

PR #38807 完整报告

sgl-project/sglang

[NPU]glm5.2 fp8 memory opt

合并时间: 2026-09-11 18:48

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38807>

执行摘要

- 一句话: NPU 分离式下释放 MLA 预处理冗余权重换显存
- 推荐动作: 建议精读, 尤其是正在做 NPU/MLA 显存优化的同学: 这是一个 "22 行改动、两个关键设计点" 的典型案列。

值得关注的设计决策:

1. 把判定提成 `uses_mlaprolog()`: 破坏性释放必须与运行时路由共用同一判定, 这个重构不是风格问题而是正确性前提, 值得作为同类改动的模板。
2. 按分离式角色分档释放: decode 释放 `w_kc`、prefill 保留, 体现了对 absorption 路径使用面的精细区分; 但需要作者补充说明不对称的理由。
3. 审慎跟进点: `qkv_weight.data.untyped_storage().resize_(0)` 作用于 `qkv_a_proj` 权重视图, 请确认模型不支持权重热更新 /LoRA, 且 `NPUFusedMLAPreprocess` 只在 `mlaprolog` 路径构造; 同时确认 `get_disagg()` 在权重预处理阶段已就绪, 否则优化会静默失效。

建议后续补一个最小验证: 在 `disaggregated decode + mlaprolog` 配置下跑一次端到端精度对比与显存采样, 证明释放生效且输出不变; prefill 侧是否需要同样释放 `w_kc` 也值得单独评估。

功能与动机

PR body 中 Motivation 与 Modifications 使用了同一句话: "Certain weights can be released after use to increase the available NPU memory.", 标题点明目标是 `[NPU]glm5.2 fp8 memory opt`, 即在 GLM5.2 FP8 的 NPU 部署中把「用完的权重」显式归还给设备显存。从代码看, 被释放的对象是两类 `mlaprolog` 路径不再消费的权重: `mlaprolog_preprocess_weight()` 里已由 `npu_format_cast` 生成独立 NZ 副本的原始 `qkv_weight`, 以及 absorption 已被 `npu_mla_prolog_v3` 融合后不再读取的 `m.w_kc`。PR 未关联 Issue, Accuracy Tests / Speed Tests 均标注 None。

实现拆解

1. 抽出 `mlaprolog` 判定, 作为释放与路由的唯一真相 (`python/sglang/srt/hardware_backend/npu/attention/mla_preprocess.py`): 把原先内联在 `NPUFusedMLAPreprocess.forward()` 中的 `_is_arch35_dsa or (hasattr(self.quant_config, "ignore") and any(re.fullmatch(r".*kv_b_proj", l) for l in self.quant_config.ignore))` 提取为新方法 `uses_mlaprolog()`, `forward()` 改为

`_is_mlaprolog = self.uses_mlaprolog()`。这样做的直接原因是：本 PR 新引入的是不可逆的 `resize_(0)`，如果「按 A 判定释放、按 B 判定使用」两处逻辑漂移，就会读到空 storage；抽成同一方法从结构上排除了这个隐患。

