PR #25455 完整报告

sgl-project/sglang

[NPU] MiMo-V2-Flash Adaptation

合并时间：2026-06-10 09:13

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25455

执行摘要

- 一句话：在 Ascend NPU 上实现 MiMo-V2-Flash 多 Token 预测推理支持
- 推荐动作：该 PR 展示了 NPU 后端适配推测解码的完整流程，包括图运行器继承模式、注意力内核迁移、数据结构调整等，设计决策清晰。推荐相关开发者精读新增的 multi_layer_eagle_draft_extend_npu_graph_runner.py 以及 ascend_backend.py 中的 forward_mtp 变化，以了解 NPU 图捕获与 CUDA 图的差异以及 NPU 融合注意力内核的约束。同时 memory_pool_npu.py 的缓冲区分离模式也是对 MHA/MLA 混合场景的有益实践。

功能与动机

支持在 Ascend NPU 上运行 MiMo-V2-Flash 模型的多 Token 预测（MTP），以利用推测解码提升生成吞吐量。PR body 明确指出 'Support Multi-Token Prediction (MTP) for MiMo-V2-Flash model on Ascend NPU (requires CANN 9.0)'，并提供了端到端 accuracy 和 speed 测试结果。

实现拆解

1. 新增 NPU 图运行器：在 multi_layer_eagle_draft_extend_npu_graph_runner.py 中创建 MultiLayerEagleDraftExtendNpuGraphRunner 和 MultiLayerEagleMultiStepDraftExtendNpuGraphRunner，继承自 CUDA 图运行器基类，重写图创建、捕获初始化、捕获和回放方法以适配 NPU 架构。关键改进包括使用 torch.npu.NPUGraph 和 torch.npu.graph 上下文管理器，以及在 _replay 中通过独立线程执行 graph.update 避免阻塞。
2. 优化融合注意力内核：在 ascend_backend.py 的 forward_mtp 方法中，迁移到 npu_fused_infer_attention_score_v2 内核，支持 draft_extend_v2 模式，增加对 SWA 层和 hybrid SWA 的正确支持，传递 sinks 参数，并调整 actual_seq_lengths 计算逻辑。同时，forward_extend 和 forward_decode_graph 也进行了对应调整以支持新内核接口。
3. 分离 K/V 缓冲区：在 memory_pool_npu.py 中，将原先共享的 kv_buffer 拆分为独立的 k_buffer 和 v_buffer，使 V 缓冲区支持独立的 v_head_dim，适应 MTP 模型中 QK 和 V 维度不同的场景。同时优化了 FIA 模式下的 per-layer 视图创建，避免 torch.compile 捕获时 OOM。
4. 图运行器注册与条件分支：在 multi_layer_eagle_worker_v2.py 的 init_cuda_graphs 中，根据设备是否为 NPU 选择不同的图运行器（CUDA 或 NPU 版本），实现无侵入式的硬件适配。

5. 数据契约与兼容性调整：在 mimo_v2.py 中添加 is_ascend_fuseep() 分支以启用 NPU fused MoE EP；在 npu_graph_runner.py 中添加 if_use_v2 判断和 TARGET_VERIFY 更新属性，支持 Target Verify 模式的图更新。此外，清理了冗余的 model_runner_kv_cache_mixin.py 中的硬编码，并为全注意力窗口大小引入命名常量 FULL_ATTENTION_WINDOW 替换 magic number。

