PR #30392 完整报告

sgl-project/sglang

[FEAT] Decouple multimodal global cache from Mooncake

合并时间：2026-08-10 19:23

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/30392

执行摘要

- 一句话：多模态全局缓存解耦 Mooncake，引入可插拔后端工厂
- 推荐动作：值得精读：抽象契约（批量 GET/PUT + 多缓冲变体）与工厂模式写得干净，是理解 SGLang 多模态全局缓存扩展点的重要入口。重点关注 EmbeddingCacheController 的注入点与 _io_loop 对后端异常的兜底处理；同时留意该抽象目前只有 Mooncake 一个实现，设计是否合理需待第二个后端落地后回看。

功能与动机

PR body 明确指出："Currently the multimodal (embedding) global cache is tightly coupled to the Mooncake backend, which makes it impossible to run the feature in environments where Mooncake is unavailable or not desired. This PR decouples the embedding cache from any specific backend so that different storage backends can be plugged in." 作者在 review 回复中也强调："today the multimodal embedding cache is hard-wired to Mooncake, so the feature simply cannot be enabled in environments where Mooncake is not deployed or not desired."

实现拆解

实现按 5 步拆解：

1. 新增抽象层与工厂（python/sglang/srt/mem_cache/embedding_store.py，新增 127 行）：定义 EmbeddingStore 抽象基类，抽象方法包括 batch_get、batch_put、batch_get_into_multi_buffers、batch_put_from_multi_buffers、batch_is_exist，并给出 register_buffer、get_key 的默认实现；同时实现 EmbeddingStoreFactory，以模块路径 + 类名注册后端，加载时校验类型、错误信息包含可用后端列表。模块级直接注册 mooncake 后端。
2. 迁移并改造 EmbeddingCacheController（从 mem_cache/storage/mooncake_store/embedding_cache_controller.py 移至 mem_cache/embedding_cache_controller.py）：构造函数新增必填参数 embedding_store: EmbeddingStore，删除内部 self.mooncake_store = MooncakeEmbeddingStore() 的直接构造；_register_pool_buffer、_io_loop 中的 batch_get_into_multi_buffers / batch_put_from_multi_buffers、batch_is_exist 全部改走 self.embedding_store。控制器从此不再感知具体后端。

3. Mooncake 后端适配接口（mooncake_embedding_store.py）：MooncakeEmbeddingStore 改为多重继承 MooncakeBaseStore, EmbeddingStore，复用既有 Mooncake setup 逻辑，接口契约由抽象类保证。
4. 接线与配置（encode_server.py、server_args.py）：encode server 初始化时经 EmbeddingStoreFactory.create_backend(get_mm().mm_global_cache_backend) 创建实例并注入 controller；server_args.py 新增 mm_global_cache_backend 参数（choices=["mooncake"]，默认 "mooncake"，namespace mm），保证默认行为不变。
5. 测试配套（test/registered/unit/mem_cache/test_embedding_cache_controller.py）：更新 import 路径为迁移后的新模块，mock 属性由 mooncake_store 改为 embedding_store，确保现有单元测试继续覆盖控制器行为。

关键文件：

- python/sglang/srt/mem_cache/embedding_store.py（模块 存储抽象层；类别 source；类型 dependency-wiring；符号 EmbeddingStore, EmbeddingStoreFactory, batch_get, batch_put）：本 PR 的核心新增文件：定义 EmbeddingStore 抽象基类与 EmbeddingStoreFactory 注册工厂，是解耦 Mooncake 的关键抽象层。
- python/sglang/srt/mem_cache/embedding_cache_controller.py（模块 缓存控制器；类别 source；类型 rename-or-move；符号 EmbeddingCacheController, EmbeddingCacheController.init, _register_pool_buffer, _io_loop）：控制器从 mooncake_store 目录迁移至 mem_cache 根目录，并改为构造注入 embedding_store，是解耦的核心落点。
- python/sglang/srt/disaggregation/encode_server.py（模块 编码服务；类别 source；类型 dependency-wiring；符号 EncodeServer.init）：encode server 是多模态全局缓存的装配点，改为经工厂创建后端并注入控制器，完成解耦接线。
- python/sglang/srt/mem_cache/storage/mooncake_store/mooncake_embedding_store.py（模块 存储后端；类别 source；类型 core-logic；符号 MooncakeEmbeddingStore）：Mooncake 后端补实现 EmbeddingStore 接口，成为工厂默认注册的实现。
- python/sglang/srt/server_args.py（模块 启动参数；类别 source；类型 core-logic；符号 mm_global_cache_backend）：新增 mm_global_cache_backend 启动参数，为后端选择提供配置入口。
- test/registered/unit/mem_cache/test_embedding_cache_controller.py（模块 单元测试；类别 test；类型 test-coverage；符号 _make_controller）：单元测试随控制器迁移与属性改名同步适配，保证既有行为验证不失效。