2. 权重预处理阶段回收原始 qkv_a 权重（同文件 `mlaprolog_preprocess_weight()`）：在 `self.q_a_proj_weight / self.kv_a_proj_weight` 已经通过 `npu_format_cast(qkv_weight[:, : q_lora_rank].contiguous())` 等生成独立副本之后，新增 `if get_disagg().disaggregation_mode != "null": qkv_weight.data.untyped_storage().resize_(0)`。注意条件覆盖 `prefill` 与 `decode` 两种分离角色（只要不是 `null`），因为副本已独立，原始张量在任意角色下都无消费者。
3. 调用点按角色回收 `w_kc`（python/`sglang/srt/hardware_backend/npu/modules/deepseek_v2_attention_mla_npu.py`）：在 `npu_mla_preprocess()` 首次惰性构造 `m.mla_preprocess = NPUFusedMLAPreprocess(...)` 之后，新增 `if get_disagg().disaggregation_mode == "decode" and m.mla_preprocess.uses_mlaprolog() and m.w_kc is not None: m.w_kc.untyped_storage().resize_(0)`。之所以限定 `decode` 且要求 `uses_mlaprolog()` 为真，是因为 `w_kc` 仍被 `forward_absorb_prepare_npu_rms_norm_cache` 等非 `mlaprolog` 分支使用，只有在 `mlaprolog` 融合了 `absorption` 的路径上才能安全丢弃。
4. 清理与依赖调整（同文件）：删除已无用的 `import re`（判定逻辑内聚到 `uses_mlaprolog()`），新增 `from sglang.srt.runtime_context import get_disagg; npu_mla_preprocess()` 中原先内联的 `_is_mlaprolog = (_is_npu_arch35 and get_token_to_kv_pool().index_head_dim is not None) or ...` 整体删除，改用 `m.mla_preprocess.uses_mlaprolog()`。模块级 `_is_npu_arch35 = is_npu_arch35()` 常量与导入保留（文件中仍有其它使用点），因此不存在残留未使用导入。
5. 配套缺失：无测试文件、无 benchmark 数据、无文档更新；PR 检查清单五项全部未勾选，body 中三条 PR Test 流水线（Base / Extra / AMD ROCm 10）均为失败状态，最终带 `bypass-fastfail` 标签由 `iforgetmyname` 合并。

关键文件：

- python/`sglang/srt/hardware_backend/npu/attention/mla_preprocess.py`（模块 注意力预处理；类别 source；类型 core-logic；符号 `NPUFusedMLAPreprocess.uses_mlaprolog`, `NPUFusedMLAPreprocess.mlaprolog_preprocess_weight`, `NPUFusedMLAPreprocess.forward`）：核心实现文件：新增 `uses_mlaprolog()` 把 `mlaprolog` 判定单点化，并在 `mlaprolog_preprocess_weight()` 末尾对原始 `qkv_weight` 执行 `untyped_storage().resize_(0)`，是本次显存收益与全部风险的主要来源。
- python/`sglang/srt/hardware_backend/npu/modules/deepseek_v2_attention_mla_npu.py`（模块 注意力装配；类别 source；类型 dependency-wiring；符号 `npu_mla_preprocess`）：调用点改造：首次构造 `NPUFusedMLAPreprocess` 时按 `disagg` 角色决定是否释放 `m.w_kc`，并用 `uses_mlaprolog()` 替换内联判定（同时删除 `import re`、新增 `get_disagg` 导入）。

关键符号： `NPUFusedMLAPreprocess.uses_mlaprolog`,
`NPUFusedMLAPreprocess.mlaprolog_preprocess_weight`,
`NPUFusedMLAPreprocess.forward`, `npu_mla_preprocess`