关键文件：

- python/sglang/srt/hardware_backend/npu/graph_runner/multi_layer_eagle_draft_extend_npu_graph_runner.py（模块 图运行器；类别 source；类型 dependency-wiring；符号 MultiLayerEagleDraftExtendNpuGraphRunner, init, _create_graph, _capture_init）：新增 NPU 图运行器核心文件，包含两个图 runner 类，实现了 NPU 特定图创建、捕获初始化、捕获和回放逻辑，是整个 MTP 适配的关键入口。
- python/sglang/srt/hardware_backend/npu/attention/ascend_backend.py（模块 注意力层；类别 source；类型 core-logic）：核心注意力后端文件，主要修改了 forward_mtp 方法以支持 NPU 融合注意力内核 v2，处理 draft_extend_v2 模式、SWA 层、sinks 传入等，对 MTP 正确性和性能有直接影响。
- python/sglang/srt/hardware_backend/npu/memory_pool_npu.py（模块 KV 缓存；类别 source；类型 core-logic）：K/V 缓存池的关键修改，将原先共享的 kv_buffer 拆分为独立的 k_buffer 和 v_buffer，并支持 v_head_dim 参数；这是适应 MTP 模型 QK/V 不同维度的基础数据结构变更。
- python/sglang/srt/speculative/multi_layer_eagle_worker_v2.py（模块 推测解码；类别 source；类型 dependency-wiring）：接入 NPU 图运行器的条件分支：根据 is_npu() 选择使用 CUDA 还是 NPU 图运行器，实现硬件自适应。
- python/sglang/srt/hardware_backend/npu/graph_runner/npu_graph_runner.py（模块 图运行器；类别 source；类型 core-logic）：NPU 通用图运行器修改：添加 if_use_v2 判断和 TARGET_VERIFY 更新属性，支持 Target Verify 模式的图元数据更新。
- python/sglang/srt/models/mimo_v2.py（模块 模型；类别 source；类型 data-contract）：MoE 后端适应：添加 is_ascend_fuseep() 分支，使 NPU 能够使用 fused MoE 的 expert parallelism，提升 MTP 推理效率。

关键符号：MultiLayerEagleDraftExtendNpuGraphRunner.init, MultiLayerEagleDraftExtendNpuGraphRunner._create_graph, MultiLayerEagleDraftExtendNpuGraphRunner._capture_init, MultiLayerEagleDraftExtendNpuGraphRunner._capture_graph, MultiLayerEagleDraftExtendNpuGraphRunner._replay, MultiLayerEagleMultiStepDraftExtendNpuGraphRunner._init_and_capture, forward_mtp (ascend_backend.py), NPUMHATokenToKVPool._create_buffers, NPUMHATokenToKVPool.set_kv_buffer, MultiLayerEagleDraftWorker.init_cuda_graphs (multi_layer_eagle_worker_v2.py), NPUGraphRunner._get_update_attr_name (npu_graph_runner.py)

关键源码片段

python/sglang/srt/hardware_backend/npu/graph_runner/multi_layer_eagle_draft_extend_npu_graph_runner.py

新增 NPU 图运行器核心文件，包含两个图 runner 类，实现了 NPU 特定图创建、捕获初始化、捕获和回放逻辑，是整个 MTP 适配的关键入口。

```
# 文件：multi_layer_eagle_draft_extend_npu_graph_runner.py
"""Run the multi-layer eagle draft extend model with npu graph."""

import threading
import torch
from sglang.srt.model_executor.forward_batch_info import ForwardBatch
from sglang.srt.speculative.multi_layer_eagle_draft_extend_cuda_graph_runner import (
    MultiLayerEagleDraftExtendCudaGraphRunner,
    MultiLayerEagleMultiStepDraftExtendCudaGraphRunner,
)

class MultiLayerEagleDraftExtendNpuGraphRunner(
    MultiLayerEagleDraftExtendCudaGraphRunner
):
    # NPU 专用图运行器，继承自 CUDA 版本，
    # 重写图创建、捕获、回放方法以适应 Ascend NPU 的图语义。
    def __init__(self, eagle_worker, step):
        super().__init__(eagle_worker, step)

    def _create_graph(self):
        # 创建 NPU 图对象（代替 CUDA Graph）
        return torch.npu.NPUGraph()

    def _capture_init(self, run_once_fn):
        # 预热阶段：同步两次确保状态稳定，并执行 TP barrier
        for _ in range(2):
            torch.npu.synchronize()
            self.model_runner.tp_group.barrier()
            run_once_fn()

    def _capture_graph(self, graph, pool, stream, run_once_fn):
        # 使用 torch.npu.graph 上下文管理器捕获图
        with torch.npu.graph(
            graph,
            pool=pool,
            stream=stream,
            auto_dispatch_capture=True, # 自动分发捕获
        ):
            out = run_once_fn()
        return out

    def _replay(self, forward_batch: ForwardBatch):
        # 回放前通过独立线程更新图输入的 KV 长度元数据
```

```
        seq_lens = self.buffers.seq_lens_cpu[:self.raw_bs].tolist() + [0] * (
            self.bs - self.raw_bs
        )
        thread = threading.Thread(
            target=self.graphs[self.bs].update,
            kwargs={"cpu_update_input": [{"actual_seq_kvlen": seq_lens}]},
        )
        thread.start()
        self.graphs[self.bs].replay()
        thread.join()


class MultiLayerEagleMultiStepDraftExtendNpuGraphRunner(
    MultiLayerEagleMultiStepDraftExtendCudaGraphRunner
):
    # 多步 draft extend 的 NPU 图运行器
    def __init__(self, eagle_worker):
        super().__init__(eagle_worker)

    def _init_and_capture(self):
        if self.eagle_worker.server_args.disable_cuda_graph:
            self.runners = [None] * self.speculative_num_steps
            return

        self.runners = []
        for step in range(self.speculative_num_steps):
            if self.draft_extend_attn_backend_list[step]:
                runner = MultiLayerEagleDraftExtendNpuGraphRunner(
                    self.eagle_worker, step
                )
                self.runners.append(runner)
                # ... 省略 buffer 初始化细节
            else:
                self.runners.append(None)
        # 创建公共 CPU buffer 用于 seq_lens 更新
        self.cuda_graph_buffers["seq_lens_cpu"] = torch.full(
            (self.max_bs,), self.seq_len_fill_value, dtype=torch.int32
        )
```