关键符号：EmbeddingStore.batch_get, EmbeddingStore.batch_put, EmbeddingStore.batch_get_into_multi_buffers, EmbeddingStore.batch_put_from_multi_buffers, EmbeddingStore.batch_is_exist, EmbeddingStoreFactory.register_backend, EmbeddingStoreFactory.create_backend, EmbeddingCacheController.init, EmbeddingCacheController._register_pool_buffer, MooncakeEmbeddingStore.init

关键源码片段

python/sglang/srt/mem_cache/embedding_cache_controller.py

控制器从 mooncake_store 目录迁移至 mem_cache 根目录，并改为构造注入 embedding_store，是解耦的核心落点。

```
class EmbeddingCacheController:
    def __init__(
        self,
        tp_rank,
        tp_size,
        embedding_store: EmbeddingStore, # 由工厂创建后注入，控制器不再感知具体后端
        max_pool_size_gb=4.0,
        hidden_dims: dict = None,
        tp_group=None,
        all_rank_get=False,
        enable_eviction: bool = True,
        max_eviction_batch: int = 100,
        dtype: torch.dtype = torch.float32,
    ):
        self.tp_world_size = tp_size
        self.tp_group = tp_group
        self.tp_rank = tp_rank
        self.all_rank_get = all_rank_get
        self.hidden_dims = hidden_dims or {}
        # Pool dtype 必须与模型 embedding dtype 一致，避免拷贝时发生类型转换
        self.dtype = dtype
        self.element_size = _dtype_element_size(self.dtype)
        self.enable_eviction = enable_eviction
        self.max_eviction_batch = max_eviction_batch

        # 依赖注入：原先这里直接构造 MooncakeEmbeddingStore()，
        # 现在后端由上层工厂决定，Mooncake 仅是一个可替换实现
        self.embedding_store = embedding_store
        self.total_pool_size_bytes = int(max_pool_size_gb * 1024**3)
        self.vision_pool, self.audio_pool = self._create_pools(pin_memory=True)
        self.pools = {
            "vision": self.vision_pool,
            "audio": self.audio_pool,
        }
        self._register_pool_buffer(self.vision_pool)
        self._register_pool_buffer(self.audio_pool)
        ...

    def _register_pool_buffer(self, pool: EmbeddingPool):
        if pool.tensor.numel() == 0:
            logger.warning(
                f"[Rank {self.tp_rank}] {pool.modality} embedding pool has zero pages; "
                f"dim={pool.dim}, budget={pool.pool_size_bytes} bytes"
            )
```

```
            return
        # 把 host 侧 pool 注册给后端，Mooncake 可用它做零拷贝传输
        self.embedding_store.register_buffer(pool.tensor)
        logger.info(
            f"[Rank {self.tp_rank}] Registered {pool.modality} embedding pool: "
            f"dim={pool.dim}, pages={pool.num_pages}, "
            f"page_tokens={pool.page_size}, "
            f"capacity={pool.num_pages * pool.page_bytes / 1024**2:.2f} MB"
        )
```

