PR #35858 完整报告

sgl-project/sglang

[diffusion] Allow Cache-DiT with DiT layerwise offload

合并时间：2026-08-30 20:55

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35858

执行摘要

- 一句话：Cache-DiT 与层间卸载解除互斥，低显存加速达 6.8×
- 推荐动作：值得精读。核心设计决策“只信任真正运行的层”把两个正交特性的耦合点收敛到一个状态字段和两个释放函数上，改动极小却解决了复杂的顺序 / 跳层竞态；参数化测试矩阵（Fn/Bn × prefetch_size × residency policy × hit/miss/hit-again 序列）是很好的回归防护。建议关注三个细节：post-hook 记录 _last_forwarded_layer 的时机、_release_skip_gap 与 _retained_set 的差集保护、以及 prepare_for_next_req 对 Bn=0 遗留预取的兜底清理。

功能与动机

PR body 明确指出旧有的启动 ValueError 是“implementation accident, not a real mutex”，两条优化轴本质正交：Layerwise 让大部分权重留在 CPU、当前层复制到 GPU 并后台预取下一层；Cache-DiT 在一个 step 中可能跳过中间块（DBCache 先跑前 Fn 块，命中后跳过 Mn 块，可选跑末尾 Bn 块）。跳过块意味着更少的 H2D 传输，但层间卸载假设每步按 0..N-1 顺序遍历并预取 i+1，导致 layer 1 在 layer 0 期间被预取却永不发布 release hook，或 wrap/release 后给下一个计算层留下 empty((1,)) → shape mismatch。解除禁止后，24 GB 4090 上 MiniMax-H3 端到端从 721.2 s 降至 173.2 s（4.16×），叠加 sage_attn 后 106.4 s（6.78×），推荐画质档 Fn=1/Bn=2/W=8/R=0.08/MC=2 达 30.10 dB PSNR、1.84×。

实现拆解

1. 移除启动互斥（入口）：server_args.py 的 _validate_offload 删除“DiT layerwise offload cannot be enabled together with cache-dit”的 raise ValueError 分支，并把 --dit-layerwise-offload 的 help 从“Cannot be used together with cache-dit”改为“Compatible with cache-dit: skipped blocks are not streamed”；FSDP 与 Cache-DiT 的互斥校验原样保留。
2. 跟踪最后真正运行的层：layerwise_offload.py 的 __init__新增 self._last_forwarded_layer: int | None = None（注释点明 skip-compute 可以跳层）；post-forward hook 在 release_layer(i) 之前先记录 _last_forwarded_layer = i，只有真正执行过 forward 的层才参与跳层判断。
3. 跳层缝隙释放：pre-forward hook 增加 elif 分支，当 i > _last_forwarded_layer + 1 时调用新增的 _release_skip_gap(last_ran, next_ran)，把开区间内被投机预取但永不运行的层立即 release_layer；随后若 i 不在 _gpu_layers 中，则走原有 prefetch_layer(i,

non_blocking=False) 同步加载（跳层后首层的一次 PCIe 传输）。新方法 _release_unneeded_streamed_layers 与 _release_skip_gap 都会先与 _retained_set 求差集，避免误释放 resident 层。

4. step 边界清理：prepare_for_next_req 开头重置 _last_forwarded_layer = None 并调用 _release_unneeded_streamed_layers(keep=set(self._head_of_stream()))，专门回收 Bn=0 默认档下"layer 1 被预取但从未运行"的遗留；全栈遍历时的顺序预取与尾部 % N wrap 行为保持不变。
5. 测试与文档配套：test_layerwise_offload.py 新增 179 行测试，_dbcache_layers 按 Fn/Bn 模拟 DBCache 调用序列，_layer_weight_ok 用 weight.shape != (1,) 表达真实权重不变式；test_dbcache_layer_patterns_never_see_empty_weights 以 num_layers × fn/bn × prefetch_size × residency_policy 四维参数化覆盖 hit/miss/hit-again，test_mixed_scm_and_dbcache_step_schedule 覆盖 SCM/TaylorSeer 风格的全栈与跳层混合调度；test_server_args.py 新增 test_cache_dit_allows_explicit_dit_layerwise_offload。文档同步更新 MiniMax-H3 cookbook、cli.mdx、cache_dit.mdx、caching-acceleration.mdx 与 performance SKILL.md，统一移除"不兼容层间卸载"的表述并补充跳层后同步加载的说明。