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

核心注意力后端文件，主要修改了 forward_mtp 方法以支持 NPU 融合注意力内核 v2，处理 draft_extend_v2 模式、SWA 层、sinks 传入等，对 MTP 正确性和性能有直接影响。

```
# ascend_backend.py 中的 forward_mtp 方法核心片段
# （已简化 import 和上下文）
def forward_mtp(self, q, k, v, layer, forward_batch,
                save_kv_cache, q_rope=None, k_rope=None, sinks=None):
    # 当需要保存 KV 缓存时写入
    if save_kv_cache:
```

```
    if self.use_mla:
        ... # MLA 分支保持不变
    else:
        # 使用 forward_batch 维护的 token_to_kv_pool 写入
        # 支持 MTP 中多步 draft 的 KV 写入
        forward_batch.token_to_kv_pool.set_kv_buffer(
            layer, forward_batch.out_cache_loc, k, v
        )

# 获取 KV 缓存 buffer
k_cache = self.token_to_kv_pool.get_key_buffer(layer.layer_id)
v_cache = self.token_to_kv_pool.get_value_buffer(layer.layer_id)

# 处理 query 维度
query = q.reshape(-1, layer.tp_q_head_num, layer.qk_head_dim).contiguous()

# 计算实际序列长度（支持 draft_extend 和 draft_extend_v2）
if forward_batch.forward_mode.is_draft_extend() or \
   forward_batch.forward_mode.is_draft_extend_v2():
    actual_seq_lengths = np.array(
        forward_batch.extend_seq_lens_cpu).cumsum().tolist()
else:
    actual_seq_lengths = [1] * (
        self.speculative_num_draft_tokens + query.shape[0],
        self.speculative_num_draft_tokens
    )

# 根据 SWA 层选择 block table
is_swa_layer = layer.sliding_window_size != -1
if is_swa_layer and self.is_hybrid_swa and \
   hasattr(self.forward_metadata, "block_tables_swa"):
    block_table = self.forward_metadata.block_tables_swa
else:
    block_table = self.forward_metadata.block_tables

# 构造 mask 和 sparse_mode
if layer.attn_type == AttentionType.ENCODER_ONLY:
    mask = None
    sparse_mode = 0
else:
    mask = self.mtp_mask
    sparse_mode = 4 if is_swa_layer else 3

# 调用 NPU 融合注意力内核 v2
attn_output, _ = torch_npu.npu_fused_infer_attention_score_v2(
    query,
    k_cache,
    v_cache,
    block_table=block_table,
```

```
            block_size=self.page_size,
            num_query_heads=layer.tp_q_head_num,
            num_key_value_heads=layer.tp_k_head_num,
            input_layout="TND",
            atten_mask=mask,
            softmax_scale=layer.scaling,
            actual_seq_qlen=actual_seq_lengths,
            actual_seq_kvlen=actual_seq_lengths_kv,
            sparse_mode=sparse_mode,
            pre_tokens=(layer.sliding_window_size if is_swa_layer
                        else FULL_ATTENTION_WINDOW),
            next_tokens=0 if is_swa_layer else FULL_ATTENTION_WINDOW,
        )

        return attn_output
```

