PR #23533 完整报告

sgl-project/sglang

support Hy3 preview

合并时间：2026-04-25 03:03

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23533

执行摘要

- 一句话：支持腾讯混元 V3(Hy3-preview) 模型推理与工具调用
- 推荐动作：该 PR 值得精读，特别是双流 MoE 重叠方案、自定义融合 TopK kernel 的实现、以及流式工具解析器的增量输出设计。建议后续关注 group topk kernel 的重构和 AMD 兼容性修复。

功能与动机

用户需要在 sglang 中部署和推理腾讯最新 Hunyuan V3（Hy3-preview）模型，该模型采用 MoE 架构并支持函数调用与推理模式。PR 通过提供完整的模型适配、专用解析器和调优配置，使 sglang 成为支持 Hy3-preview 的推理框架之一。

实现拆解

1. 模型定义：新建 hunyuan_v3.py 和 hunyuan_v3_nextn.py，实现 HYV3FeedForward、HYV3MoEFused（双流 MoE）、HYV3Attention 等核心模块，支持单流 / 双流前向、专家偏置加载、分组 TopK 路由。
2. 性能优化融合核：在 jit_kernel/grouped_topk.py 中新增 grouped_topk 函数，调用 JIT 编译的 CUDA kernel 实现 Sigmoid+ 偏置 +TopK+ 重归一化的融合，替代原先多次 torch.topk 调用；CUDA 源文件 grouped_topk.cuh 实现具体逻辑，支持单分组、低于 512 专家、TopK≤8。
3. 流式工具调用解析器：hunyuan_detector.py 实现 HunyuanDetector，解析 Hy3-preview 的 XML 风格工具调用格式，支持两阶段流式输出：阶段一输出函数名，阶段二增量流式输出参数 JSON，对纯字符串参数实现字符级流式。同时包含模式感知的类型强制转换。
4. 推理解析器：在 reasoning_parser.py 中添加 HunyuanDetector，支持 <think>...</think> 标签，并根据 chat_template_kwargs 中的 reasoning_effort 值（high/low/none/no_think）正确切换推理模式。
5. 配置与部署配套：新增多个调优的 MoE Triton 配置文件（H20/B200，fp8/bf16），覆盖 E=192,N=192 场景；修改 topk.py 以支持 use_grouped_topk 路由；更新 serving_chat.py 推理请求处理；新增使用文档 hy3_preview.md。
6. 测试覆盖：test_hunyuan_detector.py 包含 50 个测试，覆盖 HasToolCall、DetectAndParse、StreamingParse 等场景；test_reasoning_parser.py 新增 TestHunyuanDetector 覆盖推理解析与工具中断。

关键文件：

- python/sglang/srt/models/hunyuan_v3.py（模块 模型定义；类别 source；类型 data-contract；符号 HYV3FeedForward, init, forward, HYV3MoEFused）：模型核心实现，包含 HYV3MoEFused 双流 MoE、HYV3Attention、专家偏置加载等，新增 587 行。
- python/sglang/srt/models/hunyuan_v3_nextn.py（模块 MTP 模型；类别 source；类型 data-contract；符号 HYV3ModelNextN, init, forward, HYV3ForCausalLMNextN）：实现 MTP（Multi-Token Prediction）推测解码模型，重用 HYV3DecoderLayer，新增 253 行。
- python/sglang/srt/function_call/hunyuan_detector.py（模块 工具解析器；类别 source；类型 dependency-wiring；符号 HunyuanDetector, init, _normalize_type, _get_arg_schema）：实现 Hy3-preview 的 XML 风格工具调用解析器，支持两阶段流式输出和模式感知类型转换，新增 476 行。
- python/sglang/jit_kernel/grouped_topk.py（模块 内核封装；类别 source；类型 core-logic；符号 _jit_grouped_topk_module, _jit_grouped_topk_op, grouped_topk）：新增融合分组 TopK kernel 的 Python 封装，调用 JIT 编译的 CUDA kernel，用于 MoE 路由优化，新增 89 行。
- test/registered/unit/function_call/test_hunyuan_detector.py（模块 测试；类别 test；类型 test-coverage；符号 _make_tools, TestHunyuanDetectorHasToolCall, setUp, test_has_tool_call_true）：单元测试，50 个用例覆盖工具调用检测、流式解析、类型转换等，确保 HunyuanDetector 正确性。
- test/registered/unit/parser/test_reasoning_parser.py（模块 测试；类别 test；类型 test-coverage；符号 TestHunyuanDetector, setUp, test_init, test_detect_and_parse_normal_reasoning）：新增 Hunyuan 推理解析器测试，覆盖正常推理、无推理、工具中断、流式场景。
- python/sglang/srt/parser/reasoning_parser.py（模块 推理解析；类别 source；类型 core-logic；符号 HunyuanDetector, init）：修改推理解析器，增加 HunyuanDetector 类，支持 标签与 reasoning_effort 控制。
- python/sglang/jit_kernel/csrc/moe/grouped_topk.cuh（模块 内核 CUDA；类别 other；类型 dependency-wiring）：CUDA 源文件，实际实现融合分组 TopK kernel，267 行。
- python/sglang/srt/layers/moe/topk.py（模块 路由层；类别 source；类型 dependency-wiring）：修改 TopK 模块，支持 use_grouped_topk 选项，用于路由分组。
- python/sglang/srt/layers/moe/fused_moe_triton/configs/triton_3_5_1/E=192,N=192,device_name=NVIDIA_H20,dtype=fp8_w8a8.json（模块 MoE 配置；类别 config；类型 configuration）：H20 fp8 MoE 调优配置，新增 146 行。

