PR #36062 完整报告

sgl-project/sglang

[diffusion] cache LoRA-merged weights in files the page cache can hold

合并时间：2026-08-24 14:18

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36062

执行摘要

- 一句话：新增 LoRA 合并权重文件缓存，峰值主机内存降 75%
- 推荐动作：值得精读。重点看三处设计：①用文件后备存储把合并结果放进 page cache 的思路（对任何大权重 merge 场景有普遍借鉴意义）；②通过 install_merged_weight 把状态转换收归 layer 所有，避免查询带副作用；③零拷贝 view 与原地 merge 的语义边界划分（_ensure_base_snapshot_owned）。建议同时关注缓存 key 的哈希范围和并发写入保护是否足够。

功能与动机

PR body 明确说明：在 layerwise offload 下服务蒸馏 LoRA（如 lightx2v 的 4-step MiniMax-H3-Turbo）时，原地 merge 会对 checkpoint 映射做 copy-on-write，61.7 GB 变为匿名内存，LoRA wrapper 又为 unmerge 克隆第二个完整 base 快照（再占 38 GB）。实测峰值 75.3 GiB，host memory available 显示 0.0 GiB，所有 pin plan 归零，真实 32 GB 机器直接死机。修复目标是让该工作负载能在 32 GiB host 上运行，同时保持原有原地路径语义不变。

实现拆解

1. 新增文件型合并缓存层（python/sglang/multimodal_gen/runtime/pipelines_core/lora/lora_merge_cache.py）：lora_merge_cache_key 用 base 检查点路径与有序 (lora_path, strength, alpha) 三元组（含文件大小）算 SHA-1 key；LoraMergeCache.put 把每层合并结果写入 SGLANG_DIFFUSION_CACHE_ROOT/lora_merge_cache// 下的 safetensors 文件，finalize 写 manifest.json；is_complete/get 让后续启动直接 mmap 回读，shape/dtype 不匹配则整体丢弃缓存。_ensure_writable 预检磁盘余量（1.15 倍 headroom），不足时返回不可写。
2. 改造 LoRA 合并启动路径（runtime/pipelines_core/lora/pipeline.py）：__init__ 改为 convert_to_lora_layers(snapshot_base=False) + set_lora(..., cache_merged=True)；新增 _merge_cache_for（仅对 CPU-backed 层启用缓存）与 _merge_via_cache（逐层 copy-merge：同一 device 临时量 + fp32 策略，写回 CPU 后入缓存并 mmap 返回）。缓存未命中、不完整或磁盘不足时回退原地 merge，正确性优先。
3. 零拷贝 unmerge 快照（runtime/layers/lora/linear.py）：BaseLayerWithLoRA.__init__ 新增 snapshot_base 参数，为 False 时 cpu_weight 只是 base_layer.weight.detach() 的 view（_base_is_view=True），避免 38 GB clone；新增 compute_merged_weight（

device 上计算，不写 base）、install_merged_weight（参数指向 mapped 张量并切换 merged 状态）、_ensure_base_snapshot_owned（动态 set_lora 原地合并前先物化clone）；_use_owned_base_snapshot 限定零拷贝快照仅适用 CPU-backed 层。

4. 配套改动：envs.py 注册 SGLANG_DIFFUSION_DISABLE_LORA_MERGE_CACHE 逃生开关；新增 test_lora_merge_cache.py 覆盖 put/finalize/is_complete/ 不匹配拒绝 / 磁盘不足 /key 独立性；test_lora_pipeline.py 新增两条测试锁定“缓存仅接受 CPU-backed 权重”和“零拷贝快照仅限 CPU-backed 层”。

关键文件：

- python/sglang/multimodal_gen/runtime/pipelines_core/lora/lora_merge_cache.py（模块 合并缓存；类别 source；类型 core-logic；符号 lora_merge_cache_key, LoraMergeCache, is_complete, get）：新增文件后备合并权重缓存，是全 PR 收益的核心来源：合并结果逐层写入 safetensors 并 mmap 回读，把字节从匿名内存移到 page cache；同时承担 key 计算、完整性校验、磁盘余量检查。
- python/sglang/multimodal_gen/runtime/pipelines_core/lora/pipeline.py（模块 LoRA 管道；类别 source；类型 core-logic；符号 convert_to_lora_layers, _merge_via_cache, _merge_cache_for）：LoRAPipeline 启动路径从原地 merge 切到缓存合并：snapshot_base=False + cache_merged=True，并新增 _merge_cache_for/_merge_via_cache 做逐层 copy-merge 与回退协调。
- python/sglang/multimodal_gen/runtime/layers/lora/linear.py（模块 LoRA 层；类别 source；类型 core-logic；符号 _ensure_base_snapshot_owned, compute_merged_weight, install_merged_weight, _use_owned_base_snapshot）：BaseLayerWithLoRA 引入 snapshot_base 参数与零拷贝 view 快照语义，新增 compute_merged_weight / install_merged_weight / _ensure_base_snapshot_owned, 划定缓存合并与原地合并的状态边界。
- python/sglang/multimodal_gen/test/unit/test_lora_merge_cache.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_put_round_trips_and_returns_a_mapping, test_an_incomplete_cache_is_not_complete, test_a_mismatched_entry_is_refused, test_disk_shortage_returns_none）：覆盖缓存读写往返、完整性判定、shape/dtype 不匹配拒绝、磁盘不足返回 None、key 区分组合，是缓存正确性的验证主体。
- python/sglang/multimodal_gen/test/unit/test_lora_pipeline.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_merge_cache_only_accepts_cpu_backed_weights, test_zero_copy_snapshot_is_limited_to_cpu_backed_layers）：验证缓存仅接受 CPU-backed 权重、零拷贝快照仅限 CPU-backed 层，防止后续改动破坏内存语义边界。
- python/sglang/multimodal_gen/envs.py（模块 环境变量；类别 source；类型 configuration）：注册 SGLANG_DIFFUSION_DISABLE_LORA_MERGE_CACHE 逃生开关，保证异常情况下可一键恢复旧行为。