python/sglang/srt/hardware_backend/npu/memory_pool_npu.py

K/V 缓存池的关键修改，将原先共享的 kv_buffer 拆分为独立的 k_buffer 和 v_buffer，并支持 v_head_dim 参数；这是适应 MTP 模型 QK/V 不同维度的基础数据结构变更。

```
# memory_pool_npu.py 中的 NPUMHATokenToKVPool 类核心方法

class NPUMHATokenToKVPool(MHATokenToKVPool):
    def __init__(self, size, page_size, dtype, head_num, head_dim, layer_num,
                 device, enable_memory_saver, v_head_dim=None, swa_head_num=None,
                 swa_head_dim=None, swa_v_head_dim=None, start_layer=None,
                 end_layer=None, enable_alt_stream=True, enable_kv_cache_copy=False):
        self.use_fia = get_bool_env_var("ASCEND_USE_FIA", "False")
        super().__init__(size=size, page_size=page_size, dtype=dtype,
                         head_num=head_num, head_dim=head_dim,
                         layer_num=layer_num, device=device,
                         enable_memory_saver=enable_memory_saver,
                         v_head_dim=v_head_dim,
                         swa_head_num=swa_head_num, swa_head_dim=swa_head_dim,
                         swa_v_head_dim=swa_v_head_dim,
                         start_layer=start_layer, end_layer=end_layer,
                         enable_alt_stream=enable_alt_stream,
                         enable_kv_cache_copy=enable_kv_cache_copy)

    def _create_buffers(self):
        with self.memory_saver_adapter.region(GPU_MEMORY_TYPE_KV_CACHE):
            # K 缓冲区：[layer_num, page_num+1, page_size, head_num, head_dim]
            self.k_buffer = torch.zeros(
                (self.layer_num,
                 self.size // self.page_size + 1,
                 self.page_size,
                 self.head_num,
                 self.head_dim),
                dtype=self.store_dtype,
```

```
            device=self.device,
        )
        # V 缓冲区：[layer_num, page_num+1, page_size, head_num, v_head_dim]
        self.v_buffer = torch.zeros(
            (self.layer_num,
             self.size // self.page_size + 1,
             self.page_size,
             self.head_num,
             self.v_head_dim),
            dtype=self.store_dtype,
            device=self.device,
        )

        if self.use_fia:
            # 使用 per-layer 视图避免 torch.compile 捕获整张量导致 OOM
            self.k_buffer = [
                self.k_buffer[i].view(-1, 1, self.head_num, self.head_dim)
                for i in range(self.layer_num)
            ]
            self.v_buffer = [
                self.v_buffer[i].view(-1, 1, self.head_num, self.v_head_dim)
                for i in range(self.layer_num)
            ]

    def set_kv_buffer(self, layer, cache_loc, cache_k, cache_v):
        # 使用 scatter_nd_update 写入 KV 缓存
        # 注意 v_head_dim 独立处理
        layer_id = layer.layer_id - self.start_layer
        loc = cache_loc.to(torch.int32)
        if self.memory_saver_adapter.enable_memory_saver:
            # ... 略
        else:
            # 写入 K 缓冲区
            torch_npu.npu_scatter_nd_update_(
                self.k_buffer[layer_id],
                loc.view(-1, 1),
                cache_k.view(-1, self.page_size, self.head_num, self.head_dim),
            )
            # 写入 V 缓冲区，使用 v_head_dim
            torch_npu.npu_scatter_nd_update_(
                self.v_buffer[layer_id],
                loc.view(-1, 1),
                cache_v.view(-1, self.page_size, self.head_num, self.v_head_dim),
            )
```

