PR #33880 完整报告

sgl-project/sglang

[diffusion] optimization: reduce minimax h3 mps memory pressure

合并时间：2026-08-19 13:35

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33880

执行摘要

- 一句话：H3 MPS 内存优化：权重分阶段卸载、前向分块流式
- 推荐动作：值得精读，尤其 _forward_mps_streamed_attention 是“分块流式计算规避统一内存峰值”的典型范例，LayerwiseOffloadManager 对同步 / 异步双路径的抽象也有借鉴价值。阅读时建议对照 commit 历史中多次 revert 的 chunk size 调参过程，理解 MPS 内存 - 吞吐权衡的敏感性；但由于 Draft 合并且 CI Extra 失败，合入后应优先补齐端到端精度验证，再在真实 MPS 设备上推广。

功能与动机

MiniMax H3 的 packed 长序列和大组件工作集在 MPS 统一内存上压力极大。PR body 的动机原文是："MiniMax-H3's long packed sequence and large component working set need stricter placement on MPS unified memory." 具体而言，融合 QKV 投影输出在 768px 分辨率下约 1.45 GiB，与 packed residual、SDPA workspace 同时驻留易挤占系统内存；同时 MPS 没有 CUDA 式 pinned memory 与独立 copy stream，原 layerwise offload 的异步预取与 Event 同步模型并不适用。

实现拆解

实现按五步展开，全部落在 multimodal_gen 运行时，CUDA 路径仅做兼容性收口：

1. LayerwiseOffloadManager 的 MPS 化改造（python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py）：__init__ 新增 initialize 参数使初始化可延迟；识别 _synchronous_mps 后禁用 pinned memory 与 copy stream，prefetch_size 显式保持 0 避免下一层与当前层同时驻留；新增 _mps_cpu_weights 字典保留每层原始 CPU tensor，_initialize_mps_cpu_weights 把权重迁回 CPU 并用共享空 tensor 占位；prefetch_layer 在 MPS 分支做同步搬移，release_layer 强制 torch.mps.synchronize() + empty_cache()，sync_layer_to_cpu 直接跳过（推理不改参数）。同时新增 _capture_mps_cpu_non_layer_weights/materialize_mps_non_layer_weights/release_mps_non_layer_weights/restore_mps_cpu_non_layer_weights 管理条件投影、时间嵌入等非逐层权重。
2. 加载链路先 CPU 后设备（transformer_loader.py、vae_loader.py、text_encoder_loader.py、fsdp_load.py）：三个 loader 新增 customized_load_kwargs_for_component，在 MPS 且组件被配置为 layerwise offload 时返回 cpu_offload_flag=True；load_customized 接受该参数，checkpoint_load_device

强制为 CPU，WeightLoadPlan.for_component 增加 mps_layerwise_cpu_staging 标志；VAE loader 用 load_state_dict(..., assign=True) 直接复用 CPU 原始 tensor，避免加载后再拷一份。

3. DiT 前向分块流式化（python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py、sdpa.py）：_forward_mps_streamed_attention 第一阶段按 128 token chunk 生成完整 K/V 且不保留 Q，第二阶段逐序列逐 chunk 投影 Q、立即过 attention 与 out_proj 回写；MLP 与最终投影（video_out/audio_out）同样按 128 chunk 分块；_apply_rope 从 _apply_rope_qk 中抽取 eager 路径；qkv 权重标记 mps_zero_copy_unsafe=True 表明 checkpoint 行交错需重排、不可零拷贝；forward 入口以 x.device.type == "mps" and not ulysses_active 切到流式路径。
4. 准入与平台参数收紧（pipeline_configs/minimax_h3.py、server_args.py）：validate_server_args 在 MPS 下强制 transformer/text_encoder/video_vae/audio_vae 全部为 LAYERWISE_OFFLOAD 并拒绝 torch.compile；_adjust_platform_specific 收窄为仅允许 num_gpus=1、residency 只支持 RESIDENT 或 LAYERWISE_OFFLOAD，移除原先“MPS 强制关闭 layerwise offload”的逻辑。
5. 平台杂项与配套测试：gpu_worker.py 改 capture_memory_snapshot() 统计峰值（max_memory_reserved() 在 MPS 不可用）并跳过 set_device 与 reset_peak_memory_stats；vae_vit.py 与 decoding.py 用 _cuda_autocast_disabled/nullcontext 让 autocast 只在 CUDA 生效；hf_diffusers_utils.py 为 client has been closed 增加指数退避重试；test_minimax_h3_admission.py 新增 MPS 准入用例 test_mps_admission_requires_layerwise_residency_for_every_h3_component。