关键文件：

- python/sglang/multimodal_gen/runtime/managers/memory_managers/layerwise_offload.py（模块 层间卸载；类别 source；类型 core-logic；符号 _last_forwarded_layer, _release_unneeded_streamed_layers, _release_skip_gap, prepare_for_next_req）：核心实现文件：新增 _last_forwarded_layer 状态、_release_skip_gap 与 _release_unneeded_streamed_layers 两个释放路径，并改造 pre/post forward hook 使其具备 Cache-DiT 跳层感知能力；prepare_for_next_req 增加 step 边界清理。
- python/sglang/multimodal_gen/test/unit/test_layerwise_offload.py（模块 层间卸载；类别 test；类型 test-coverage；符号 _layer_weight_ok, _dbcache_layers, _run_layer_set, _assert_gpu_layers_have_real_weights）：179 行参数化测试是本次变更的主要质量保障：模拟 DBCache hit/miss/hit-again 与 SCM 混合调度，用『GPU 上必有真实权重』不变式防止 empty((1,)) 回归。
- python/sglang/multimodal_gen/runtime/server_args/server_args.py（模块 服务参数；类别 source；类型 core-logic）：删除"Cache-DiT 与 layerwise offload 互斥"的启动 ValueError，并同步更新 --dit-layerwise-offload 的 CLI help；FSDP 互斥校验保留，是本次功能解锁的入口改动。
- python/sglang/multimodal_gen/test/unit/test_server_args.py（模块 服务参数；类别 test；类型 test-coverage；符号 test_cache_dit_allows_explicit_dit_layerwise_offload）：新增 test_cache_dit_allows_explicit_dit_layerwise_offload，验证显示开启 dit_layerwise_offload 时不再抛错且组件选择正确，与删除互斥校验的源码改动一一对应。
- docs/cookbook/diffusion/MiniMax/MiniMax-H3.mdx（模块 文档；类别 docs；类型 documentation）：H3 是本次验证的旗舰场景：更新 cookbook 的 Cache-DiT 手动调参说明、24 GB 层间卸载 recipe 与 Warning 文案，新增『skipped blocks are not streamed』的兼容性声明。

- docs/docs/sglang-diffusion/cache_dit.mdx（模块 文档；类别 docs；类型 documentation）：Limitations 章节新增『DiT layerwise offload: Compatible』条目，并补充 SCM 在 steps - 1 NFE 场景下的上游 steps_mask 限制提示。

关键符号：_release_unneeded_streamed_layers, _release_skip_gap, prepare_for_next_req, register_forward_hooks, test_dbcache_layer_patterns_never_see_empty_weights, test_skip_middle_layers_loads_destination_weights, test_mixed_scm_and_dbcache_step_schedule, test_cache_dit_allows_explicit_dit_layerwise_offload