评论区精华

- 图模式与非图模式：Hexq0210 询问非图模式下移除 padding 截断是否有问题，作者回复已通过测试验证。

- 硬编码枚举改进：Hexq0210 要求将 'actual_seq_kvlen' 硬编码改为枚举，作者后续提交进行了修复。
- scatter_nd_update 的 v_head_dim：AndyLi429 指出在 set_kv_buffer 中未使用 v_head_dim，可能引起 dtype 不匹配；作者确认后修复。
- D2H 操作是否会中断图：McZyWu 担心 actual_seq_len_kv.cpu() 等 D2H 操作打断 CUDA 图回放，作者测试后表示无中断。
- 扩展类而非修改基类：iforgetmyname 建议不要直接修改 multi_layer_eagle_draft_extend_cuda_graph_runner.py，而是新建 NPU 类继承；作者接受并新增独立文件。
- K/V 缓冲区简化建议：gxy9808 建议移除 else 分支，统一使用独立缓冲区；Hexq0210 进一步要求使用列表推导替代 for 循环。最终采用了独立 k_buffer/v_buffer 分配且使用列表推导创建 per-layer 视图。
  - 非图模式兼容性 (correctness): 作者通过测试确认非图模式运行正常，无需特殊处理。
  - 硬编码字符串改为枚举 (design): 作者在 npu_graph_runner.py 中添加了 _init_arch_map 中的 'TARGET_VERIFY' 键，使用了类内字典管理，避免了直接字符串。
  - D2H 操作对图回放的影响 (performance): 作者确认测试没有发生图中断，但保留了潜在性能影响。
  - 建议继承而非修改基类 (design): 作者采纳，创建了独立的 multi_layer_eagle_draft_extend_npu_graph_runner.py。
  - K/V 缓冲区分离与简化 (design): 最终代码采用了独立 k_buffer/v_buffer 分配，FIA 模式下使用列表推导创建视图，消除了 else 分支。

风险与影响

- 风险：
  - 回归风险（非 NPU 后端）：mimo_v2.py 中修改了 MoE 后端的条件判断，添加了 is_ascend_fuseep() 分支。如果该函数不恰好在非 NPU 场景下返回真，可能影响 GPU 上的 MoE 行为。需确认该函数仅对 NPU 设备返回真。
  - v2 注意力内核兼容性：forward_mtp 和部分 decode 路径迁移到 npu_fused_infer_attention_score_v2 内核，该内核对输入 layout、mask 等有不同要求。讨论中 McZyWu 担忧其对其他模型（尤其是 qk_head_dim != v_head_dim 的纯 MHA 模型）的性能和正确性影响。作者已通过 is_hybrid_swa 限制非 SWA 层的行为，但仍需全面验证。
  - D2H 操作图中断风险：在 forward_decode_graph 中对非 SWA 层引入了 contiguous() 和 torch.tensor([1]...) 等 D2H 操作，作者测试没有中断，但理论上仍可能在某些 NPU 图实现中引入同步开销。
  - 内存池改动影响：K/V 缓冲区从共享 kv_buffer 拆分为独立 k_buffer/v_buffer，依赖于新布局的 get_contiguous_buf_infos 等函数如果被其他后端共享使用，可能需要适配。
  - 缺少测试覆盖：该 PR 未附带单元测试，仅提供手动 benchmark 和 accuracy 测试，可能遗漏边界情况如多 step MTP、混合 SWA 层等。
- 影响：

- 用户：在 Ascend NPU 上使用 MiMo-V2-Flash 模型时，可启用 EAGLE 推测解码（MTP v2），获得约 40% 的吞吐提升和更低的 TTFT，最大 inter-token 延迟从 8.3 s 降至 0.24 s。
- 系统：新增了一个 NPU 专用图运行器类，保持与 CUDA 图运行器的兼容接口；对 NPU 内存池进行了结构重塑，K 和 V 使用独立缓冲区并支持独立 v_head_dim。
- 团队：NPU 硬件后端团队完成了一次关键功能适配，明确了 MTP 所需的图捕获、注意力内核迁移、KV 缓存等步骤，为后续其他模型 NPU MTP 支持提供了参考模式。
- 风险标记：非 NPU 兼容性风险，v2 注意力内核回归风险，D2H 图中断风险，缺少测试覆盖

关联脉络

- PR #27607 Support spec v2 for Frozen-KV MTP; remove v1 worker: 该 PR 实现了 speculative v2 框架，本 PR 的 NPU MTP 适配依赖于 spec v2 接口（如 draft_extend_v2、target_verify 等），两者在同一功能演进线路上。