PR #35986 完整报告

sgl-project/sglang

[diffusion] re-home decode-dtype VAE weights to a file-backed store

合并时间：2026-08-23 09:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35986

执行摘要

- 一句话：VAE 解码权重改为文件后备存储，降低主机内存占用
- 推荐动作：值得精读。该 PR 展示了一个低成本高收益的内存账本优化模式：将不可回收的匿名副本转为可丢弃的页缓存，并用 " 校验 + 回退 + kill switch" 三重保障保证正确性与可控性。_rehome_cast_weights_to_file 的原子写、失败删除、二次采用逻辑可以直接作为其他组件缓存落盘的参考模板。建议关注后续是否有针对缓存失效（revision 变化）的补充校验。

功能与动机

PR body 明确指出："The decode-dtype cast copies were anonymous host memory, and on a budgeted host every one of those bytes comes out of the pin budget the stepped components live on"，在 64 GiB 主机上约 4.5 GiB 的匿名副本让 DiT 少 pin 了三层（39 到 36）。因此需要把这些字节从匿名内存账本中移除，使其成为可丢弃的页缓存，并让第二次启动无需重复 cast。safetensors 能字节级精确往返张量，因此 #35967 的 bit-identity 论证可以原样延续。

实现拆解

实现分为 5 步：

1. 新增缓存路径计算函数 _decode_dtype_store_path（vae_loader.py）：用 os.path.realpath(component_model_path)、component_name 和 dtype 拼接字符串后取 SHA1 前 16 位作为文件名，落在 SGLANG_DIFFUSION_CACHE_ROOT/decode_dtype_store/ 下，扩展名为 .safetensors。realpath 钉住了 checkpoint 的 revision 快照，避免软链或路径别名造成 key 漂移。
2. 新增存储校验与采用函数 _assign_matching_store：对映射中的每个 tensor 检查对应参数是否存在、shape 是否一致、dtype 是否等于目标 dtype；全部通过才 load_state_dict(..., assign=True) 并返回 True，否则拒绝采用，交给上层丢弃。
3. 新增核心函数 _rehome_cast_weights_to_file：若缓存文件已存在，先尝试加载并采用，成功则直接返回（无需 cast）；否则调用 prepare(dtype) 执行 cast，收集 CPU 上 dtype 匹配的张量，用 safetensors_save_file 写入临时文件（{path}.tmp.{pid}）再 os.replace 原子替换，随后重新加载映射回模块；任何异常都会删除损坏文件并回退到 prepare(dtype) 的内存副本路径。

4. 修改 _hold_decoder_weights_in_decode_dtype 签名与调用点：新增 component_model_path 参数；调用路径（自定义 VAE 加载和 ModelRegistry 加载两处）均传入该路径；当路径非空且未设置 SGLANG_DIFFUSION_DISABLE_VAE_DECODER_STORE 时走文件存储分支，否则保持原内存 cast。日志补充了 file-backed 或 anonymous host memory 标记。
5. 配置与测试配套：envs.py 注册 SGLANG_DIFFUSION_DISABLE_VAE_DECODER_STORE 布尔环境变量；新增 test_vae_decoder_store.py，用 _TinyVAE 模拟小模型覆盖四个场景：cast 结果文件后备、二次启动采用存储且 prepare_calls == 0、不匹配存储被丢弃并保留 cast、kill switch 不产生缓存文件。测试通过 monkeypatch 将 SGLANG_DIFFUSION_CACHE_ROOT 指向 tmp_path，隔离真实缓存目录。

关键文件：

- python/sglang/multimodal_gen/runtime/loader/component_loaders/vae_loader.py（模块 VAE 加载器；类别 source；类型 core-logic；符号 _decode_dtype_store_path, _assign_matching_store, _rehome_cast_weights_to_file, _hold_decoder_weights_in_decode_dtype）：核心实现文件：新增缓存路径计算、存储校验 / 采用、cast 权重落盘与回退逻辑，并修改两处调用点把 component_model_path 传入。
- python/sglang/multimodal_gen/test/unit/test_vae_decoder_store.py（模块 单元测试；类别 test；类型 test-coverage；符号 _TinyVAE, prepare_decoder_autocast_weights, test_the_cast_weights_end_up_file_backed, test_a_second_start_adopts_the_store_without_casting）：新增单元测试，覆盖文件后备、二次采用、不匹配存储丢弃、kill switch 四条关键路径，验证核心行为。
- python/sglang/multimodal_gen/envs.py（模块 环境配置；类别 source；类型 configuration）：注册新的 kill switch 环境变量 SGLANG_DIFFUSION_DISABLE_VAE_DECODER_STORE，控制存储行为是否启用。