关键源码片段

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

核心实现文件：新增 _last_forwarded_layer 状态、_release_skip_gap 与 _release_unneeded_streamed_layers 两个释放路径，并改造 pre/post forward hook 使其具备 Cache-DiT 跳层感知能力；prepare_for_next_req 增加 step 边界清理。

```
# 核心原则：只信任「真正执行过 forward」的层。
# 层间卸载的旧假设是每个 denoising step 都按 0..N-1 顺序遍历，
# 于是层 i 之后预取 i+1、尾部用 % N 环绕到下一轮 layer 0。
# Cache-DiT 命中时会跳过中间块（例如 0 -> 6），被跳过层既不 forward
# 也不释放，残留的 empty((1,)) 占位权重会在下一个计算层上引发
# shape mismatch。


def _release_unneeded_streamed_layers(self, *, keep: Set[int]) -> None:
    """释放 GPU 上不在 keep 集合、也不属于 resident 集合的流式层。

    prepare_for_next_req 在 step 开头调用，清掉上一轮 Cache-DiT
    命中后「预取了但从未 forward」的层（默认 Bn=0 时 layer 1 会这样挂着）。
    """
    retain = set(self._retained_set) | keep
    for layer_idx in list(self._gpu_layers):
        if layer_idx not in retain:
            self.release_layer(layer_idx)


def _release_skip_gap(self, *, last_ran: int, next_ran: int) -> None:
    """跳层后释放 (last_ran, next_ran) 开区间内的投机预取层。

    例如 0 -> 6 的跳变：1..5 被预取过但永远不会运行，
    立即回收比等到下一个 step 再清理更省显存。
    """
    if next_ran <= last_ran + 1:
        return
    retain = set(self._retained_set)
```

```
    for layer_idx in range(last_ran + 1, next_ran):
        if layer_idx not in retain:
            self.release_layer(layer_idx)


# --- forward hooks 的跳层感知改造 ---

def make_pre_hook(i):
    def hook(module, input):
        if i == 0:
            # 每个 denoising step 的 layer 0：激活驻留集、重置跳层记录，
            # 并回收上一轮遗留的投机预取。
            self._activate_residency()
            self.prepare_for_next_req(non_blocking=False)
        elif (
            self._last_forwarded_layer is not None
            and i > self._last_forwarded_layer + 1
        ):
            # 检测到 DBCache 跳层（如 0 -> 6）：立即释放中间缝隙，
            # 避免被跳过层继续占住显存。
            self._release_skip_gap(
                last_ran=self._last_forwarded_layer, next_ran=i
            )
        if i not in self._gpu_layers:
            # 跳层目的地未必已被预取到位：同步加载（一次 PCIe 传输）。
            self.prefetch_layer(i, non_blocking=False)
        if i in self._prefetch_events and self.copy_stream is not None:
            # 等待该层对应的异步拷贝事件，确保权重真正驻留。
            torch.get_device_module().current_stream().wait_event(
                self._prefetch_events[i]
            )
        # 后续按 residency policy 补充预取（leading 突发 / strided 逐层
        # 补流）的逻辑保持不变，prefetch_layer 幂等，重复请求无额外开销。
    return hook


def make_post_hook(i):
    def hook(module, input, output):
        # 只记录真正跑过的层：它是下一轮 pre-hook 判断跳层缝隙的依据。
        self._last_forwarded_layer = i
        self.release_layer(i)
    return hook


def prepare_for_next_req(self, non_blocking=True):
    """新一轮 denoising 循环前的准备：清理遗留、预取驻留集与流头。"""
    self._last_forwarded_layer = None
    # Bn=0 命中后 layer 1 会被预取却从不 forward，这里统一回收。
    self._release_unneeded_streamed_layers(keep=set(self._head_of_stream()))
```

```
    # 随后预取 resident 集合与流头（顺序与异步策略保持原样）。
    for layer_idx in sorted(self._retained_set):
        self.prefetch_layer(layer_idx, non_blocking=non_blocking)
    if not non_blocking and self.copy_stream is not None:
        torch.get_device_module().current_stream().wait_stream(self.copy_stream)
    for layer_idx in self._head_of_stream():
        self.prefetch_layer(layer_idx, non_blocking=True)
```

