PR #20918 完整报告

sgl-project/sglang

[NPU] Support MTP for Qwen3.5

合并时间：2026-04-27 10:44

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/20918

执行摘要

- 一句话：Ascend NPU 上为 Qwen3.5 添加 MTP 推测解码支持
- 推荐动作：值得精读，特别是 NPU 注意力后端的架构设计以及如何复用 GPU 端的抽象接口。建议关注作者在 attention_registry.py 中的条件路由模式，以及使用 ExitStack 管理线程安全环境变量的做法。

功能与动机

适配 Qwen3.5 模型在 Ascend NPU 平台上的 MTP（多 Token 预测）推测解码功能，修复推理错误，确保稳定高效的模型运行。

实现拆解

1. 新增 NPU 专用 GDN 注意力后端：在 python/sglang/srt/hardware_backend/npu/attention/ascend_gdn_backend.py 中实现 AscendGDNAttnBackend，继承自 AscendMambaAttnBackendBase，封装了 NPU 上的 fused_gdn_gating、causal_conv1d 等算子，并实现 prepare_gdn_inputs、forward_decode、forward_extend 等核心方法。
2. 新增 Ascend 混合线性注意力后端基类：在 ascend_hybrid_linear_attn_backend.py 中定义 AscendMambaAttnBackendBase，扩展了 GPU 的 MambaAttnBackendBase，增加了 state_indices_list_gdn 以支持 GDN 的 verify 模式，并重写 init_cuda_graph_state、_capture_metadata、_replay_metadata 等 CUDA Graph 相关方法。
3. 调整注意力后端路由：在 attention_registry.py 的 attn_backend_wrapper 中，当运行在 NPU 上时，将 GDNAttnBackend、HybridLinearAttnBackend、Mamba2AttnBackend 分别替换为 Ascend 版本，实现平台自动切换。
4. 修改 MTP 模型以适配 NPU 无量化运行：在 qwen3_5_mtp.py 和 qwen3_next_mtp.py 中，当运行在 NPU 且 draft 模型未指定量化时，强制 quant_config = None，并通过 ExitStack 临时设置环境变量 SGLANG_DEEPEP_BF16_DISPATCH 和 DEEP_NORMAL_MODE_USE_INT8_QUANT 以禁用量化路径，确保兼容性。
5. 扩展 conv state 内存分配：在 memory_pool_npu.py 中新增 _init_npu_conv_state 函数，根据 speculative_num_draft_tokens 在 conv state 的 conv_width 维度上增加额外长度，以容纳 MTP draft tokens 的中间状态。

关键文件：

- python/sglang/srt/hardware_backend/npu/attention/ascend_gdn_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 AscendGDNAttnBackend, init, prepare_gdn_inputs, init_forward_metadata）：新增 NPU 专用的 GDN 注意力后端，包含 MTP 推测解码所需的所有前向逻辑
- python/sglang/srt/hardware_backend/npu/attention/ascend_hybrid_linear_attn_backend.py（模块 混合注意力后端；类别 source；类型 dependency-wiring；符号 AscendMambaAttnBackendBase, init, init_cuda_graph_state, _capture_metadata）：新增 Ascend 混合线性注意力后端基类，提供了 GDN 相关的 CUDA Graph 状态管理和 verify 模式下索引生成
- python/sglang/srt/hardware_backend/npu/memory_pool_npu.py（模块 内存池；类别 source；类型 core-logic；符号 _init_npu_conv_state）：新增 _init_npu_conv_state 函数，根据 speculative_num_draft_tokens 扩展 conv state 的宽度，确保 MTP draft tokens 的卷积状态空间足够
- python/sglang/srt/models/qwen3_5_mtp.py（模块 MTP 模型；类别 source；类型 data-contract）：修改 MTP 模型初始化，在 NPU 且无 draft 量化时强制 disable quant_config，并在 forward 中用 ExitStack 临时设置环境变量以禁用量化相关路径
- python/sglang/srt/models/qwen3_next_mtp.py（模块 MTP 模型；类别 source；类型 data-contract）：与 qwen3_5_mtp.py 类似，为 Qwen3Next MTP 模型添加相同的 NPU 无量化逻辑
- python/sglang/srt/layers/attention/attention_registry.py（模块 注意力路由；类别 source；类型 dependency-wiring）：修改注意力后端路由，当检测到 NPU 时替换 GPU 后端为 Ascend 专用版本
- python/sglang/srt/layers/layernorm.py（模块 层归一化；类别 source；类型 dependency-wiring）：小调整，可能与 NPU 的 LayerNorm 兼容性有关
- python/sglang/srt/mem_cache/memory_pool.py（模块 内存池；类别 source；类型 dependency-wiring）：小调整，为 NPU 的 conv state 初始化提供钩子
- python/sglang/srt/environ.py（模块 环境变量；类别 source；类型 core-logic）：添加了两个环境变量 SGLANG_DEEPEP_BF16_DISPATCH 和 DEEP_NORMAL_MODE_USE_INT8_QUANT 的可配置定义，用于 MTP 无量化模式
- python/sglang/srt/layers/attention/mamba/mamba2_metadata.py（模块 元数据结构；类别 source；类型 core-logic）：添加 mamba_cache_indices_gdn 字段支持