关键符号：lora_merge_cache_key, LoraMergeCache.is_complete, LoraMergeCache.get, LoraMergeCache.put, LoraMergeCache.finalize, convert_to_lora_layers, _merge_via_cache, _merge_cache_for, _ensure_base_snapshot_owned, compute_merged_weight, install_merged_weight, _use_owned_base_snapshot

关键源码片段

python/sglang/multimodal_gen/runtime/pipelines_core/lora/lora_merge_cache.py

新增文件后备合并权重缓存，是全 PR 收益的核心来源：合并结果逐层写入 safetensors 并 mmap 回读，把字节从匿名内存移到 page cache；同时承担 key 计算、完整性校验、磁盘余量检查。

```
# lora_merge_cache.py —— 文件后备的 LoRA 合并权重存储
# 核心思路：合并结果逐层写入 safetensors 文件并 mmap 回读，使字节归入
# page cache（可回收）而不是匿名内存；offload manager 无需协调即可识别。

def lora_merge_cache_key(base_paths, adapters):
    """以 base 检查点路径 + 有序 (lora_path, strength, alpha) 三元组计算缓存键。"""
    parts = [os.path.realpath(p) for p in sorted(base_paths)]
    for path, strength, alpha in adapters:
        real = os.path.realpath(path)
        try:
            size = os.path.getsize(real) # 文件大小纳入 key，覆盖同名替换
        except OSError:
            size = -1
        parts.append(f"{real}|{size}|{strength}|{alpha}")
    return hashlib.sha1("||".join(parts).encode()).hexdigest()[:16]


class LoraMergeCache:
    """分层流式写入合并权重，每层一个 safetensors 文件。"""

    def __init__(self, key, expected_bytes):
        self.root = os.path.join(envs.SGLANG_DIFFUSION_CACHE_ROOT, "lora_merge_cache",
        key)
        self.manifest_path = os.path.join(self.root, _MANIFEST)
        self.expected_bytes = expected_bytes
        self._entries = {}
        self._writable = None

    def is_complete(self):
        """上一次同组合运行留下的完整存储：manifest 中所有文件都必须在场。"""
        try:
            with open(self.manifest_path) as handle:
                manifest = json.load(handle)
        except (OSError, ValueError):
            return False
        entries = manifest.get("layers")
        if not isinstance(entries, dict) or not entries:
            return False
        for meta in entries.values():
            if not os.path.exists(os.path.join(self.root, meta.get("file", ""))):
```

```
                return False
        self._entries = entries
        return True

    def get(self, name, shape, dtype):
        """查询同名张量；缺失或 shape/dtype 不匹配返回 None，并整体丢弃缓存。"""
        meta = self._entries.get(name)
        if meta is None:
            return None
        mapped = safetensors_load_file(os.path.join(self.root, meta["file"]))
        tensor = mapped.get("weight")
        if tensor is None or tuple(tensor.shape) != tuple(shape) or tensor.dtype != dtype:
            self._entries = {}
            return None
        return tensor

    def _ensure_writable(self):
        """检查磁盘余量，不满足则禁用缓存（正确性优先，内存第二）。"""
        if self._writable is not None:
            return self._writable
        try:
            os.makedirs(self.root, exist_ok=True)
            usage = shutil.disk_usage(self.root)
            if usage.free < self.expected_bytes * _DISK_HEADROOM:
                self._writable = False
                return False
            self._writable = True
        except OSError:
            self._writable = False
        return self._writable
```