python/sglang/multimodal_gen/test/unit/test_layerwise_offload.py

179 行参数化测试是本次变更的主要质量保障：模拟 DBCache hit/miss/hit-again 与 SCM 混合调度，用『GPU 上必有真实权重』不变式防止 empty((1,)) 回归。

```
def _layer_weight_ok(layer: torch.nn.Module) -> bool:
    """GPU 上的层必须持有真实权重，而不是 empty((1,)) 占位符。"""
    return tuple(layer.weight.shape) != (1,)


def _dbcache_layers(num_layers: int, fn: int, bn: int) -> list[int]:
    """模拟 CachedBlocks 在一次 DBCache step 中会调用的层序列。

    前 Fn 块固定计算；命中时跳过中间 Mn 块；可选再跑末尾 Bn 块。
    用这个序列喂给模型，就能复现 Cache-DiT 的跳层执行模式。
    """
    fn = min(max(fn, 0), num_layers)
    bn = min(max(bn, 0), num_layers - fn)
    layers = list(range(fn))
    if bn:
        layers.extend(range(num_layers - bn, num_layers))
    return layers


@pytest.mark.parametrize("num_layers", [8, 12])
@pytest.mark.parametrize("fn,bn", [(1, 0), (1, 2), (2, 0), (4, 2), (3, 5), (8, 0)])
@pytest.mark.parametrize("prefetch_size", [1, 2])
@pytest.mark.parametrize(
    "residency_policy", [RESIDENCY_POLICY_LEADING, RESIDENCY_POLICY_STRIDED]
)
def test_dbcache_layer_patterns_never_see_empty_weights(
    monkeypatch, num_layers, fn, bn, prefetch_size, residency_policy
):
    """hit / miss / hit-again 序列下，计算层永不读到空占位权重。"""
    ...
    def _assert_gpu_layers_have_real_weights() -> None:
        for idx in range(num_layers):
            # 核心不变式：层在 GPU 上 ⇔ 持有真实权重；
            # 不在 GPU 上的层必须保持 empty((1,)) 占位符。
            on_gpu = idx in manager._gpu_layers
            assert _layer_weight_ok(model.blocks[idx]) is on_gpu, idx
```

```
    # 依次执行 hit（Fn + Bn）、miss（全栈）、连续两次 hit，
    # 每段之间调用 prepare_for_next_req 模拟 step 边界。
    hidden = _run_layer_set(model, hit_layers)
    assert hidden.shape == (1, 2)
    _assert_gpu_layers_have_real_weights()
    manager.prepare_for_next_req(non_blocking=False)
    hidden = _run_layer_set(model, miss_layers)
    assert hidden.shape == (1, 2)
    _assert_gpu_layers_have_real_weights()
    manager.prepare_for_next_req(non_blocking=False)
    hidden = _run_layer_set(model, hit_layers)
    assert hidden.shape == (1, 2)
    _assert_gpu_layers_have_real_weights()
```

python/sglang/multimodal_gen/runtime/server_args/server_args.py

删除“Cache-DiT 与 layerwise offload 互斥”的启动 ValueError，并同步更新 --dit-layerwise-offload 的 CLI help；FSDP 互斥校验保留，是本次功能解锁的入口改动。

```
  # 互斥校验重构：删除 Cache-DiT 与 DiT 层间卸载的互斥分支。
  # 之前这里抛出的 ValueError 是「实现事故」而非真实约束：
  # 层间卸载现已具备跳层感知能力（跳过块不 stream、跳层后首层同步加载），
  # 因此二者可以共存。FSDP 与 Cache-DiT 的互斥仍然保留。
  if envs.SGLANG_CACHE_DIT_ENABLED and self.use_fsdp_inference:
      if self.is_arg_explicitly_set("use_fsdp_inference"):
          raise ValueError(
              "FSDP inference cannot be enabled together with cache-dit. "
              "cache-dit wraps known DiT block structures, while FSDP wraps "
              "and shards modules before cache-dit can inspect them. "
              "Please disable --use-fsdp-inference or disable "
              "SGLANG_CACHE_DIT_ENABLED."
          )
      logger.warning(
          "cache-dit is enabled, automatically disabling use_fsdp_inference."
      )
      self.use_fsdp_inference = False

  # --dit-layerwise-offload 的 help 同步更新为：
  # "Compatible with cache-dit: skipped blocks are not streamed. "
  # "Cannot be used together with use_fsdp_inference. ..."
```

评论区精华

该 PR 的 review 过程没有留下任何代码评论：维护者 mickqian 直接 APPROVED（空正文），唯一的交互是在 Issue 评论发布 /tag-and-rerun-ci 触发 CI 重跑。技术权衡讨论主要沉淀在 PR body 中：

> “The old startup ValueError was an implementation accident, not a real mutex.”