评论区精华

核心讨论集中在两点：

1. 抽象是否超前：ShangmingCai 在 *embedding_store.py* 上提问 "Do we have another backend right now? It seems strange to refactor this in advance when there is no other option." 作者 liusy58 回应这正是本 PR 要解决的问题 —— 当前只有 *Mooncake* 后端，而全局缓存被硬编码绑定导致无 *Mooncake* 环境根本无法启用该功能，解耦是为了先打开接入新后端的口子。
2. 中间版本 FileEmbeddingStore 的代码质量问题：*gemini-code-assist* 对早期版本的 FileEmbeddingStore 提出高优先级问题 —— 多进程并发写同一 hash 时共享固定 tmp_path 会产生竞态与文件损坏；batch_get 仅捕获 FileNotFoundError 无法兜底其他 OSError；构造函数新增必填 *embedding_store* 参数破坏向后兼容。最终提交历史中出现 "*delete class FileEmbeddingStore*"，这些针对 File 后端的评论已被消解；兼容性意见虽未采纳（保持必填参数），但全部调用点与测试已同步更新。

- 只有 *Mooncake* 一个后端时提前抽象是否合理 (design): 作者确认抽象前置是为解除 *Mooncake* 硬绑定、打开接入新后端的口子，设计意图被接受。
- 中间版本 FileEmbeddingStore 的并发写竞态 (correctness): 最终提交删除 FileEmbeddingStore 类，相关竞态代码不再存在，问题随删除消解。
- *embedding_store* 必填参数破坏向后兼容 (correctness): 作者选择保持必填参数，同步更新了 *encode_server.py* 与测试中全部调用点，无遗留引用。
- FileEmbeddingStore 读取异常兜底不足 (correctness): 对应 FileEmbeddingStore 代码已在最终版本删除，问题不再存在；但当前 MooncakeEmbeddingStore 路径由 _io_loop 统一 *catch* Exception 兜底。

风险与影响

- 风险：风险点如下：
  - 构造函数签名变更：EmbeddingCacheController.__init__新增必填位置参数 *embedding_store*，任何外部直接实例化的代码（当前只有 *encode_server.py* 与测试）必须同步传入；遗留的旧 import 路径 sglang.srt.mem_cache.storage.mooncake_store.embedding_cache_controller 已失效，需确认仓库内无其它引用残留。
  - 抽象收益未经验证：当前 *choices* 只有 *mooncake*，工厂与抽象契约尚未被第二个真实后端验证，接口设计（如 batch_get_into_multi_buffers 的多缓冲语义）可能在新后端落

地时暴露不足。

- Mooncake 初始化时序：MooncakeEmbeddingStore 继承自 MooncakeBaseStore，其 setup 在构造时执行；若未来后端构造较重，直接放在 encode_server 初始化路径（持有重建锁之前）可能拉长启动时间，当前 Mooncake 行为保持不变。
- 测试覆盖：现有测试仅做 mock 注入适配，没有新增针对 EmbeddingStoreFactory 注册 / 加载失败路径的单元测试，异常分支（ImportError、AttributeError、非子类 TypeError）未被覆盖。

- 影响：影响范围中等：
  - 用户 / 运维：新增 --mm-global-cache-backend 启动参数（当前仅接受 mooncake），为无 Mooncake 环境启用全局缓存提供了配置入口；默认值 mooncake 保证现有部署零感知。
  - 系统：encode server 初始化链路从硬编码 Mooncake 改为工厂创建 + 依赖注入，EmbeddingCacheController 与存储实现解耦，后续新后端无需改动控制器与 encode server 主逻辑。
  - 团队：模块归属更清晰，存储后端可独立开发与注册；但控制器文件移动属于跨目录 break，需要维护者周知。
  - 风险标记：构造函数新增必填参数，控制器文件路径迁移，抽象接口缺少第二个后端验证，工厂加载失败路径未测试

关联脉络

- PR #34163 fix(vlm): preserve Kimi-K3 GPU JPEG accuracy: 与本 PR 共同修改 encode_server.py 的多模态处理路径，反映 encode server 在全局缓存与图像解码两侧持续演进；与本 PR 的缓存解耦无直接功能依赖。