关键符号：AscendGDNAttnBackend.init, AscendGDNAttnBackend.prepare_gdn_inputs, AscendGDNAttnBackend.init_forward_metadata, AscendGDNAttnBackend.forward_decode, AscendGDNAttnBackend.forward_extend, AscendMambaAttnBackendBase.init_cuda_graph_state, AscendMambaAttnBackendBase._capture_metadata, _init_npu_conv_state, Qwen3_5ForCausalLMMTP.forward

关键源码片段

python/sglang/srt/hardware_backend/npu/attention/ascend_gdn_backend.py

新增 NPU 专用的 GDN 注意力后端，包含 MTP 推测解码所需的所有前向逻辑

```
from typing import Optional, Tuple, Union
import torch
from sgl_kernel_npu.fla.fused_gdn_gating import fused_gdn_gating_npu
from sgl_kernel_npu.mamba.causal_conv1d import causal_conv1d_fn_npu, causal_conv1d_
update_npu
from sglang.srt.hardware_backend.npu.attention.ascend_hybrid_linear_attn_backend import
AscendMambaAttnBackendBase
from sglang.srt.layers.attention.linear.gdn_backend import GDNKernelDispatcher
from sglang.srt.layers.attention.linear.utils import get_linear_attn_decode_backend, get_linear_
attn_prefill_backend
from sglang.srt.layers.radix_linear_attention import RadixLinearAttention
from sglang.srt.mem_cache.memory_pool import MambaPool
from sglang.srt.model_executor.forward_batch_info import ForwardBatch, ForwardMode
from sglang.srt.model_executor.model_runner import ModelRunner
from sglang.srt.speculative.eagle_info import EagleDraftInput, EagleVerifyInput

# 将 NPU 版本的函数赋值给统一名称
fused_gdn_gating = fused_gdn_gating_npu
causal_conv1d_fn = causal_conv1d_fn_npu
causal_conv1d_update = causal_conv1d_update_npu

class AscendGDNAttnBackend(AscendMambaAttnBackendBase):
    """Ascend NPU 专用的 GDN 注意力后端，适配了 MTP 推测解码的 verify 模式"""

    def __init__(self, model_runner: ModelRunner):
        super().__init__(model_runner)
        # 初始化卷积状态形状：维度交换以满足 NPU conv1d 算子的要求
        self.conv_states_shape = torch.Size((
            *model_runner.req_to_token_pool.mamba_pool.mamba_cache.conv[0].shape[:-2],
            model_runner.req_to_token_pool.mamba_pool.mamba_cache.conv[0].shape[-1],
            model_runner.req_to_token_pool.mamba_pool.mamba_cache.conv[0].shape[-2],
        ))
        decode_backend = get_linear_attn_decode_backend()
        prefill_backend = get_linear_attn_prefill_backend()
        self.kernel_dispatcher = GDNKernelDispatcher(decode_backend, prefill_backend)

    def prepare_gdn_inputs(
        self,
        bs: int,
        forward_mode: ForwardMode,
        spec_info: Optional[Union[EagleDraftInput, EagleVerifyInput]],
    ):
        """根据 forward_mode 准备 GDN 输入：在 verify 模式下生成连续的 ssm_state_indices"""
        cache_indices = self.forward_metadata.mamba_cache_indices
        self.num_accepted_tokens = torch.ones([bs], dtype=torch.int32, device=cache_indices.
        device)
        self.actual_seq_lengths = torch.ones([bs], dtype=torch.int32, device=cache_indices.device)
```

```
        if forward_mode.is_target_verify():
            seq_len = spec_info.draft_token_num
            self.actual_seq_lengths = self.actual_seq_lengths * seq_len
            # 生成连续的索引用于 verify 时按顺序访问中间状态
            self.ssm_state_indices = torch.arange(
                cache_indices.shape[0] * seq_len,
                dtype=torch.int32, device=cache_indices.device
            )
        else:
            self.ssm_state_indices = cache_indices

    def init_forward_metadata(self, forward_batch: ForwardBatch):
        # 跳过 draft extend 模式（其元数据由其他方式维护）
        if forward_batch.forward_mode.is_draft_extend(True):
            return
        super().init_forward_metadata(forward_batch)
        self.prepare_gdn_inputs(forward_batch.batch_size, forward_batch.forward_mode, forward_
        batch.spec_info)
        self.graph_mode = False
```

python/sglang/srt/hardware_backend/npu/attention/ascend_hybrid_linear_attn_backend.py

