

PR #37266 完整报告

sgl-project/sglang

[diffusion] MiniMax-H3: tiered AdaLN plan cache (pinned-host tier + per-plan LRU)

合并时间: 2026-09-03 22:09

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/37266>

执行摘要

- 一句话: 为 MiniMax-H3 AdaLN 在线重建添加分层缓存, 减少重复请求的检查点读取延迟。
- 推荐动作: 该 PR 值得精读, 特别是分层缓存设计和权重更新安全钩子。关键决策包括: 组级 LRU 而非计划级 LRU (因为重建传递流式所有层), 主机同步保证 rank 一致性, 以及通过 BaseDiT 钩子实现模型无关的验证。适用于需要高吞吐的多调度部署场景。

功能与动机

在 `--minimax-h3-adaln-online` 模式下, 任何计划未命中都会从检查点重新读取所有 24.2 GiB 的 `adaln_proj` 权重 (nsys 测量每次请求 GPU 空闲 5.8-6.7 秒), 且 slab 溢出会触发完全重置, 导致两个交替的 50 步计划在每个请求上重新读取检查点。PR body 指出:

“Repeated-schedule requests drop 94.8 -> 88.4 s on 1x H200 with bit-identical outputs”, 表明动机是消除冗余 I/O 以提升吞吐。

实现拆解

1. 缓存模块重构: 将 `MiniMaxH3AdalnCache` 类从 `minimax_h3.py` 移至新文件 `minimax_h3_adaln_cache.py`, 添加 `MiniMaxH3AdalnHostTier` 类实现固定主机缓存层, 包含组级 LRU 驱逐、引用计数和 H2D/D2H 流同步。
2. GPU 层优化: 修改 `MiniMaxH3AdalnCache` 的驱逐策略为每计划 LRU, 替换原来的全 slab 重置; 在 `release_metadata.py` 中添加准入检查, 拒绝超出 GPU 计划槽位的请求。
3. 计划槽位解析: `prepare_adaln_plans` 方法现在在主机上解析计划槽位并返回设备张量, `block_all()` 使用单次 `block_all()` 替代 50 次 per-block 收集, 消除捕获破坏性主机同步。
4. 权重更新安全钩子: 在 `BaseDiT` 中新增 `validate_weight_update_source`、`validate_lora_layers` 和 `refresh_weight_derived_caches` 方法, `weights_updater.py` 在写入权重前调用这些钩子, 拒绝不兼容的更新。
5. CLI 与配置: 新增 `--minimax-h3-adaln-host-cache-gb` 参数 (默认 8 GB), 环境变量 `SGLANG_DIFFUSION_MINIMAX_H3_ADALN_GPU_PLANS` 和 `SGLANG_DIFFUSION_MINIMAX_H3_ADALN_FP32` 作为专家调优旋钮; 测试文件扩展覆盖缓存行为、LRU、交换和拒绝场景。

关键文件:

- `python/sglang/multimodal_gen/runtime/models/dits/minimax_h3_adaln_cache.py` (模块缓存层; 类别 `source`; 类型 `data-contract`; 符号 `_plan_key`,

native_adaln_weight_files, _fp32_gemm_guard, MiniMaxH3AdalnCacheStats)：新增核心缓存模块，实现分层缓存、LRU 驱逐、H2D/D2H 同步和统计收集。

- python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py（模块 模型主文件；类别 source；类型 data-contract；符号 _plan_key, MiniMaxH3AdalnCache, init, load）：修改主模型文件，移除内联缓存代码，改为导入新模块，并添加缓存槽位解析逻辑。
- python/sglang/multimodal_gen/runtime/models/dits/base.py（模块 模型基类；类别 source；类型 data-contract；符号 validate_weight_update_source, validate_lora_layers, refresh_weight_derived_caches）：在基类中添加权重更新验证钩子，为模型提供安全更新接口。
- python/sglang/multimodal_gen/runtime/server_args/server_args.py（模块 配置参数；类别 source；类型 configuration；符号 _validate_minimax_h3_adaln）：添加新 CLI 参数和验证逻辑，控制主机缓存大小。
- python/sglang/multimodal_gen/runtime/post_training/weights_updater.py（模块 权重更新；类别 source；类型 dependency-wiring）：修改权重更新器，集成验证钩子，确保更新前检查兼容性。
- python/sglang/multimodal_gen/test/unit/test_minimax_h3_adaln_cache.py（模块 缓存测试；类别 test；类型 test-coverage；符号 _weights_fill, _reference_block, test_host_tier_swap_in_restores_evicted_plans_bit_exactly）：测试文件扩展，覆盖缓存行为、LRU、交换和拒绝场景。