关键源码片段

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

核心实现文件：新增 `uses_mlaprolog()` 把 mlaprolog 判定单点化，并在 `mlaprolog_preprocess_weight()` 末尾对原始 `qkv_weight` 执行 `untyped_storage().resize_(0)`，是本次显存收益与全部风险的主要来源。

```
# python/sglang/srt/hardware_backend/npu/attention/mla_preprocess.py
from sglang.srt.runtime_context import get_disagg # 新增：读取 PD 分离角色 (null / prefill / decode)

class NPUFusedMLAPreprocess:
    def mlaprolog_preprocess_weight(self):
        # ... 省略 fp8 (weight_quant_mode == 3) / int8 (mode == 1) 分支的 scale 与 q_b_proj_weight 预处理 ...
        # 下面两步把 qkv_a_proj 权重按 q_lora_rank 切成 q_a / kv_a 两段，
        # npu_format_cast 会生成带独立 storage 的 NZ 布局副本，因此原始 qkv_weight 此后再无消费者
        self.q_a_proj_weight = npu_format_cast(
            qkv_weight[:, : self.q_lora_rank].contiguous()
        )
        self.kv_a_proj_weight = npu_format_cast(
            qkv_weight[:, self.q_lora_rank :].contiguous()
        )

        # 本次新增：分离式部署 (prefill 或 decode 任一角色) 下直接把底层 storage resize 成 0,
        # 等于把这块权重显存立刻还给 NPU；非分离式 (null) 走原逻辑不做释放
        # 注意：qkv_weight 是 self.qkv_a_proj.weight.data 的转置视图，本行会一并清空参数自身的 storage
        if get_disagg().disaggregation_mode != "null":
            qkv_weight.data.untyped_storage().resize_(0)

    def uses_mlaprolog(self) -> bool:
        # 本次新增：把原先内联在 forward() 里的判定提成方法，
        # 让「运行时路由」与「调用方决定是否释放权重」共用同一份判定，避免两处逻辑漂移
        _is_arch35_dsa = (
            self.is_npu_arch35 and get_token_to_kv_pool().index_head_dim is not None
        )
        return _is_arch35_dsa or (
            hasattr(self.quant_config, "ignore")
            and any(re.fullmatch(r".*kv_b_proj", l) for l in self.quant_config.ignore)
        )

    def forward(self, positions, hidden_states, forward_batch, zero_allocator):
        # assert self.quant_config and self.quant_config.get_name() == "modelslim"
        # route by `qkv_a_proj` quant type as MTP layers can be unquantized
        _is_w8a8 = (
```

```

        hasattr(self.qkv_a_proj.quant_method, "quantization_config")
        and self.qkv_a_proj.quant_method.quantization_config.get_name()
        == "modelslim"
    )
    _is_arch35_dsa = (
        self.is_npu_arch35 and get_token_to_kv_pool().index_head_dim is not None
    )
    # with the mlaprolog enabled, the kv_b_proj layers are unquantized
    _is_mlaprolog = self.uses_mlaprolog() # 行为不变，只是改为复用公共判定
    if _is_w8a8 and not _is_arch35_dsa:
        return self.forward_mlapo(
            positions, hidden_states, forward_batch, zero_allocator
        )
    elif _is_mlaprolog:
        return self.forward_mlaprolog(positions, hidden_states, forward_batch)
    else:
        return self.forward_absorb_prepare_npu_rms_norm_cache(
            positions, hidden_states, forward_batch, zero_allocator
        )

```

python/sglang/srt/hardware_backend/npu/modules/deepseek_v2_attention_mla_npu.py

调用点改造：首次构造 `NPUFusedMLAPreprocess` 时按 `disagg` 角色决定是否释放 `m.w_kc`，并用 `uses_mlaprolog()` 替换内联判定（同时删除 `import re`、新增 `get_disagg` 导入）。

```

# python/sglang/srt/hardware_backend/npu/modules/deepseek_v2_attention_mla_npu.py
from sglang.srt.runtime_context import get_disagg # 新增导入；同时删除了不再需要的 import re

```

```

def npu_mla_preprocess(
    m: "DeepseekV2AttentionMLA",
    hidden_states: torch.Tensor,
    positions: torch.Tensor,
    forward_batch: "ForwardBatch",
    zero_allocator: "BumpAllocator",
):
    dynamic_scale = None
    if not hasattr(m, "mla_preprocess"):
        # 惰性构造：只有真正走到这条路径才会创建，因此释放动作也只会发生一次
        m.mla_preprocess = NPUFusedMLAPreprocess(
            m.fused_qkv_a_proj_with_mqa,
            m.q_a_layernorm,
            m.kv_a_layernorm,
            m.q_b_proj,
            m.w_kc, # w_kc 由 mlaprolog 之外的 absorption 分支继续使用，不能无条件释放
            m.rotary_emb,
            m.layer_id,
            m.num_local_heads,
            m.qk_nope_head_dim,