关键文件：

- python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py（模块 权重卸载；类别 source；类型 dependency-wiring；符号 initialize, _initialize_mps_cpu_weights, _capture_mps_cpu_non_layer_weights, _matches_mps_weight_prefix）：整个 PR 的核心：LayerwiseOffloadManager 从纯 CUDA 异步模型扩展为同步 MPS 与异步 CUDA 双路径，新增 _mps_cpu_weights 结构、MPS 初始化 / 取层 / 释放 / 非层权重生命周期管理，是所有 MPS 组件驻留行为的地基。
- python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py（模块 H3 模型；类别 source；类型 data-contract；符号 _apply_rope, _forward_mps_streamed_attention）：DiT 前向的 MPS 流式执行主体：新增 _forward_mps_streamed_attention、MLP 与最终投影的 128 token 分块路径、_apply_rope 抽取和 mps_zero_copy_unsafe 标记，直接决定内存峰值下降幅度。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/transformer_loader.py（模块 组件加载；类别 source；类型 core-logic；符号 customized_load_kwargs_for_component）：DiT 组件在 MPS 下先加载到 CPU 的入口：customized_load_kwargs_for_component 下发 cpu_offload_flag，load_customized 据此强制 checkpoint_load_device 为 CPU 并配置 mps_layerwise_cpu_staging。
- python/sglang/multimodal_gen/runtime/loader/component_loaders/vae_loader.py（模块 组件加载；类别 source；类型 core-logic；符号 customized_load_kwargs_for_compone

nt）：VAE 组件同路径改造，关键差异是 load_state_dict 使用 assign=True 以复用 CPU 原始 tensor，避免 MPS 下额外拷贝。

- python/sglang/multimodal_gen/configs/pipeline_configs/minimax_h3.py（模块 配置校验；类别 source；类型 dependency-wiring）：MPS 准入校验：强制 transformer/text_encoder/video_vae/audio_vae 全部 LAYERWISE_OFFLOAD，并拒绝 torch.compile，防止未经验证的组合在 MPS 上运行。
- python/sglang/multimodal_gen/runtime/server_args/server_args.py（模块 启动参数；类别 source；类型 core-logic）：解除 MPS 对 layerwise offload 的旧限制并新增 num_gpus=1 约束，是平台参数层的策略翻转，支撑本 PR 的 MPS 驻留方案。
- python/sglang/multimodal_gen/runtime/managers/gpu_worker.py（模块 工作进程；类别 source；类型 dependency-wiring）：MPS 适配的必要杂项：跳过 set_device 与 reset_peak_memory_stats，内存峰值改用 capture_memory_snapshot()，因为 MPS 下 max_memory_reserved() 不可用。
- python/sglang/multimodal_gen/runtime/models/vaes/minimax_h3_video_vae/vae_vit.py（模块 VAE 模型；类别 source；类型 data-contract；符号 _cuda_autocast_disabled）：autocast 平台适配：_cuda_autocast_disabled 让 CUDA 之外的设备（含 MPS）走 nullcontext，避免 MPS 上触发不受支持的 autocast。
- python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/minimax_h3/stages/decoding.py（模块 解码阶段；类别 source；类型 data-contract）：解码阶段的 autocast 同样改为 CUDA 专属，MPS 下用 nullcontext 包裹 VAE decode。
- python/sglang/multimodal_gen/test/unit/test_minimax_h3_admission.py（模块 准入测试；类别 test；类型 test-coverage；符号 test_mps_admission_requires_layerwise_residency_for_every_h3_component）：新增 MPS 准入测试，锁定 MiniMax H3 在 MPS 下必须全组件 layerwise offload 的契约，防止后续参数调整破坏该约束。

关键符号：LayerwiseOffloadManager.initialize, LayerwiseOffloadManager._initialize_mps_cpu_weights, LayerwiseOffloadManager.prefetch_layer, LayerwiseOffloadManager.release_layer, LayerwiseOffloadManager.sync_layer_to_cpu, LayerwiseOffloadManager.materialize_mps_non_layer_weights, LayerwiseOffloadManager.release_mps_non_layer_weights, MiniMaxH3DiTModel._forward_mps_streamed_attention, _apply_rope, TransformerLoader.customized_load_kwargs_for_component, VAELoader.customized_load_kwargs_for_component