关键符号：HYV3MoEFused.forward, HYV3MoEFused.ebias_weight_loader, grouped_topk, HunyuanDetector.parse_streaming_increment, HunyuanDetector.detect_and_parse, HYV3ModelNextN.forward, HYV3ForCausalLMNextN.load_weights, HunyuanDetector.init

关键源码片段

python/sglang/srt/models/hunyuan_v3.py

模型核心实现，包含 HYV3MoEFused 双流 MoE、HYV3Attention、专家偏置加载等，新增 587 行。

```
# -*- coding: utf-8 -*-
# Copyright (c) 2026 The HunYuan team.

import torch
from torch import nn
from sglang.srt.layers.moe.topk import TopK

class HYV3MoEFused(nn.Module):
    """
    双流 MoE 层：支持单流（默认）和双流（通过 alt_stream 异步执行 gate 与 expert 计算）
    使用 fused grouped top-k 路由，结合 sigmoid 评分与 expert bias 修正。
    """
    def __init__(self, config, layer_id, quant_config, prefix="", alt_stream=None):
        super().__init__()
        # ( 第一部分 ): 初始化超参数与路由 gate
        self.tp_size = get_moe_tensor_parallel_world_size()
        self.ep_size = get_moe_expert_parallel_world_size()
        self.layer_id = layer_id
        self.alt_stream = alt_stream # 双流模式下使用独立的 CUDA stream

        # expert_bias: 用于路由修正的偏置参数，权重加载器设置为 ebias_weight_loader
        self.expert_bias = nn.Parameter(
            torch.empty(config.num_experts, dtype=torch.float32)
        )
        self.expert_bias.weight_loader = HYV3MoEFused.ebias_weight_loader

        # 路由评分函数为 sigmoid，由 TopK 层实现分组 top-k + renormalize
        self.gate = ReplicatedLinear(
            config.hidden_size, config.num_experts,
            bias=False, quant_config=None, params_dtype=torch.float32,
            prefix=f"{prefix}.gate",
        )
        self.topk = TopK(
            top_k=config.num_experts_per_tok,
            use_grouped_topk=True,
            num_expert_group=1,
            topk_group=1,
            renormalize=config.route_norm,
            scoring_func="sigmoid",
            correction_bias=self.expert_bias, # 路由偏置修正
            routed_scaling_factor=getattr(config, "router_scaling_factor", 1.0),
            apply_routed_scaling_factor=True,
        )

    def forward(self, hidden_states, forward_batch, residual):
        # (forward 逻辑 ) 调用 self.gate 得 expert_weights，然后通过 FusedMoE 计算
```

```
        # 此处省略后续细节
        pass
```

python/sglang/srt/models/hunyuan_v3_nextn.py

实现MTP（Multi-Token Prediction）推测解码模型，重用HYV3DecoderLayer，新增253行。

```
# -*- coding: utf-8 -*-
# MTP（Multi-Token Prediction）模型，用于 speculative decoding 的 draft 阶段。
# 它会重用 HYV3DecoderLayer，强制使用 MoE（即使 config 中 first_k_dense_replace > 0）。

class HYV3ModelNextN(nn.Module):
    def __init__(self, config, quant_config=None, prefix=""):
        super().__init__()
        self.embed_tokens = VocabParallelEmbedding(config.vocab_size, config.hidden_size)
        self.enorm = RMSNorm(config.hidden_size, eps=config.rms_norm_eps)
        self.hnorm = RMSNorm(config.hidden_size, eps=config.rms_norm_eps)
        # 拼接前一刻的 hidden_states 与当前 embedding
        self.eh_proj = nn.Linear(2 * config.hidden_size, config.hidden_size, bias=False)

        self.alt_stream = torch.cuda.Stream() if is_cuda() else None
        # 强制当前层为 MoE：临时覆盖 first_k_dense_replace
        orig_first_k = getattr(config, "first_k_dense_replace", 0)
        config.first_k_dense_replace = 0
        self.decoder = HYV3DecoderLayer(
            config=config, layer_id=0, quant_config=quant_config,
            prefix=f"{prefix}.decoder", alt_stream=self.alt_stream,
        )
        config.first_k_dense_replace = orig_first_k

        self.shared_head = nn.Module()
        self.shared_head.norm = RMSNorm(config.hidden_size, eps=config.rms_norm_eps)

    def forward(self, input_ids, positions, forward_batch, input_embeds=None):
        # 获取 hidden_states 并拼接前一层的 hidden_states
        hidden_states = self.embed_tokens(input_ids) if input_embeds is None else input_embeds
        if hidden_states.shape[0] > 0:
            hidden_states = self.eh_proj(
                torch.cat([self.enorm(hidden_states),
                           self.hnorm(forward_batch.spec_info.hidden_states)], dim=-1)
            )
        hidden_states, residual = self.decoder(positions, hidden_states, forward_batch, None)
        # 归一化输出（仅在非 idle 模式）
        if not forward_batch.forward_mode.is_idle():
            hidden_states = self.shared_head.norm(hidden_states if residual is None else hidden_
            states + residual)
        return hidden_states
```