```

```

        m.qk_rope_head_dim,
        m.v_head_dim,
        m.quant_config,
    )
    # 本次新增：仅分离式 decode + mlaprolog + w_kc 存在时，才把 w_kc 的 storage 清空。
    # decode 侧 mlaprolog 已把 absorption 融进 npu_mla_prolog_v3, w_kc 不再被读取；
    # 先用 uses_mlaprolog() 兜住，避免在仍需要 w_kc 的路径上误释放
    if (
        get_disagg().disaggregation_mode == "decode"
        and m.mla_preprocess.uses_mlaprolog()
        and m.w_kc is not None
    ):
        m.w_kc.untyped_storage().resize_(0)

# mlaprolog does not require additional calculation of q_lora
if m.mla_preprocess.uses_mlaprolog():
    # 与上面的释放判定同源，保证「释放了什么」与「接下来走哪条分支」始终一致
    (
        q_pe,
        k_pe,
        q_nope_out,
        k_nope,
        q_lora,
        forward_batch,
        positions,
        dynamic_scale,
    ) = m.mla_preprocess.forward(
        positions, hidden_states, forward_batch, zero_allocator
    )
else:
    # 非 mlaprolog 分支（走 forward_absorb_prepare_npu_rms_norm_cache 等）仍依赖 w_kc,
    # 正因为这里没有被释放，该路径才能正常工作
    if m.alt_stream is not None:
        mla_event = torch.npu.Event()
        mla_event.record()
        with torch.npu.stream(m.alt_stream):
            # alt stream waits for the completion of the event on the main stream to ensure
            # data dependency is complete
            torch.npu.current_stream().wait_event(mla_event)
            (
                q_pe,
                k_pe,
                q_nope_out,
                k_nope,
                forward_batch,
                zero_allocator,
                positions,
            ) = m.mla_preprocess.forward(

```

```
positions, hidden_states, forward_batch, zero_allocator
)
```

评论区精华

本 PR 几乎没有技术层面的评审：两次 review 均为 `iforgetmyname` 的 APPROVED 且 body 为空，review_comments 为 0，仅有两条 Issue 评论——`iforgetmyname` 的 `/tag-run-ci-label` 与作者 `Liwansi` 的 `/rerun-failed-ci`。也就是说，`untyped_storage().resize(0)` 这种不可逆操作、以及 `prefill/decode` 释放范围不对称（`qkv_weight` 在 `!= "null"` 即释放，`w_kc` 只在 `== "decode"` 释放）都没有在评审中被讨论或记录理由，风险完全依赖代码结构本身（`uses_mlaprolog()` 单点判定 + `disagg` 角色门控）来自证。

- 不可逆的 `resize(0)` 释放缺少评审确认 (design): 未讨论。安全性完全依赖 `uses_mlaprolog()` 单点判定加 `get_disagg().disaggregation_mode` 门控，没有任何书面约定或测试来固化这一前提。
- CI 三条流水线失败仍带 `bypass-fastfail` 合并 (testing): 合并时没有可用的自动化绿灯，改动只在真实 Ascend 环境 (GLM5.2 FP8) 上做过人工验证，缺少可复现的测试或 benchmark 证据。
- 释放范围在 `prefill / decode` 之间不对称 (question): 未说明。可能是 `prefill` 侧仍存在需要 `w_kc` 的路径（或验证不足），但也可能只是保守取值，存在未兑现的显存优化空间，建议作者补充依据。