关键符号：_plan_key, MiniMaxH3AdalnHostTier.init, MiniMaxH3AdalnHostTier.store_group, MiniMaxH3AdalnHostTier.copy_to_gpu, MiniMaxH3AdalnCache.build, MiniMaxH3AdalnCache.resolve_slots, MiniMaxH3DiTModel.prepare_adaln_plans, BaseDiT.validate_weight_update_source, BaseDiT.validate_lora_layers, BaseDiT.refresh_weight_derived_caches

关键源码片段

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

新增核心缓存模块，实现分层缓存、LRU 驱逐、H2D/D2H 同步和统计收集。

```
# MiniMaxH3AdalnHostTier: 固定主机缓存层，存储已构建的 AdaLN 计划，按组 LRU 驱逐
# 关键设计：每页存储一个计划条目 (block + final 参数)，引用计数管理，同步提交保证 rank 一致性
```

```
class MiniMaxH3AdalnHostTier:
```

```
    """Bounded pinned-host store of built AdaLN plans, group-LRU evicted."""
```

```
    def __init__(self, *, num_layers: int, block_width: int, final_width: int, capacity_bytes: int,
                 device: torch.device) -> None:
```

```
        # 计算每页大小 (字节数)，基于层数、block_width 和 final_width
```

```
        self.block_numel = num_layers * block_width
```

```
        self.page_numel = self.block_numel + final_width
```

```
        page_bytes = self.page_numel * 2 # bf16 存储
```

```
        # 分配固定主机页，容量为 MIN(capacity_bytes, 可用预算)
```

```

budget = _host_cache_budget_bytes()
num_pages = max(0, min(capacity_bytes, budget)) // page_bytes
self._slab, self.pinned, num_pages = _allocate_host_slab(num_pages=num_pages, page_
numel=self.page_numel)
# 跨 rank 同步页数, 保证多 rank 一致
num_pages = _all_ranks_min(num_pages, device=device)
self.num_pages = num_pages
self._free_pages = list(range(num_pages))
# 计划键到 _HostPlan 的映射, 组级 LRU 用于驱逐
self._plans: dict[tuple[int, ...], _HostPlan] = {}
self._groups: OrderedDict[tuple[tuple[int, ...], ...], tuple] = OrderedDict()
# 使用 CUDA 流进行 H2D/D2H 异步复制
self._stream = torch.cuda.Stream(device=device) if device.type == "cuda" and self.pinned
else None

def store_group(self, *, group_key, entries: dict, block_params: torch.Tensor, final_params:
torch.Tensor) -> None:
    """D2H-copy a freshly built group into host pages and commit it."""
    # 同步提交: 复制完成后才使组可见, 保证 rank 状态确定性
    if self._stream is not None:
        self._stream.wait_stream(torch.cuda.current_stream())
    staged: dict[tuple[int, ...], _HostPlan] = {}
    with self._stream_ctx():
        for key, (slot, length) in new_plans.items():
            pages = [self._free_pages.pop() for _ in range(length)]
            for index, page in enumerate(pages):
                row = self._slab[page]
                # 非阻塞复制 block_params 和 final_params 到主机页
                row[:self.block_numel].view(self.num_layers, self.block_width).copy_(block_
params[slot, index], non_blocking=True)
                row[self.block_numel:].copy_(final_params[slot, index], non_blocking=True)
            staged[key] = _HostPlan(pages=pages, refcount=1)
    self.synchronize() # 等待所有复制完成
    self._plans.update(staged)
    self._groups[group_key] = tuple(entries)

```

python/sglang/multimodal_gen/runtime/models/dits/minimax_h3.py

修改主模型文件, 移除内联缓存代码, 改为导入新模块, 并添加缓存槽位解析逻辑。

```

# prepare_adaln_plans: 在主机上解析计划槽位, 返回设备张量供 forward 使用
def prepare_adaln_plans(self, step_timesteps: list[torch.Tensor]) -> torch.Tensor | None:
    """Fill the AdaLN cache for this request before denoising starts.
    Returns the per-step slab slots as a device tensor (None without a cache); forward
    consumes one scalar view per step so it never has to match timesteps on device.
    """
    cache = self.adaln_cache
    if cache is None:
        return None
    # 计算计划键, 避免重复 D2H 同步