关键符号：_decode_dtype_store_path, _assign_matching_store, _rehome_cast_weights_to_file, _hold_decoder_weights_in_decode_dtype

关键源码片段

python/sglang/multimodal_gen/runtime/loader/component_loaders/vae_loader.py

核心实现文件：新增缓存路径计算、存储校验 / 采用、cast 权重落盘与回退逻辑，并修改两处调用点把 component_model_path 传入。

```
# 核心：把 VAE 解码器提前 cast 到 decode dtype 的权重从匿名内存搬到文件后备存储

def _decode_dtype_store_path(
    component_model_path: str, component_name: str, dtype: torch.dtype
) -> str:
    # key 钉住 realpath（即 checkpoint 目录的真实路径）与组件名、dtype，
    # 换 revision 或换 dtype 都会生成不同的缓存文件，避免互相污染。
    key = hashlib.sha1(
        f"{os.path.realpath(component_model_path)}|{component_name}|{dtype}".encode()
```

```
    ).hexdigest()[:16]
    return os.path.join(
        envs.SGLANG_DIFFUSION_CACHE_ROOT, "decode_dtype_store", f"{key}.safetensors"
    )


def _assign_matching_store(vae, mapped: dict, dtype: torch.dtype) -> bool:
    """只有当存储里的每个张量都能对上模块参数时才采用，否则拒绝。"""
    state = vae.state_dict()
    for name, tensor in mapped.items():
        param = state.get(name)
        # 参数缺失、shape 不一致或 dtype 不是目标 dtype，都说明存储已过期 / 损坏
        if param is None or param.shape != tensor.shape or tensor.dtype != dtype:
            return False
    # assign=True 直接替换参数，保持 file-backed 映射，避免再复制一份匿名内存
    vae.load_state_dict(mapped, strict=False, assign=True)
    return True


def _rehome_cast_weights_to_file(
    vae, dtype: torch.dtype, component_model_path: str, component_name: str, prepare
) -> tuple[int, bool]:
    """返回 (持有的权重数, 是否 file-backed)。

    核心收益：cast 副本原本是匿名内存，内核无法回收；写成文件再 mmap 回来
    后变成页缓存，内存压力下可被丢弃、按需重新读回，同时不占 pin 预算。
    safetensors 字节级精确往返，所以文件里的值与 #35967 的内存 cast 完全一致。
    """
    path = _decode_dtype_store_path(component_model_path, component_name, dtype)
    try:
        # 已有存储：先校验并采用，成功就直接返回，连 cast 都不用做
        if os.path.exists(path):
            mapped = safetensors_load_file(path)
            if mapped and _assign_matching_store(vae, mapped, dtype):
                return len(mapped), True
            raise ValueError("existing decode-dtype store does not match the module")

        # 首次运行：执行 cast，把 CPU 上落到目标 dtype 的张量收集起来写盘
        converted = prepare(dtype)
        if not converted:
            return 0, False
        cast_state = {
            name: tensor
            for name, tensor in vae.state_dict().items()
            if tensor.dtype == dtype and tensor.device.type == "cpu"
        }
        os.makedirs(os.path.dirname(path), exist_ok=True)
        # 原子写：先写 tmp 再 rename，避免把半截文件暴露给别人
        tmp = f"{path}.tmp.{os.getpid()}"
```

```
            safetensors_save_file({k: v.contiguous() for k, v in cast_state.items()}, tmp)
            os.replace(tmp, path)

            # 回读校验 key 集合是否一致，再以 assign 方式挂回模块
            mapped = safetensors_load_file(path)
            if set(mapped) != set(cast_state):
                raise ValueError("decode-dtype store does not match the cast weights")
            vae.load_state_dict(mapped, strict=False, assign=True)
            return converted, True
        except Exception as exc:
            # 任何失败都不阻断加载：删掉坏文件，退回纯内存 cast
            logger.warning(
                "VAE: could not re-home %s decode-dtype weights to %s (%s); "
                "keeping in-memory copies",
                component_name,
                path,
                exc,
            )
            try:
                if os.path.exists(path):
                    os.remove(path)
            except OSError:
                pass
            return prepare(dtype), False
```

