PR #35701 完整报告

sgl-project/sglang

[diffusion] feat: let offloaded weights stay on the checkpoint mapping

合并时间：2026-08-21 10:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35701

执行摘要

- 一句话：offload 权重保留 checkpoint 映射，H3 可在 32 GB 内存加载
- 推荐动作：值得精读。重点关注三点：MappedRegions 如何用一次 /proc/self/maps 快照把“tensor 是否在文件映射”的判定从二次方降到对数复杂度；host_copies_would_not_fit 如何统一 DiT 与 encoder 两条路径的容量判定；以及把 MPS 专用零拷贝逻辑推广为通用“拷贝放不下”策略的权衡。若未来要优化大模型低内存部署，本 PR 是很好的范本。

功能与动机

PR body 明确指出："Layerwise offload copies every layer into host memory at startup, so host memory has to be at least as large as the component. MiniMax-H3's DiT is 61.73 GiB, which rules out a 12 GiB + 32 GiB budget outright -- not by a slow margin, but at initialization." 实测显示 50.53 GiB 的 DiT 权重到达 offload 管理器时仍是 checkpoint 的视图（_can_assign_cpu_tensor_without_copy 直接复用已加载 tensor），二次拷贝才是装不下的原因。

实现拆解

1. 统一内存预算判定：host_memory_budget.py 新增 HOST_COPY_RESERVE_BYTES = 4 * GIB_BYTES 与 host_copies_would_not_fit(weight_bytes)，用 weight_bytes >= host_memory_available_bytes() - HOST_COPY_RESERVE_BYTES 判断“拷贝放不下”，把 DiT 路径和 encoder 路径各自重复的 4 GiB reserve 与比较逻辑收敛到一处。
2. 文件映射识别基础设施：loader/utils.py 新增 MappedRegions 类，一次性读取 /proc/self/maps 快照，用二分查找回答“指针 / tensor 是否在文件映射中”；替换 component_residency_bytes 内联的 _read_process_mappings 逻辑，避免对 H3 数万权重逐个读 /proc 的二次方开销。
3. 分层卸载新增第三种 host 存储：layerwise_offload.py 的 LayerwiseOffloadManager 新增 _mapped_cpu_weights 与 _mapped_bytes，存放仍引用 checkpoint 映射的权重；_keep_weights_on_their_mapping 决定是否启用该路径。三个关键方法同步调整：sync_layer_to_cpu 跳过 mapped 权重（防止写触发 copy-on-write 变成匿名内存）、prefetch_layer 直接从映射拷贝且不再把 consolidated store 当作“是否有权重”的 gate、iter_cpu_weights 仍产出 mapped 权重以保证 checksum 可见。
4. 文本编码器接入同一策略：text_encoder_loader.py 新增 _checkpoint_bytes（加载任何权重前即可读取磁盘大小）与 _keep_this_checkpoint_mapped；将

_mps_zero_copy_weight_loading 重命名为 _keep_checkpoint_mapping，mps_zero_copy_unsafe 重命名为 checkpoint_mapping_unsafe，条件从 current_platform.is_mps() 扩展为"MPS 或拷贝放不下"；minimax_h3.py、minimax_h3_qwen3vl.py、fsdp_load.py 同步适配新标志。

5. 测试配套：test_layerwise_offload.py 新增 4 个用例（文件映射权重保留、可负担时仍拷贝、映射权重不被写回、checksum 仍可见映射权重），test_host_memory_budget.py 新增 TestHostCopiesWouldNotFit 5 个用例（覆盖 H3 62 GiB 场景、reserve 不可动用、cgroup cap 优先于机器内存）。

关键文件：

- python/sglang/multimodal_gen/runtime/loader/utils.py（模块 映射识别；类别 source；类型 core-logic；符号 MappedRegions, init, available, holds_pointer）：新增 MappedRegions 类，用 /proc/self/maps 一次性快照回答 tensor 是否位于文件映射，替换 component_residency_bytes 内联逻辑，是整条映射保留路径的识别基础。
- python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py（模块 分层卸载；类别 source；类型 core-logic；符号 _keep_weights_on_their_mapping）：核心变更文件：新增 _mapped_cpu_weights 第三类 host 存储与 _keep_weights_on_their_mapping 决策入口，并同步调整 sync_layer_to_cpu、prefetch_layer、iter_cpu_weights 三个关键方法。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/text_encoder_loader.py（模块 编码器加载；类别 source；类型 dependency-wiring；符号 _checkpoint_bytes, _keep_this_checkpoint_mapped）：将文本编码器的零拷贝逻辑从 MPS 专用（_mps_zero_copy_weight_loading）扩展为通用 _keep_checkpoint_mapping，新增 _checkpoint_bytes 与 _keep_this_checkpoint_mapped，使 H3 的 62.13 GiB encoder 也能在 32 GiB 主机加载。
- python/sglang/multimodal_gen/runtime/managers/memory_managers/host_memory_budget.py（模块 内存预算；类别 source；类型 core-logic；符号 host_copies_would_not_fit）：新增 host_copies_would_not_fit 与 HOST_COPY_RESERVE_BYTES，把 DiT 与 encoder 两条路径重复的 4 GiB reserve 和比较逻辑统一为单一判定入口。
- python/sglang/multimodal_gen/test/unit/test_layerwise_offload.py（模块 卸载测试；类别 test；类型 test-coverage；符号 _FileBackedBlock, init, _FileBackedModel, _mapped_manager）：新增 4 个测试覆盖映射保留路径的核心行为：文件映射权重保留、可负担时仍拷贝、映射权重不被写回、checksum 仍可见映射权重。
- python/sglang/multimodal_gen/test/unit/test_host_memory_budget.py（模块 内存预算测试；类别 test；类型 test-coverage；符号 _host_has, TestHostCopiesWouldNotFit, test_a_checkpoint_larger_than_the_host_does_not_fit, test_a_checkpoint_well_under_the_host_fits）：新增 TestHostCopiesWouldNotFit 5 个用例，验证 H3 62 GiB 场景判定、reserve 不可动用、cgroup cap 优先于机器内存等边界条件。
- python/sglang/multimodal_gen/runtime/loader/fsdp_load.py（模块 FSDP 加载；类别 source；类型 core-logic）：适配零拷贝标志从 _mps_zero_copy_weight_loading 到