风险与影响

- 风险：
 1. 破坏性释放不可逆，且在共享 storage 上执行：`mha_preprocess.py` 可见分支中 `qkv_weight = self.qkv_a_proj.weight.data.transpose(0, 1)` 是参数本身的视图，`qkv_weight.data.untyped_storage().resize(0)` 实际清空的是 `qkv_a_proj` 权重的底层 storage。虽然 `q_a_proj_weight / kv_a_proj_weight` 已由 `npu_format_cast` 生成独立副本，但任何后续对该参数或视图的访问（权重更新 / 热加载、LoRA 挂载、MTP 层复用、非 `mlaprolog` 回退）都会直接报错或读到无效内存。
 2. 释放判定依赖运行时上下文时序：`get_disagg()` 取自 `sglang.srt.runtime_context.mlaprolog_preprocess_weight()` 属于权重预处理（模型构建期），若此时 `disaggregation mode` 尚未注入，条件恒为假，优化会静默失效（不报错、只缺省省显存），排查成本高；而在 `npu_mha_preprocess()` 中该判定发生在 `forward` 期，时序上更安全。
 3. `prefill/decode` 释放范围不对称：`w_kc` 只在 `decode` 释放，而 `qkv_weight` 在 `prefill` 也会释放。若 `prefill` 侧存在同一 `NPUFusedMLAPreprocess` 实例被复用或后续需要 `w_kc` 的路径，则解码侧省显存、预填充侧不省，收益与预期不一致；反之若 `prefill` 也安全，则存在未兑现的优化空间。代码与 PR body 都没有解释。
 4. 缺乏验证证据：无单测、无精度对比、无显存 / 吞吐数据，且 CI (Base / Extra / AMD ROCm 10) 在快照时全为失败，合并依赖 `bypass-fastfail`，意味着该改动是在没有自动化绿灯的前提下进入 main 的。

5. 缓存化判定的副作用: `is_mla_preprocess_enabled()` / `is_npu_arch35()` 等为

`lru_cache` 或模块级常量, `uses_mlaprolog()` 现在被调用两次 (释放判定 + 路由), 每次都会重新读 `get_token_to_kv_pool().index_head_dim`, 若 KV pool 在该时刻未初始化, 判定结果可能与后续不一致。

- 影响: 影响面集中在 Ascend NPU + MLA 架构模型 (GLM5.2 FP8、DeepSeek 系列 MLA/MTP) 的部署:
 - 对分离式 decode 实例: 释放 `w_kc` 与原始 `qkv_weight` 后可用显存增加, 理论上可提升 KV cache 容量 / 并发, 是本次改动的直接受益者; 层数越多、每层 MLA 权重越大 (FP8 大模型), 累计收益越明显。
 - 对单机 (`disaggregation_mode == "null"`) 部署: 仅 `uses_mlaprolog()` 判定被重构, 行为等价, 无显存变化、无功能影响。
 - 对 prefill 实例: `qkv_weight` 会被释放, `w_kc` 保留, 行为介于两者之间, 需要实测确认无回退。
 - 对系统与团队: 改动只触及两个 NPU 后端文件, 接口无变化, 不会影响 CUDA/ROCm/XPU 路径; 但引入了一种「按运行时角色释放权重」的新范式, 后续 NPU 显存优化可能沿用, 值得团队建立相应的约定 (何时可释放、如何验证)。
 - 对用户: 不改变输出语义 (前提是释放判定正确), 失败模式是崩溃而非静默算错, 因此风险以可用性为主。
 - 风险标记: 核心路径变更, 破坏性显存释放, 缺少测试覆盖, CI 未全绿合并, 平台特定分支

关联脉络

- PR #38827 [NPU][Hicache] Add Ascend Memcache Hicache L3 storage backend: 同属 Ascend NPU 的显存 / KV 分层存储优化脉络: 一个从分级存储扩容, 一个从权重侧腾显存, 都在解决 NPU 上 KV cache 空间受限的问题。
- PR #38954 [Refactor] Generalize DeepSeek V4 compressed pool management: 同样涉及 DeepSeek MLA 系列在 NPU 上的内存池管理 (改到 `hardware_backend/npu/dsv4/dsv4_memory_pool.py`), 与本 PR 的「按角色精算显存」属于同一条 NPU 显存治理方向。
- PR #38993 [Refactor] Generalize attention graph variants in the decode runner: 本 PR 的 `uses_mlaprolog()` 中包含 arch35 DSA 分支 (依赖 `get_token_to_kv_pool().index_head_dim`), 与 #38993 抽取 MLA/DSA 注意力变体契约的工作同源, 都是把易漂移的变体判定收敛成单一可复用入口。