

PR #38783 完整报告

sgl-project/sglang

[Diffusion] Quiet request-path cache diagnostics

合并时间: 2026-09-11 22:45

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38783>

执行摘要

- 一句话: 把 diffusion 请求路径的 5 处缓存诊断日志由 info 降为 debug
- 推荐动作: 值得快速浏览, 但不值得精读。三点建议: (1) 合并前确认 diffusion 侧已有 metrics 或看板覆盖 built_plans / host_hit_plans / host_pressure_skips, 否则这次降级等于把唯一的信号源关掉; (2) 在运行手册中登记“需要开启 DEBUG 才能看到 AdaLN host tier / layerwise paged-out 诊断”, 避免排障时误判功能缺失; (3) 后续若有独立 PR 为这三个计数器补充遥测导出 (可参考 PR#38782 把 telemetry 暴露到 nightly 看板的做法), 可以把观测能力补齐, 届时本 PR 的降级才真正无代价。

功能与动机

PR body 明确给出三点动机: 一是把 layerwise residency 的页回收诊断从 info 移到 debug; 二是把 MiniMax-H3 AdaLN 缓存的主机内存压力与缓存命中诊断从 info 移到 debug; 三是把 Helios 请求级质量融合挂载诊断从 info 移到 debug。归结起来就是 body 中的那句 “preserve startup, warning, and error visibility while avoiding per-request log noise” —— 这些日志都挂在单次请求的执行路径上 (每次缓存跳过、每次 rebuild、每次 host 命中、每次阶段换出), 在 diffusion 在线服务下会随请求量线性增长, 属于典型的日志噪声而非故障信号。

实现拆解

1. MiniMax-H3 AdaLN 主机缓存的 3 处请求级日志降级 (`python/sglang/multimodal_gen/runtime/models/dits/minimax_h3_adaln_cache.py`)。分别位于 `_skip_for_pressure()` (内存压力下跳过某组 plan 的缓存)、`build()` 尾部 (重建 plan 后打印 rebuild 次数与常驻比例)、`_swap_in_from_host()` 尾部 (从主机固定页换入 plan)。三处都保留前置的 `self.stats` 计数 (`host_pressure_skips`、`built_plans`、`host_hit_plans`), 因此降级的只是文本输出, 缓存命中、重建与压力跳过的量化观测能力不变。这样改的原因是这三条路径在稳态下每次请求都可能触发, 属于“正常但频繁”, 不是异常。
2. Helios 质量融合挂载日志降级 (`python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/helios_denoising.py`)。改动点只在 `HeliosChunkedDenoisingStage._maybe_toggle_quality_fusions()` 中 `mount_helios_gated_residual()` 成功分支的 `logger.debug`。注意这里的 `want == self._quality_fusions_mounted` 提前 return 保证了挂载 / 卸载只在质量档位真正切换时发生, 日志本身频率并不高, 但仍然被归入“请求路径诊断”而一起降级; `unmount` 分支本来就

没有日志，因此挂载与卸载的日志对称性没有变化。

3. Layerwise 逐层换出的页回收诊断降级（`python/sglang/multimodal_gen/runtime/managers/memory_managers/component_residency_strategies.py`）。位于 `finish_use()` 中 CUDA 共享主机内存池分支：在 `empty_cache()` 之后，遍历 `module.layerwise_offload_managers` 调用 `advise_mapped_pages_cold(room_bytes=...)` 统计 `paged_out`，只有 `paged_out` 非零时才打印。降为 debug 后，这条“本组件页被主动标冷”的信息默认不可见，但 `SHARED_POOL_NEXT_STAGE_HEADROOM_BYTES` 参与的 `room_bytes` 计算逻辑与冷页建议调用本身完全未动。
4. 测试、配置与部署配套：无。本 PR 未新增或修改任何测试文件、CI 配置或部署脚本，也没有 schema / 接口契约变化（`change_kind` 中标注的 `data-contract` 仅对应日志文案参数，不涉及数据格式）。
5. 分支与合并动作：head 分支为 `codex/diffusion-cache-log-hygiene`，共 2 个 commit —— 主体提交 `chore(diffusion): quiet request-path cache diagnostics` 和一次 `Merge main to pick up IPC JIT recovery for diffusion CI`，后者说明作者在 CI 失败后选择同步 main 而不是修改本 PR 的代码。