关键源码片段

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

整个 PR 的核心：LayerwiseOffloadManager 从纯 CUDA 异步模型扩展为同步 MPS 与异步 CUDA 双路径，新增 _mps_cpu_weights 结构、MPS 初始化 / 取层 / 释放 / 非层权重生命周期管理，是所有 MPS 组件驻留行为的地基。

```
# layerwise_offload.py 节选：MPS 同步层权重的卸下与取回
# 前置状态：__init__ 识别到 current_platform.is_mps() 时，
```

```
# 会设置 _synchronous_mps=True、pin_cpu_memory=False、copy_stream=None，
# 并用 _mps_cpu_weights 字典存放每层原始 CPU tensor。

@torch.compiler.disable
def _initialize_mps_cpu_weights(self) -> None:
    """把每层权重迁回 CPU，并在设备上替换为共享空 tensor 占位。

    CUDA 路径会把同层同 dtype 的权重合并成扁平化的 pinned CPU 副本；
    MPS 与主机共享物理内存，直接保留原始 CPU tensor 即可，避免再驻留
    一份主机拷贝。
    """
    for name, tensor in self._named_parameters.items():
        layer_idx = self._match_layer_idx(name)
        if layer_idx is None or layer_idx >= self.num_layers:
            continue
        local_tensor = self._to_local_tensor(tensor).detach()
        cpu_tensor = (
            local_tensor
            if local_tensor.device.type == "cpu"
            else local_tensor.to("cpu")
        )
        self._mps_cpu_weights.setdefault(layer_idx, {})[name] = cpu_tensor
        self._weight_metadata.setdefault(layer_idx, {})[name] = {
            "dtype": cpu_tensor.dtype,
        }
        # 用共享的 1 元素占位 tensor 顶替原参数，释放设备侧大块显存
        tensor.data = self._get_shared_empty_tensor_for_target(
            tensor, cpu_tensor.dtype
        )

    torch.mps.empty_cache()
    self.register_forward_hooks()
    self._configured = True
    logger.info(
        f"Initialized synchronous MPS layerwise offload with "
        f"{self.num_layers} layers"
    )


def prefetch_layer(self, layer_idx: int, non_blocking: bool = True) -> None:
    """幂等地把指定层权重搬回设备；MPS 下是同步搬移。"""
    if not self.enabled or self.device is None:
        return
    if layer_idx < 0 or layer_idx >= self.num_layers:
        return
    if layer_idx in self._gpu_layers:
        return

    if self._synchronous_mps:
```

```
        cpu_weights = self._mps_cpu_weights.get(layer_idx)
        if not cpu_weights:
            return
        with torch.inference_mode(False), torch.no_grad():
            for name, cpu_tensor in cpu_weights.items():
                target = self.get_target_with_name(name)
                # 统一内存下 CPU 到 MPS 的搬移没有异步语义，non_blocking 恒为 False
                target.data = self._wrap_for_target(
                    target,
                    cpu_tensor.to(device=self.device, non_blocking=False),
                )
        self._gpu_layers.add(layer_idx)
        return

    # 以下是 CUDA 异步路径：pinned 副本 + copy stream + Event 记录
    # MPS 分支已在上方返回，CUDA 路径行为保持不变
```

评论区精华

本 PR 没有任何 reviewer 评论，唯一交互是作者在 Issue 评论区触发 /tag-and-rerun-ci 重跑 CI。PR body 明确标注 Draft 状态并声明“end-to-end and accuracy validation are pending”，且 CI Extra 运行（Run #32208109060）状态为失败，说明正确性验证尚未收敛；这些未决事项直接决定了本 PR 的合入风险等级。

- 暂无高价值评论线程

风险与影响

- 风险： 1) 正确性未验证：PR body 明说 end-to-end and accuracy validation are pending，CI Extra 失败；MPS 流式 attention 按 chunk 重算 norm 与 RoPE，其等价性依赖 _apply_rope 与 CUDA 路径逐字节一致，建议补齐精度对照后再在生产启用。 2) 性能同步开销：每个 chunk 后强制 torch.mps.synchronize() + torch.mps.empty_cache()，同步点密集；最终 chunk size 128 来自 30 个 commit 中多次“Increase/Batch”后被 revert、再“Lower/Tune”的反复试验，说明对具体机型内存 - 吞吐权衡高度敏感。 3) 全局下载重试改动：hf_diffusers_utils.py 新增 client has been closed 的捕获与退避重试，修改 snapshot_download 的错误语义，影响所有平台的模型拉取路径。 4) 内存统计口径变化：gpu_worker.py 在 MPS 下改用 snapshot 峰值，与 CUDA 的 max_memory_reserved() 口径不同，跨平台对比数据时易产生误读。 5) 双份权重字典契约：_mps_cpu_weights 与 _consolidated_cpu_weights 并存，update_cpu_weights 等写回路径需同时维护两套结构，后续扩展时容易漏改一侧。
- 影响：影响面集中在 MPS（Apple Silicon）平台的 MiniMax H3 推理：所有分支均以 current_platform.is_mps() 或 x.device.type == "mps" 为条件，CUDA/ROCm 主路径行为不变；唯一全局影响是 Hugging Face 下载重试逻辑。对用户而言，MiniMax H3 在 MPS 上首次具备可运行性且可支撑更长序列；对团队而言，LayerwiseOffloadManager 从此承载同步（MPS）与异步（CUDA）两套执行模型，后续新增平台需同时考虑两条分支；由于 Draft 且 CI Extra 失败，该能力目前处于“可运行但未验证”状态，不建议直接

作为发布基线。

- 风险标记：正确性未验证（Draft），CI Extra 失败，MPS 专用路径，CUDA 不受影响，全局 Hugging Face 下载重试改动，同步点密集，吞吐受限，chunk size 经验调参敏感

关联脉络

- PR #34993 [diffusion] fix: make MiniMax-H3 AdaLN cache rebuild transactional: 同改 python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py，同为 MiniMax H3 在 diffusion 运行时的稳定性补强；本 PR 引入 MPS 流式执行，34993 修复 AdaLN 缓存重建，两条链路共同保障 H3 推理正确与稳定。
- PR #35339 [diffusion] Per-request lossy accelerations: Cache-DiT, CFG gating, attention backend override: 同改 MiniMax H3 的 denoising 阶段与采样配置，说明 H3 pipeline 正在被系统性优化；本 PR 关注 MPS 平台内存，35339 关注按请求精度开关，方向互补。