python/sglang/srt/function_call/hunyuan_detector.py

实现 Hy3-preview 的 XML 风格工具调用解析器，支持两阶段流式输出和模式感知类型转换，新增 476 行。

```
# -*- coding: utf-8 -*-
# HunyuanDetector：解析 Hy3-preview 模型的工具调用格式。
# 格式为 <tool_calls><tool_call>name<tool_sep><arg_key>k</arg_key><arg_value>v</arg_
value></tool_call></tool_calls>
# 流式解析分为两个阶段：
# 阶段 1：遇到 <tool_sep> 后立即输出函数名
# 阶段 2：增量构建参数 JSON，纯字符串参数支持字符级流式，复合类型等待完整闭合后输出。

class HunyuanDetector(BaseFormatDetector):
    def __init__(self):
        super().__init__()
        # 定义 token 常量
        self.bot_token = "<tool_calls>"
        self.eot_token = "</tool_calls>"
        self.tool_call_start_token = "<tool_call>"
        self.tool_call_end_token = "</tool_call>"
        self.tool_sep_token = "<tool_sep>"
        self.arg_key_start_token = "<arg_key>"
        self.arg_key_end_token = "</arg_key>"
        self.arg_value_start_token = "<arg_value>"
        self.arg_value_end_token = "</arg_value>"

        # 流式状态
        self._in_tool_calls = False
        self._streaming_tool_name: Optional[str] = None
        self._completed_args: Dict[str, Any] = {}
        self._streamed_json_len = 0

    def parse_streaming_increment(self, new_text: str, tools: List[Tool] = None) ->
    StreamingParseResult:
        # 增量解析逻辑：处理 <tool_calls> 起始标签，拆分阶段，
        # 纯字符串参数先在内部追加字符，仅在遇到 </arg_value> 时输出完整键值。
        # 参数类型从 tools 中获取 schema，决定是否可流式。
        result = StreamingParseResult()
        # ... 具体解析过程略
        return result
```

评论区精华

AMD 兼容性风险：AMD 验证者 @andyluo7 在 Issue 评论中报告，在 MI300X 和 MI355X 上使用 CUDA graph 模式时发生 HSA_STATUS_ERROR_EXCEPTION 崩溃，根因定位到 AITER 的 custom all-reduce（默认在 HIP 上启用）。Eager mode 可正常工作。建议用户在使用 AMD 时先使用 --disable-cuda-graph。 组 TopK Kernel 后续重构：合并者 Qiaolin-Yu 表示 group topk kernel 后续可能需要重构，但当前设计可以合并。

- AMD CUDA Graph 兼容性 (correctness): 暂时建议 AMD 用户使用 --disable-cuda-graph ，后续修复。
- Group TopK Kernel 后续重构 (design): 当前设计可以合并，后续会优化。

风险与影响

- 风险：AMD CUDA Graph 兼容性：CUDA graph 模式下在 AMD GPU 上出现 HSA 异常，需要用户禁用 CUDA graph 或等待后续修复。可能影响 AMD 用户的部署体验。 新引入代码回归风险：模型、解析器、kernel 均为新增，尽管有测试覆盖，但集成到完整推理 pipeline 中可能出现未预料的交互。MoE 双流重叠与 alt_stream 的使用在 CUDA graph 捕获模式下可能有潜在冲突。 配置依赖：新增的 Moe Triton 配置文件依赖特定 Triton 版本（3_5_1），若环境版本不匹配可能导致运行时错误。 安全考虑：工具调用解析器中的流式处理涉及部分标签持有后输出，若输入异常可能引入解析状态异常，但测试覆盖了边界情况。
- 影响：用户侧：Hy3-preview 用户可直接在 sglang 中使用模型推理、工具调用、推理模式切换。新增 --tool-call-parser hunyuan 和 --reasoning-parser hunyuan 命令行参数。 系统侧：新增约 4K 行代码，主要影响模型加载、推理内核、解析器模块。模型代码基于现有层组件，对现有模型无影响。MoE 路由路径新增 grouped_topk 选项，但默认不影响其他模型。 团队侧：需要维护新模型的后向兼容性，特别是当 Hy3-preview 正式发布后可能引入 checkpoint 变更。AMD 专家需关注 CUDA graph 问题。
- 风险标记：AMD CUDA Graph 兼容，MoE 双流重叠可能锁定，Triton 版本依赖，新代码回归风险

关联脉络

- 暂无明显关联 PR