关键文件：

- `python/sglang/multimodal_gen/runtime/models/dits/minimax_h3_adaln_cache.py`（模块 缓存诊断；类别 source；类型 logging；符号 `_skip_for_pressure`, `build`, `_swap_in_from_host`）：本 PR 改动最集中的文件，涉及请求路径上最多的日志降级（3 处），也是 AdaLN 主机缓存可观测性的关键位置。
- `python/sglang/multimodal_gen/runtime/managers/memory_managers/component_residency_strategies.py`（模块 常驻策略；类别 source；类型 core-logic；符号 `finish_use`）：layerwise 逐层换出策略中唯一一处请求路径诊断日志被降级，涉及 CUDA 共享主机内存池的冷页建议路径。
- `python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/helios_denoising.py`（模块 去噪阶段；类别 source；类型 logging；符号 `_maybe_toggle_quality_fusions`）：Helios chunked denoising 阶段按请求质量档位挂载 / 卸载 `gated residual` 的日志被降级，是本次请求路径日志治理的一部分。

关键符号： `_skip_for_pressure`, `build`, `_swap_in_from_host`, `_maybe_toggle_quality_fusions`, `finish_use`

关键源码片段

`python/sglang/multimodal_gen/runtime/models/dits/minimax_h3_adaln_cache.py`

本 PR 改动最集中的文件，涉及请求路径上最多的日志降级（3 处），也是 AdaLN 主机缓存可观测性的关键位置。

```
class _MiniMaxH3AdaLNHostTier:
    # ... 省略 host slab 分配与 group 管理部分 ...

    def _skip_for_pressure(self, group_key) -> None:
```

"""内存压力下放弃把某组 plan 写入主机缓存时调用。

这条日志原本是 info 级，而它在稳态下会随请求频率反复触发，因此本 PR 改为 debug：默认安静，需要时再开 DEBUG 观察。注意 self.stats.host_pressure_skips 仍然照常累加，量化观测能力没有随日志级别一起消失。

"""

```
if self.stats is not None:
    self.stats.host_pressure_skips += 1
logger.debug(
    "MiniMax H3 AdaLN host tier: group of %d plan(s) skipped under "
    "memory pressure; it will recompute on its next occurrence",
    len(group_key),
)
```

class _MiniMaxH3AdaLNCache:

... 省略 slot 分配、checkpoint 投影与 host tier 交互部分 ...

def build(self, *, wanted, missing, group_key) -> None:

"""按需重建缺失的 plan，并把整组 plan 写回主机缓存。"""

pending_slots = self._allocate_slots(wanted=wanted, missing=missing)

... 省略 adaln 输入准备、ColumnParallelLinear 分片投影与异常回滚 ...

Commit host metadata only after every layer has been written.

self._slots.update(pending_slots)

self.rebuilds += 1

self.stats.built_plans += len(missing)

rebuild 属于缓存淘汰后的正常重算路径，不是异常；

降为 debug 以免在请求路径上每请求刷一行，

真实重建次数仍由 stats.built_plans / self.rebuilds 记录。

```
logger.debug(
    "MiniMax H3 AdaLN: rebuilt %d plan(s), %d/%d resident, pass #%"
    "d",
    len(missing),
    len(self._slots),
    self.max_plans,
    self.rebuilds,
)
```

def _swap_in_from_host(self, *, wanted, missing) -> None:

"""从固定主机页换入缺失的 plan，不需要再读权重。"""

assert self._host_tier is not None

assignments = self._allocate_slots(wanted=wanted, missing=missing)

... 省略 H2D 拷贝、fence_gpu_reads 与异常时归还 slot ...

self._slots.update(assignments)

self.stats.host_hit_plans += len(missing)

host 命中同样是逐请求事件，降为 debug；

命中量可从 stats.host_hit_plans 读出。

```

logger.debug(
    "MiniMax H3 AdaLN: %d plan(s) from the host cache, %d/%d resident",
    len(missing),
    len(self._slots),
    self.max_plans,
)