```

```

keys = [_adaln_plan_key(timesteps) for timesteps in step_timesteps]
if cache.weight_files is not None:
    # 重建路径: 嵌入时间步并构建计划
    def embed(timesteps: torch.Tensor) -> torch.Tensor:
        return timesteps[:, None].expand(-1, self.arch.time_embed_dim)
    cache.build(step_timesteps, embed=embed)
# 解析槽位并返回设备张量, 用于 denoise 循环
slots = cache.resolve_slots(step_timesteps)
# 断言捕获兼容性: 槽位作为设备标量, 避免 CUDA 图捕获时的主机同步
for step, slot in enumerate(slots):
    assert slot.device.type == "cuda" or slot.dim() == 0
return slots

# forward 合约更新: 添加 adaln_cache_slot 关键字参数
def forward(self, *, x, audio_x, adaln_cache_slot=None, ...):
    if adaln_cache_slot is not None:
        # 使用设备槽位索引, 避免主机查找
        adaln_block = self.adaln_cache.block_params[adaln_cache_slot]
    else:
        # 回退到原始路径
        adaln_block = None

```

python/sglang/multimodal_gen/runtime/models/dits/base.py

在基类中添加权重更新验证钩子, 为模型提供安全更新接口。

```

# BaseDiT 中新增的钩子, 用于权重更新安全验证
def validate_weight_update_source(self, *, weights_path: str | None) -> None:
    """Reject a weight update this model cannot stay coherent under.
    Called before any weight is written, so a rejection leaves the served model untouched.
    """
    # 默认空操作; 子类 (如 MiniMaxH3) 覆盖以拒绝不兼容更新
    return None

def validate_lora_layers(self, layer_names: list[str]) -> None:
    """Reject LoRA layers this model cannot apply.
    Called before any LoRA weight is written.
    """
    return None

def refresh_weight_derived_caches(self, *, weights_path: str | None) -> None:
    """Invalidate caches derived from weights after a weight update.
    """
    return None

```

评论区精华

review 聚焦于两个 P1 问题:

1. 权重更新后缓存一致性: BBuf 指出“sidecar deployment keeps serving stale AdaLN conditioning after a weight update”，作者 triple-mu 通过新增 `validate_weight_update_source` 钩子解决，在权重写入前拒绝不兼容更新。
 2. Tensor-RPC 更新处理: BBuf 指出“tensor-RPC updates leave cache.weight_files pointing at the startup checkpoint”，作者通过在 `validate_weight_update_source` 中拒绝 `weights_path=None` 的更新解决。讨论结论是：通过 pre-mutation 验证和缓存失效钩子确保原子性，未解决疑虑：AMD 和 NPU 平台失败与本 PR 无关，已在 main 上复现。
- 权重更新后缓存一致性 (correctness): 通过 pre-mutation 验证确保原子性，添加回归测试验证。
 - Tensor-RPC 更新处理 (correctness): 两种更新路径都添加验证钩子，拒绝不兼容更新。

风险与影响

- 风险:
 1. 数据一致性风险: 如果权重更新后缓存未正确失效，可能导致新权重与旧 AdaLN 条件配对；已通过 `validate_weight_update_source` 和 `refresh_weight_derived_caches` 钩子缓解。
 2. 性能风险: 主机内存分配失败时的降级路径 (recompute fallback) 可能影响延迟；已通过测试覆盖和容量最小化跨 rank 同步保证。
 3. 兼容性风险: 新 CLI 参数和环境变量可能破坏现有部署；默认值保持向后兼容，非在线模式忽略主机缓存参数。
 4. 多 rank 同步风险: 主机层状态转换必须同步以保持 rank 一致性；已通过 prepare 内同步提交实现。
- 影响:
 1. 用户影响: 重复调度请求的延迟降低约 6.7% (94.8s → 88.4s)，输出位相同，提升吞吐。
 2. 系统影响: 引入固定主机缓存层增加每 rank 8 GB 内存开销，但消除 24.2 GiB 检查点 I/O。
 3. 团队影响: 新增缓存架构和安全钩子，为后续模型缓存提供参考模式；测试覆盖全面，但增加维护复杂度。 - 风险标记: 核心路径变更，多 rank 同步，权重更新一致性，内存管理

关联脉络

- PR #37332 [Diffusion][minimax-h3] Add SM120 support for SubBlock sparse attention: 同属 MiniMax-H3 模型优化系列，改进注意力后端支持。
- PR #36735 [multimodal_gen] feat: support key masks on USPAttention's replicated-prefix path: 修复 BCG + SP 下掩码问题，涉及扩散模块注意力层，与 AdaLN 缓存可能交互。
- PR #35922 [diffusion] feat: add maybe_record_function profiler spans for request phases: 为扩散模块添加性能分析 spans，有助于理解缓存优化的性能影响。