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

新增单元测试，覆盖文件后备、二次采用、不匹配存储丢弃、kill switch 四条关键路径，验证核心行为。

```
# 用小模块验证文件后备存储的关键行为，避免真实 VAE 带来的重资源开销

class _TinyVAE(nn.Module):
    def __init__(self):
        super().__init__()
        self.blocks = nn.ModuleList([nn.Linear(8, 8) for _ in range(3)])
        self.head = nn.Linear(8, 8) # 保持 fp32，模拟输出投影不参与 cast
        self.prepare_calls = 0

    def prepare_decoder_autocast_weights(self, dtype) -> int:
        # 记录调用次数，供 " 二次启动不 cast" 的断言使用
        self.prepare_calls += 1
        converted = 0
        for block in self.blocks:
            if block.weight.dtype != dtype:
                block.to(dtype=dtype)
                converted += 1
        return converted

def test_a_second_start_adopts_the_store_without_casting(tmp_path):
```

```
model_path = tmp_path / "ckpt"
model_path.mkdir()
first = _TinyVAE()
_hold_decoder_weights_in_decode_dtype(first, _server_args(), "video_vae", str(model_path))

# 第二个实例先恢复到 fp32，模拟全新进程启动时的初始状态
second = _TinyVAE()
second.load_state_dict(
    {
        k: v.to(torch.float32) if v.dtype == torch.float16 else v
        for k, v in first.state_dict().items()
    }
)
_hold_decoder_weights_in_decode_dtype(second, _server_args(), "video_vae", str(model_path))

# 关键断言：存储被直接采用，prepare 一次都没被调用
assert second.prepare_calls == 0
for name in first.state_dict():
    assert torch.equal(second.state_dict()[name], first.state_dict()[name])
```

评论区精华

该 PR 没有 review 评论线程（review_comments_count = 0），核心设计权衡都记录在 PR body 中：文件后备映射让同一份字节从匿名内存变为可丢弃页缓存，代价是磁盘占用约 4.8 GB；非匹配或不可读存储会被丢弃并回退到内存 cast，保证正确性优先；kill switch 提供了对文件 I/O 敏感环境的逃生通道。由于是单 commit、作者自合，讨论空间较小，但实现中的校验与回退逻辑体现了防御式设计。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 缓存正确性风险：缓存 key 仅基于 realpath、组件名和 dtype，若同一路径下的模型文件被替换为结构相同但权重不同的 checkpoint，可能误用旧缓存。缓解：_assign_matching_store 会逐张量校验 shape 与 dtype，结构变化时拒绝采用并删除文件；但若结构完全相同、仅权重数值不同（例如同一路径下不同 revision 的文件被原地覆盖），则可能加载旧权重。需要依赖 realpath 钉住目录，且一般模型路径不会原地内容替换。
  2. 并发写竞争：多个进程同时启动时，os.replace 是原子的但两个进程可能各自写 tmp 再 replace，最后留下一个完整文件，内容一致所以无正确性问题，但会有瞬时双写。
  3. 页缓存语义变化：文件映射的权重首次访问可能触发缺页 I/O，在 decode 阶段若页缓存被回收，可能出现比匿名内存更明显的延迟抖动；不过 denoise/decode 实测时间与之前基本持平（162.4s vs 167.5s，属于 run-to-run 波动）。
  4. 磁盘占用：每个组件大约多占 4.8 GB 缓存，若同时在跑多个模型，~/.cache/sgl_diffusion 会累积大量文件，需要用户或运维定期清理。

5. kill switch 覆盖不足：_hold_decoder_weights_in_decode_dtype 只在 component_name in ("vae", "video_vae") 时触发，audio_vae 不涉及，风险有限。

- 影响：
  - 用户侧：MiniMax-H3 等 diffusion 模型在受限主机（如 12 GiB 消费卡 + 64 GiB 主机）上，DiT 可 pin 层数提升（36→38），decode 阶段更少 restream；二次启动显著加快（省去约 4.5 GiB 的 fp16 cast）。
  - 系统侧：主机匿名内存压力降低约 4.5 GiB，pin 预算释放给层式 offload 组件；代价是增加同量级磁盘缓存文件，且行为依赖页缓存回收。
  - 团队侧：新增一个环境变量和 133 行单元测试，后续修改 VAE 加载路径需要同时维护存储的采用 / 回退逻辑；envs.py 的命名规范得以延续。
  - 兼容性：默认开启新行为，但提供了 kill switch；旧版缓存不存在时会自动走原始内存路径，无迁移负担。
  - 风险标记：缓存文件可能过期，多进程并发写缓存，页缓存回收导致 I/O 抖动，磁盘占用新增约 4.8 GB，依赖形状校验防误用

关联脉络

- PR #36051 [diffusion] CI: guard the anonymous-host budget alongside peak VRAM: 同属 diffusion 内存预算主题，直接在 CI 中守护匿名主机内存预算；本 PR 正是为了把这些匿名字节移出账本。
- PR #35734 [diffusion] park a layerwise component's non-layer weights between uses: 同为 diffusion 层式 offload 内存优化，通过暂存非层权重缓解显存 / 主机内存压力，与本 PR 的 pin 预算优化互补。
- PR #36034 [diffusion] UX: clean up startup and offload logs: 涉及 layerwise_offload 与 host_memory_budget 模块，说明 diffusion 内存管理正在持续演进，本 PR 是其一部分。