python/sglang/multimodal_gen/runtime/layers/lora/linear.py

BaseLayerWithLoRA 引入 snapshot_base 参数与零拷贝 view 快照语义，新增 compute_merged_weight / install_merged_weight / _ensure_base_snapshot_owned，划定缓存合并与原地合并的状态边界。

```
# linear.py —— BaseLayerWithLoRA 的零拷贝快照与缓存合并状态切换
# 关键点：snapshot_base=False 时 cpu_weight 只是 base weight 的 view，
# 仅在合并不写 base 存储（file-backed 缓存路径）时有效；原地合并前必须先物化。

class BaseLayerWithLoRA(nn.Module):
    def __init__(self, base_layer, lora_rank=None, lora_alpha=None, snapshot_base=True):
        super().__init__()
        self.base_layer = base_layer
        self.merged = False
        if snapshot_base:
            # 传统路径：clone 一份 CPU 快照，任何原地合并都不会污染备份
            self.cpu_weight = base_layer.weight.detach().to("cpu").clone()
            self._base_is_view = False
```

```
        else:
            # 缓存路径：view 不占匿名内存（H3 的 DiT 备份 clone 一次 38 GB）
            self.cpu_weight = base_layer.weight.detach()
            self._base_is_view = True
        self.disable_lora = True

    def _ensure_base_snapshot_owned(self):
        """原地合并即将写 base 存储；若快照是 view，先物化为 clone 再继续。"""
        if self._base_is_view:
            self.cpu_weight = self.cpu_weight.clone()
            self._base_is_view = False

    @torch.no_grad()
    def compute_merged_weight(self):
        """在 device 临时量上计算合并结果，base 存储不被写；与原地路径字节一致。"""
        base = self.weight.data
        target_dtype = base.dtype
        work = base.detach().to(get_local_torch_device())
        if (
            self._should_merge_in_fp32(self.lora_weights_list)
            and work.is_floating_point()
            and work.dtype != torch.float32
        ):
            work = work.to(torch.float32)
        self._merge_lora_into_data(work, self.lora_weights_list)
        return work.to("cpu", dtype=target_dtype)

    def install_merged_weight(self, merged, base_view):
        """唯一发生缓存合并状态切换的地方：参数指向 mapped 张量，快照是零拷贝 view。"""
        self.weight.data = merged
        self.merged = True
        self.cpu_weight = base_view.detach()
        self._base_is_view = True
```

评论区精华

本 PR 无公开 review 评论，但第 4 次提交记录了 Review feedback。核心交锋：

- 缓存接口的副作用与状态归属：LoraMergeCache.get/put 原本会就地修改调用方参数（查询带副作用），且 pipeline 的 merge-via-cache 代做了 layer 的职责。结论：缓存只 vend tensors，pipeline 只做协调，merged/cpu_weight 状态转换收敛到 BaseLayerWithLoRA.install_merged_weight。
- 驻留层语义边界：两次 fix 提交（keep resident LoRA weights on device / snapshot resident LoRA base weights）表明，零拷贝快照只对 CPU-backed 层成立；驻留层保留自有 clone 快照，避免 unmerge 语义被破坏。
- 回退策略：body 声明 "correctness first, memory second"——不匹配 / 不完整缓存丢弃，磁盘不足回退原地 merge，动态 set_lora 与多 GPU 保留原地路径。

- 缓存接口的副作用与状态归属 (design): 重构后缓存只 vend tensors，pipeline 只做协调，状态转换收敛到 BaseLayerWithLoRA.install_merged_weight；put 返回 mapping 而非就地修改参数。
- 驻留层与 CPU-backed 层的语义边界 (correctness): 通过 _use_owned_base_snapshot 限制：只有 CPU-backed 层允许零拷贝快照与缓存合并；驻留层保留自有 clone 快照。
- 缓存失效与磁盘不足的回退 (design): _ensure_writable 检查 1.15 倍磁盘余量，put 失败返回 None，pipeline 回退原有点位合并；多 GPU 与动态 set_lora 始终保留原地路径。

风险与影响

- 风险：
  - 缓存一致性：key 不含模型内容哈希，依赖路径、文件大小、strength、alpha 与文件大小；同名替换但大小不变时可复用旧缓存，shape/dtype 校验拦不住字节级差异。
  - 磁盘占用：单组合缓存约 38 GB，_ensure_writable 预留 1.15 倍余量，但无清理策略，多组合并存可能长期占用磁盘。
  - 并发写入：os.makedirs + safetensors 写文件无锁，多 worker/fork 同时启动相同 key 可能互相覆盖 manifest；测试未覆盖并发。
  - 回退路径本身仍受内存限制：磁盘不足时回退原地 merge 仍可能 OOM，只是正确性优先。
  - 影响：对用户：显著正向——32 GB host 可跑 MiniMax-H3-Turbo 这类大 diffusion LoRA，第二次启动更快；pin 计划恢复，e2e 时间基本持平（76-87 s）。对系统：主机匿名内存峰值从 75.3 GiB 降至 19.2 GiB，代价是磁盘占用和新增缓存子系统。对团队：后续模型 /LoRA/loader 改动需维护 cache key 与有效性语义，新增了必须谨慎对待的缓存失效面。
  - 风险标记：缓存一致性风险，磁盘占用增加，并发写入无保护，LoRA 合并路径变更

关联脉络

- PR #36070 [Diffusion] Load pruned MiniMax H3 components natively: 与 #36062 改动了相同的 lora/linear.py 与 lora/pipeline.py，同属 diffusion LoRA 功能线；#36062 在其上叠加文件型合并缓存，共同支撑 MiniMax-H3 LoRA 在受限主机上运行。