_keep_checkpoint_mapping 的重命名与控制流调整。

- python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py（模块 H3 模型；类别 source；类型 data-contract）：DiT 模型侧同步 _mps_zero_copy_weight_loading 到 _keep_checkpoint_mapping 的标志名变更。
- python/sglang/multimodal_gen/runtime/models/encoders/minimax_h3_qwen3vl.py（模块 H3 编码器；类别 source；类型 data-contract）：H3 文本编码器侧同步零拷贝标志重命名，与 text_encoder_loader 的新逻辑保持一致。

关键符号：MappedRegions.init, MappedRegions.available, MappedRegions.holds_pointer, MappedRegions.holds, LayerwiseOffloadManager._keep_weights_on_their_mapping, host_copies_would_not_fit, _checkpoint_bytes, _keep_this_checkpoint_mapped, component_residency_bytes

关键源码片段

python/sglang/multimodal_gen/runtime/loader/utils.py

新增 MappedRegions 类，用 /proc/self/maps 一次性快照回答 tensor 是否位于文件映射，替换 component_residency_bytes 内联逻辑，是整条映射保留路径的识别基础。

```
# python/sglang/multimodal_gen/runtime/loader/utils.py

class MappedRegions:
    """判断一个 tensor 的字节是否位于文件映射中。

    查找表来自 /proc/self/maps 的一次性快照：若每个 tensor 都重新读取该
    文件，在 H3 这种拥有数万 weight 的 checkpoint 上是 O(n^2) 的开销。
    这是一个快照而非实时视图，构造之后新建立的映射对它不可见。
    """

    def __init__(self) -> None:
        # Linux only；`_read_process_mappings` 在 /proc 不可用时返回 None
        self._maps = _read_process_mappings()

    @property
    def available(self) -> bool:
        # /proc 缺失时返回 False，调用方只能把 tensor 当作匿名内存处理
        return self._maps is not None

    def holds_pointer(self, pointer: int) -> bool:
        # 二分定位指针所在区间，再判断该区间是否文件映射
        if self._maps is None or pointer == 0:
            return False
        starts, ends, backed = self._maps
        index = bisect.bisect_right(starts, pointer) - 1
        if index < 0 or pointer >= ends[index]:
            return False
        return backed[index]
```

```
    def holds(self, tensor: torch.Tensor) -> bool:
        # 只关心 CPU tensor，取 untyped storage 的 data_ptr 查表；
        # 空 offload 占位符 data_ptr 为 0，`holds_pointer` 直接返回 False
        if tensor.device.type != "cpu":
            return False
        try:
            return self.holds_pointer(tensor.untyped_storage().data_ptr())
        except Exception:
            return False
```

python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py

核心变更文件：新增 _mapped_cpu_weights 第三类 host 存储与 _keep_weights_on_their_mapping 决策入口，并同步调整 sync_layer_to_cpu、prefetch_layer、iter_cpu_weights 三个关键方法。

```
# python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py
# 第三种 host 存储：与 consolidated、strided 并列，保存仍直接引用
# checkpoint 文件映射的权重，由 page cache 决定哪些页面驻留内存。

# layer_idx -> {name: 仍指向 checkpoint 文件的 tensor}
self._mapped_cpu_weights: Dict[int, Dict[str, torch.Tensor]] = {}
self._mapped_bytes = 0

# 在权重加载完成后立刻取一次映射快照，用于后续分类
self._mapped_regions = MappedRegions()

def _keep_weights_on_their_mapping(self, layer_groups: Dict) -> bool:
    """是否把文件映射上的权重留在映射里而不是拷进 host 内存。

    拷贝能换来 pinning，pinning 能让拷贝流领先计算（实测 Wan2.1 每步
    1.03 s vs 1.90 s），所以要非常谨慎地放弃；只有当拷贝放不下时才
    选择留在映射上——H3 的 DiT 有 61.73 GiB 权重，其中 50.53 GiB 本就
    是 checkpoint 的视图，32 GiB 主机上根本无法完成拷贝。
    """
    if not self._mapped_regions.available:
        return False
    # 统计 layer_groups 中位于文件映射内的 weight 字节数，再交给
    # host_copies_would_not_fit 统一判定；后续决策会填充
    # `_mapped_cpu_weights` 并跳过对 mapped 权重的写回。
    mapped_bytes = ...
```

python/sglang/multimodal_gen/runtime/managers/memory_managers/host_memory_budget.py