```

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

layerwise 逐层换出策略中唯一一处请求路径诊断日志被降级，涉及 CUDA 共享主机内存池的冷页建议路径。

```

def finish_use(
    self,
    module: nn.Module,
    use: ComponentUse,
    state: ResidencyState,
) -> None:
    """组件本阶段使用结束，释放逐层窗口并把本组件的页标为冷页。"""
    # ... 省略 release_all / park_non_layer_weights 与 MPS 分支 ...

elif current_platform.is_cuda() and current_platform.device_shares_host_memory():
    # 阶段内流式的逐层窗口已经释放，但仍被 caching allocator 持有；
    # 在共享池上这份占用就是下一阶段映射所需的 page cache，先交还。
    empty_cache = getattr(torch.get_device_module(), "empty_cache", None)
    if empty_cache is not None:
        empty_cache()

    # 下一阶段到来前，本组件的页是缓存里价值最低的一批；
    # 可留给它的空间 = 当前可用空间 - 中间阶段需要的余量。
    room_bytes = max(
        0, shared_pool_available_bytes() - SHARED_POOL_NEXT_STAGE_HEADROOM_BYTES
    )
    paged_out = 0
    for manager in module.layerwise_offload_managers:
        advise_cold = getattr(manager, "advise_mapped_pages_cold", None)
        if advise_cold is not None:
            paged_out += int(advise_cold(room_bytes=room_bytes) or 0)

    if paged_out:
        # 该分支在每次阶段切换时都可能走到，属于稳态路径而非异常；
        # 本 PR 把这条诊断降到 debug，避免请求路径噪声，
        # 但 advise_mapped_pages_cold 的调用与 room_bytes 计算完全未改。
        logger.debug(
            "Layerwise offload: paged out the first %.1f GiB of %s so the "
            "next request's stream fits the %.1f GiB the cache can give it.",
            paged_out / 1024**3,
            use.component_name,
            room_bytes / 1024**3,

```

)

python/sglang/multimodal_gen/runtime/pipelines_core/stages/model_specific_stages/helios_denoising.py

Helios chunked denoising 阶段按请求质量档位挂载 / 卸载 gated residual 的日志被降级，是本次请求路径日志治理的一部分。

```
class HeliosChunkedDenoisingStage(PipelineStage):
    """Helios 分块去噪阶段，按请求质量档位切换 gated-residual 快路径。"""

    def __init__(self, transformer, scheduler):
        super().__init__()
        self.transformer = transformer
        self.scheduler = scheduler
        self._quality_fusions_mounted = False

    def _maybe_toggle_quality_fusions(self, batch: Req) -> None:
        # 仅当 quality 为 "extra-high"/"high" 时挂载请求级的 Helios
        # gated-residual 快路径；"lossless" 保持参考实现的 FP32 乘法，
        # 以保证逐位一致。
        quality = getattr(batch.sampling_params, "quality", "lossless")
        want = quality_allows_kernel_fusions(quality)
        # 档位未变化时提前返回，避免每个请求都重复挂载 / 卸载。
        if want == self._quality_fusions_mounted:
            return
        self._quality_fusions_mounted = want
        if self.transformer is None:
            return
        if want:
            if mount_helios_gated_residual(self.transformer):
                # 挂载成功原本走 info，本 PR 降为 debug：
                # 该信息只在质量档位切换时出现，属于诊断而非故障，
                # 默认保持安静；mount 失败本身仍无日志（行为未变）。
                logger.debug(
                    "Mounted Helios per-token gated residual for quality=%s", quality
                )
        else:
            # 卸载分支本来就不打日志，因此挂载 / 卸载的日志行为依旧不对称。
            unmount_helios_gated_residual(self.transformer)
```

评论区精华

本 PR 没有任何 review 评论（[review_comments_count](#) 为 0，Review 审核列表为空），唯一的交互是作者自己在 Issue 评论里发出的 [/tag-and-rerun-ci](#)。有效信息大多来自 PR body 中的 CI 状态块：Latest PR Test (Base) 通过，而 Latest PR Test (Extra) 与 Latest PR Test (AMD ROCm 10) 均为失败标记，最终由作者本人（merged_by: mickqian）合并。结合第二个 commit 的“Merge main to pick up IPC JIT recovery for diffusion CI”，可以判断失败的两条流水线更可能是 diffusion JIT 构建相关的基础设施问题，而非本 PR 的日志改动所致