新增 Ascend 混合线性注意力后端基类，提供了 GDN 相关的 CUDA Graph 状态管理和 verify 模式下索引生成

```
class AscendMambaAttnBackendBase(MambaAttnBackendBase):
    """Ascend NPU 上 Mamba/混合注意力后端的基类，增加了 GDN verify 模式所需的索引管理"""

    def __init__(self, model_runner: ModelRunner):
        super().__init__(model_runner)
        self.state_indices_list_gdn = [] # 用于 GDN verify 模式下的状态索引列表

    def init_cuda_graph_state(self, max_bs: int, max_num_tokens: int):
        """初始化 CUDA Graph 状态：为 GDN verify 模式预分配临时张量"""
        assert max_num_tokens % max_bs == 0
        draft_token_num = max_num_tokens // max_bs
        for i in range(max_bs):
            # 原有 mamba 索引
            self.state_indices_list.append(
                torch.full((i + 1,), self.pad_slot_id, dtype=torch.int32, device=self.device))
            # GDN 特殊索引：每请求的索引数量为 (i+1) * draft_token_num
            self.state_indices_list_gdn.append(
                torch.full(((i + 1) * draft_token_num,), self.pad_slot_id,
                           dtype=torch.int32, device=self.device))
            self.query_start_loc_list.append(torch.zeros((i + 2,), dtype=torch.int32, device=self.
            device))
            # 以下为 eagle tree 自定义掩码所需（目前仅占位）
            self.retrieve_next_token_list.append(torch.zeros((i + 1, draft_token_num), dtype=torch.
            int32, device=self.device))
```

```
            self.retrieve_next_sibling_list.append(torch.zeros((i + 1, draft_token_num), dtype=torch.
            int32, device=self.device))
            self.retrieve_parent_token_list.append(torch.zeros((i + 1, draft_token_num), dtype=
            torch.int32, device=self.device))
        # 预计算 decode 和 verify 的 query_start_loc 缓存
        self.cached_cuda_graph_decode_query_start_loc = torch.arange(0, max_bs + 1, dtype=
        torch.int32, device=self.device)
        self.cached_cuda_graph_verify_query_start_loc = torch.arange(
            0, max_bs * draft_token_num + 1, step=draft_token_num, dtype=torch.int32, device=
            self.device)

    def _capture_metadata(
        self, bs, req_pool_indices, forward_mode, spec_info
    ):
        """捕获 CUDA Graph 元数据：填充请求索引，对 verify 模式生成 GDN 连续索引"""
        mamba_indices = self.req_to_token_pool.get_mamba_indices(req_pool_indices)
        self.state_indices_list[bs - 1][:len(mamba_indices)].copy_(mamba_indices)
        if forward_mode.is_decode_or_idle():
            self.query_start_loc_list[bs - 1].copy_(
                self.cached_cuda_graph_decode_query_start_loc[:bs + 1])
        elif forward_mode.is_target_verify():
            self.query_start_loc_list[bs - 1].copy_(
                self.cached_cuda_graph_verify_query_start_loc[:bs + 1])
            # 生成连续索引用于 verify，不依赖物理缓存顺序（因为中间状态在 verify 后会被清理）
            ssm_state_indices = torch.arange(
                mamba_indices.shape[0] * spec_info.draft_token_num,
                dtype=torch.int32, device=mamba_indices.device)
            self.state_indices_list_gdn[bs - 1][
                :len(mamba_indices) * spec_info.draft_token_num
            ].copy_(ssm_state_indices)
        else:
            raise ValueError(f"Invalid forward mode: {forward_mode=}")
        # 如果 topk > 1，需要返回 eagle tree 自定义掩码的元数据
        if forward_mode.is_target_verify() and spec_info.topk > 1:
            return ForwardMetadata(
                query_start_loc=self.query_start_loc_list[bs - 1],
                mamba_cache_indices=self.state_indices_list[bs - 1],
                retrieve_next_token=self.retrieve_next_token_list[bs - 1],
                retrieve_next_sibling=self.retrieve_next_sibling_list[bs - 1],
                retrieve_parent_token=self.retrieve_parent_token_list[bs - 1],
            )
        # 默认返回标准元数据（含 GDN 索引）
        return ForwardMetadata(
            query_start_loc=self.query_start_loc_list[bs - 1],
            mamba_cache_indices=self.state_indices_list[bs - 1],
            mamba_cache_indices_gdn=self.state_indices_list_gdn[bs - 1],
        )
```

