼容性：新模型代码未经长期生产验证，可能与其他编码配置（如压缩张量加载）存在边缘问题。已在 packed_modules_mapping 中修复 fused 层识别。
  2. 工具解析稳定性：FSM 解析器对恶意构造的输入可能仍有未覆盖的异常路径，但 38 个测试包括截断、异常关闭等场景。
  3. 回归风险：模型注册修改了 model_config.py 和 __init__ 等核心文件，可能影响其他模型加载。但改动较小，且已有 CI 覆盖。
  4. 性能风险：混合 SWA 注意力与 DP attention 的组合性能未充分 benchmark（仅提供 TP=1 和 TP=4 数据）。 - 影响：用户影响：支持直接加载 Laguna-XS.2 checkpoint，无需 trust_remote_code，并且可以使用 SGLang 的优化内核（RadixAttention、FusedMoE）获得更好的推理性能。工具调用和推理模式对 agent 应用友好。系统影响：增加了约 2000 行新代码，但模块相对独立，维护成本可控。团队影响：扩展了模型支持图谱，为后续类似混合架构模型提供参考模式（如 FSM 解析器设计）。
- 风险标记：新模型支持，测试覆盖缺项，工具解析输入容错

关联脉络

- 暂无明显关联 PR