，但 PR 内没有留下明确的失败归因讨论。

- CI 失败与 /tag-and-rerun-ci: 日志改动是否需要重跑验证 (testing): 没有留下失败归因的正式讨论。就改动内容看，5 行日志级别调整不可能影响内核构建或 ROCm 路径，作者通过同步 main 带入 JIT 修复后直接合并，可视为将失败归因于基础设施问题。
- 缺少 review 意见的日志级别变更 (other): 变更由作者自行合并 (merged_by 与 author 均为 mickqian)，属于典型的低风险自治合并；观测替代方案 (stats 计数器) 未在 PR 中被显式确认。

风险与影响

- 风险:

1. 可观测性回退 (主要风险) : minimax_h3_adaln_cache.py 的 build() 与 _swap_in_from_host() 两条日志原本是排查“plan 反复重建 / host 缓存是否生效”的第一手信号。降为 debug 后，运维如果仍然依赖 grep stdout 观察主机缓存行为，将直接失去这三个维度；必须改用 stats.built_plans / host_hit_plans / host_pressure_skips 三个计数器 (这些字段在 diff 中保留) 或显式把日志级别开到 DEBUG。
1. 性能回归被掩盖的窗口: rebuilds 与 rebuild pass # 是判断缓存是否退化为“每请求全量重建”的关键线索。降级之后，即使出现 rebuild 频率异常升高，也不再有 info 级提示，可能延迟发现主机缓存失效类问题。同理，component_residency_strategies.finish_use() 中 paged_out 长期为零 (冷页建议失效) 也不会再被日志暴露。
2. 日志级别不一致的隐患: _skip_for_pressure() 与同文件 _evict_oldest_group() 属于同一语义层级，后者只累加统计不打日志，前者现在也降为 debug，一致性反而更好；但同一仓库其它模块 (如 HiCache 相关路径) 仍可能保留 info 级缓存诊断，跨模块排障时级别语义不完全统一。
3. 缺少测试覆盖: 无测试变更，日志级别本身难以被单测锁定，未来若有人误改回 info 或删除 stats 计数，CI 不会报警。
4. 兼容性风险极低: 无 API、参数、序列化格式或内核变更，改动范围仅 5 行日志调用，回滚成本接近于零。
 - 影响: 对使用者 (部署 diffusion 服务的工程师): 默认 stdout 变得干净，不再被逐请求的 AdaLN 主机缓存与 layerwise 换出信息刷屏，长跑与压测场景下日志体积和 I/O 开销小幅下降；代价是查看这些诊断需要主动设置 DEBUG 日志级别或接入 metrics。对系统: 请求路径上每次缓存跳过 / 命中 / 重建的字符串格式化开销被省掉，属于顺带的微小性能收益，不改变任何缓存、内存或调度语义。对团队: 本 PR 与同期的 diffusion 可观测性工作 (把指标搬到看板) 方向一致，意味着团队对 diffusion 运行时的观测入口正在从“读日志”转向“读指标/看板”，需要同步更新排障手册中涉及这三条日志的说明。
 - 风险标记: 可观测性回退，无测试覆盖，CI Extra/ROCm 失败未归因，缓存退化信号被静音

关联脉络

- PR #39034 [Diffusion] Recover IPC JIT initialization after interrupted builds: 本 PR 的第二个 commit 明确写着“Merge main to pick up IPC JIT recovery for diffusion CI”，说明该修复被用来解开 diffusion CI 的失败，是本 PR 能合并的前提。

- PR #38782 [Diffusion][CI] Expose nightly server telemetry coverage: 同属 diffusion 可观测性主线: 该 PR 把服务端遥测暴露到 nightly 看板, 本 PR 则把请求路径的日志噪声收掉, 两者共同推动观测入口从日志转向指标 / 看板。
- PR #39036 [ROCm] Raise HiCache JIT block quota for mapped-host throughput: 同样涉及扩散 / HiCache 主机内存映射层的诊断与调优语义, 可与本 PR 中 AdaLN host tier 与 layerwise 页回收日志降级对照阅读。