评论区精华

1. 线程安全风险（Critical）：Gemini Code Assist 指出在 forward 方法中直接修改 os.environ 存在线程安全问题。作者随后改用 sglang.srt.environ.envs 的 ExitStack 上下文管理器，确保环境变量仅在当前线程生效。
2. GPU 兼容性影响：reviewer shengzhaotian 要求确认对 hybrid_linear_attn_backend.py 基类的修改（如 get_cuda_graph_seq_len_fill_value 返回值从 1 改为 0、新增 mamba_cache_indices_gdn 字段）不会影响 GPU 路径。作者确认这些改动为 NPU 专属，不影响 GPU。
3. 冗余计算：Gemini Code Assist 发现 forward_decode 中存在对 fused_gdn_gating 的重复调用，建议复用首次计算结果。作者后续提交删除了冗余调用（见 commit acf8284）。
4. 算子迁移：reviewer shengzhaotian 建议将 Triton kernel fused_gdn_gating_kernel_without_sigmoid 移入 sgl-kernel-npu 仓库。作者已在独立 PR 中完成迁移。

- forward 中修改 os.environ 的线程安全性 (correctness): 作者将方法改为使用 sglang.srt.environ.envs 的 ExitStack 上下文管理器，确保环境变量仅在当前线程的 forward 执行期间生效。
- fused_gdn_gating 的冗余调用 (performance): 作者后续提交删除了分支内的冗余调用。
- 对 GPU 基类的修改风险 (correctness): 作者确认这些改动因为条件隔离（只在使用 mamba_cache_indices_gdn 时生效，且 GPU 路径不会生成该字段）不影响 GPU。
- fused_gdn_gating_kernel_without_sigmoid 应移入 sgl-kernel-npu (infra): 作者已将其移入 sgl-kernel-npu 仓库（对应 PR #429）。
- MTP draft 模型量化配置影响范围 (question): 作者回复是全局生效（所有平台），因为即使 GPU 上 speculative_draft_model_quantization 为 None 时也合理。但后续增加了 is_npu() 条件限定在 NPU。

风险与影响

- 风险：
  1. GPU 基类兼容风险：对 hybrid_linear_attn_backend.py 的修改（mamba_cache_indices_gdn 参数、get_cuda_graph_seq_len_fill_value 返回值）虽然作者声称不影响 GPU，但条件覆盖不足时可能导致 GPU 路径异常。需要确认 CI 已覆盖 GPU 下的 Mamba/ 混合注意力测试。
  2. 新后端缺少基准测试：除了 GSM8K 准确率测试，未提供端到端速度基准或延迟对比，无法评估性能收益。
  3. 环境变量上下文管理：改用 ExitStack 后线程安全风险基本解除，但若下游库对 os.environ 有异步读写仍存在隐患，建议后续改为函数参数传递配置。
  4. conv state 形状变更：_init_npu_conv_state 根据 draft token 数扩展 conv state 宽度，但未考虑极端 extra_conv_len 可能导致的 OOM 风险。 - 影响：范围：仅影响 Ascend NPU 平台上的 Qwen3.5 模型，且仅在启用 --speculative-algorithm NEXTN 时生效。程度：中等。对 NPU 用户这是一个重大功能完善，使 MTP 推测解码在 NPU 上可用，提升推理吞吐。对 GPU 用户无影响，但基类抽象改动可能为后续跨平台扩展奠定

基础。

- 风险标记：GPU 基类兼容风险，新后端缺少基准测试，环境变量线程安全历史，conv state 内存可能 OOM

关联脉络

- PR #22822 [Refactor] Refactor DeepEP dispatcher: 与本 PR 共享 deepseek/moe 相关文件的修改，且都涉及 NPU 上的量化配置处理。
- PR #21668 [XPU] Enable qwen3.5 on XPU: 类似跨平台适配 Qwen3.5 模型的 PR，可对比不同硬件后端的适配模式。
- PR #25587 （Issue）MTP 解码输出 Token 数不一致问题：PR 合并后被提出的 issue，反馈 Hybrid GDN MTP 在 temperature=0 时输出 token 数不一致，可能与本 PR 的实现有关，需关注后续修复。