> “Trust only layers that actually ran. A full-stack step still prefetches as before (including last-layer wrap). Jump: release the unused gap; sync-load the destination if needed. Next step: prepare drops leftover prefetch (Bn=0).”

作者主动披露三条边界：跳层后的首层可能有一次 PCIe 同步加载；上游 Cache-DiT 的 steps_mask 在 total_steps < 8 时只允许 4/6 步（SCM 建议 ≥9 步）；Spectrum/TeaCache 不建议与 Cache-DiT 叠加，因为 hooks 只覆盖整块 forward()。

- CI 重跑与合入确认 (other): CI 重跑后合入；无未解决的 review 疑虑。

风险与影响

- 风险：
  1. 核心路径回归风险：register_forward_hooks 是所有层间卸载用户的必经路径。新增分支只在检测到跳层或 step 边界时触发释放，全栈顺序遍历时行为不变，并有 test_last_layer_wraps_to_next_step_head 守护尾部 wrap；但 _release_unneeded_streamed_layers 的释放范围依赖 _head_of_stream() 与 _retained_set 在 leading/strided 两种 residency policy 下的语义，未来 policy 调整可能引入提前释放导致更多同步加载。
  2. 性能抖动：跳层后目标层不在 GPU 时 prefetch_layer(i, non_blocking=False) 同步阻塞一次 H2D 传输；默认 Bn=0 无 Bn 层可加载影响最小，但自定义 Fn/Bn 档需按模型评估。
  3. 品质不确定性：Cache-DiT 是近似计算，叠加层间卸载不改变近似性，但实测 PSNR 因参数跨度很大（15.89~30.10 dB），用户需按目标任务核对画质；B200 的 quality: high audited 路径保持 fail-closed。
  4. 组合限制：FSDP 仍与 Cache-DiT 互斥；SCM 在 MiniMax-H3 上因 8 步报告 7 NFE 会触发上游 steps_mask 断言，需 ≥9 步；Spectrum/TeaCache 叠加不可用。
  5. 测试覆盖局限：单测基于 fake device 与 _RunnableBlockModel，未覆盖真实 CUDA graph、多 GPU 或 FSDP 组合；PR 的 Extra CI 任务失败原因未在 PR 中说明（未阻断合入）。 - 影响：用户侧：24 GB 级单卡扩散推理用户是最大受益者，此前必须二选一（层间卸载保显存或 Cache-DiT 提速），现在可叠加，MiniMax-H3 端到端最高约 6.78× 加速；存量配置无 breaking change（之前互斥报错变成合法组合）。系统侧：layerwise_offload.py 是核心 offload 管理器，新增 _last_forwarded_layer 状态与两条释放路径影响所有启用层间卸载的 diffusion 管线，但其行为对非跳层场景透明。团队侧：4 处文档统一了 CLI 帮助、cookbook 与缓存加速文档的表述，消除了此前文档与实现不一致的“不兼容”声明，为后续把 Cache-DiT 与更多路径组合提供了先例。 - 风险标记：核心路径变更（forward hooks），近似算法叠加品质差异大，跳层后同步加载延迟，单测基于 fake device, CI Extra 任务失败未说明

关联脉络

- PR #35684 Spectrum 缓存方案（PR body 引用）：PR body 将其作为 Cache-DiT 对比方案：配置 11/5/1.0 在相同场景下 285.3 s / 27.4 dB，比本 PR 推荐档更快但 PSNR 略低，用于说明推荐档的取舍。

- PR #36991 [Diffusion] Add exact component precision overrides: 同属 diffusion 运行时与 server_args 区域的组件级内存 / 加载配置演进，与该 PR 的层间卸载组件选择逻辑有共同上下文。
- PR #37116 [diffusion] perf: absorb Qwen-Image output projection biases: 同属 diffusion 性能优化序列，共用 e2e/denoise/PSNR 的 benchmark 方法论，反映该模块持续的性能压榨方向。