新增 host_copies_would_not_fit 与 HOST_COPY_RESERVE_BYTES，把 DiT 与 encoder 两条路径重复的 4 GiB reserve 和比较逻辑统一为单一判定入口。

```
# python/sglang/multimodal_gen/runtime/managers/memory_managers/host_memory_budget.py
```

```
# 评估 checkpoint 与 host 内存时预留的空闲量：激活值、staging 缓冲和
# 分配器余量都不在 weight 总量里，这部分不能挪给拷贝用。
HOST_COPY_RESERVE_BYTES = 4 * GIB_BYTES


def host_copies_would_not_fit(weight_bytes: int) -> bool:
    """把 `weight_bytes` 拷贝进 host 内存是否会把 host 打满。

    拷贝的替代方案是把权重留在文件映射上，让内核在内存压力下丢弃页面、
    用时再从磁盘读回。逐字节更慢但有界，所以这个答案恰好是
    “拷贝放不下”时为真，放得下时为假。
    """
    if weight_bytes <= 0:
        return False
    return weight_bytes >= host_memory_available_bytes() - HOST_COPY_RESERVE_BYTES
```

评论区精华

本 PR 没有 review 评论，唯一 Issue 评论是作者触发的 /tag-and-rerun-ci。设计权衡在 PR body 中完整阐述：pinning 让拷贝流领先计算（实测 Wan2.1 每步 1.03 s vs 1.90 s），而文件映射无法 pin，只能退到计算流拷贝；映射带宽与 pinned 接近（12.38 GB/s vs 13.39 GB/s），真正放弃的是重叠而非带宽。结论是两条 gate 都设为“拷贝放不下”才启用映射路径，宿主内存充足时行为完全不变。

- 保留文件映射 vs 拷贝进 pinned 内存的权衡 (design): 两条 gate 都设为“拷贝放不下”才启用映射路径，因此有足够内存的宿主仍走原来的 pinned 路径，行为不变。

风险与影响

- 风险：
  1. 运行时性能回退风险（layerwise_offload.py）：映射路径下拷贝在计算流执行，失去与计算的 overlap；若 page cache 在内存压力下驱逐页面，推理时缺页需从磁盘重读，可能导致延迟抖动。PR 未提供 H3 上的运行时实测。
  2. 快照失效风险（loader/utils.py）：MappedRegions 是构造函数时刻的 /proc/self/maps 快照，加载之后新建立的映射不可见；若调用时机在权重加载完成前，分类会失准，可能漏判文件映射权重。
  3. 写回语义变更：sync_layer_to_cpu 跳过 mapped 权重，若未来有代码原地修改权重，则修改会静默失效（写入会触发 copy-on-write，把映射变成匿名内存，破坏本设计初衷）。
  4. 预算判定近似（host_memory_budget.py）：host_copies_would_not_fit 用 checkpoint on-disk 大小近似实际驻留，4 GiB reserve 为常数，在临近阈值时可能误判；cgroup 与 psutil 交叉判定依赖 #35707 的修复。
  5. 回退路径：/proc 不可用时 available 为 False，逻辑回退到原拷贝路径，安全性可接受，但行为不一致。 - 影响：对用户：MiniMax-H3 首次能在 12 GiB VRAM + 32 GiB 主机上完成加载，DiT 的 host 内存占用从 61.73 GiB 降到 12.20 GiB，encoder 从

46.18 GiB 降到 2.22 GiB，加载期峰值 VRAM 5176 MiB、host 23.42 GB。对系统：serving 主机因两条 gate 默认关闭，行为与之前完全一致；MPS 平台保持原有零拷贝语义。对团队：offload 管理器新增 _mapped_cpu_weights 数据契约与 _keep_weights_on_their_mapping 决策入口，后续需要补充 H3 运行时 benchmark 验证 page cache 驻留效果。 - 风险标记：核心加载路径变更，缺页 / 换出性能风险，MappedRegions 为加载时快照，H3 运行时性能未实测，依赖 /proc/self/maps

关联脉络

- PR #35707 [diffusion] read the cgroup this process is actually in: 同一 host_memory_budget.py 与 test_host_memory_budget.py 的近期修复，为本 PR 的 host_copies_would_not_fit 提供可用的 cgroup 内存上限判定基础。
- PR #35698 [diffusion] Fuse LTX-2.5 decoder 3D RoPE: 同为 multimodal_gen runtime 的 diffusion 性能优化，说明该区域正在持续演进。