PR #36295 完整报告

sgl-project/sglang

fix: bound CUDA memory for fast image preprocessing

合并时间：2026-08-26 09:02

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36295

执行摘要

- 一句话：请求级 CUDA 内存池限制图像预处理显存，修复 VLM VRAM 增长
- 推荐动作：值得精读。该 PR 用最小改动解决了 VLM 长跑服务中容易被忽略的显存保留问题，核心技巧是请求级 torch.cuda.MemPool 作用域化分配，适合迁移到其他存在临时显存需求的预处理 / 后处理链路。

功能与动机

PR body 指出：*potential vram leak is observed with fast-image-processor, due to the memory pool allocated by global cuda allocator held by the tokenizer processor is not released after the image feature is moved to CPU*。长周期 VLM 服务中，图像特征虽然已释放，但进程级 CUDA caching allocator 仍保留最大预处理 scratch 分配，导致 tokenizer 侧 VRAM 持续增长。

实现拆解

1. 在 base_processor.py 新增 _temporary_fast_processor_cuda_pool 上下文管理器：仅当 device 为 CUDA、keep_mm_features_on_device 为假且未启用 precompute_hash_before_cpu_transfer 时，创建独立 torch.cuda.MemPool 并通过 torch.cuda.use_mem_pool 作用域包裹；否则直接 yield 透传，保持原行为。
2. 重构 process_mm_data：将原先直接调用的 processor.__call__ 放入 with self._temporary_fast_processor_cuda_pool(processor_device)，并把特征张量搬移到 CPU 的循环保留在池上下文内，确保退出后池内显存一次性归还驱动。
3. 测试配套：在 test_processor_device_selection.py 新增 TestFastImageProcessorMemoryPool，用事件序列断言池的进入 / 退出包裹了处理器调用与 CPU 拷贝，并用参数化用例覆盖 cuda_ipc、cuda_vmm、预哈希、CPU 设备等不启用池的场景。

关键文件：

- python/sglang/srt/multimodal/processors/base_processor.py（模块 多模态；类别 source；类型 core-logic；符号 _temporary_fast_processor_cuda_pool, process_mm_data）：新增 _temporary_fast_processor_cuda_pool 并重构 process_mm_data，用请求级 MemPool 包裹处理器调用，是修复核心。

- test/registered/unit/multimodal/test_processor_device_selection.py（模块 单测；类别 test；类型 test-coverage；符号 TestFastImageProcessorMemoryPool, test_pool_is_limited_to_immediate_cpu_transport, test_processor_call_uses_private_pool_until_cpu_copy_finishes）：新增 TestFastImageProcessorMemoryPool 验证池启用条件与调用顺序，防止回归。

关键符号：_temporary_fast_processor_cuda_pool, process_mm_data

关键源码片段

python/sglang/srt/multimodal/processors/base_processor.py

新增 _temporary_fast_processor_cuda_pool 并重构 process_mm_data，用请求级 MemPool 包裹处理器调用，是修复核心。

```
from contextlib import contextmanager

@contextmanager
def _temporary_fast_processor_cuda_pool(self, device: Optional[str]):
    """Release fast-processor CUDA temporaries after CPU feature transport."""
    # 仅当满足以下条件时才启用请求级内存池：
    # - 设备确实为 CUDA；
    # - 特征不要求留在 GPU（keep_mm_features_on_device 为假）；
    # - 不需要先对 GPU 张量做哈希再搬到 CPU（此时哈希还要读显存）。
    # 否则直接透传，保持原有行为（cuda_ipc / cuda_vmm / 预哈希路径）。
    can_release = (
        device is not None
        and torch.device(device).type == "cuda"
        and not self.keep_mm_features_on_device
        and not self.precompute_hash_before_cpu_transfer
    )
    if not can_release:
        yield
        return

    # 在目标设备上创建独立 MemPool，处理器内部的临时张量都从该池分配；
    # 退出上下文后池被销毁，显存归还给 CUDA driver，避免进程级
    # caching allocator 长期保留预处理 scratch 空间的最高水位。
    with torch.cuda.device(device):
        pool = torch.cuda.MemPool()
    with torch.cuda.use_mem_pool(pool, device=device):
        yield

    processor_device = None
    if (
        hasattr(processor, "image_processor")
        and isinstance(processor.image_processor, BaseImageProcessor)
        and not self.disable_fast_image_processor
    ):
        processor_device = self._fast_image_processor_device(processor)
```

```
        if processor_device is not None:
            kwargs["device"] = processor_device

    # 用请求级内存池包裹整个处理器调用与随后的 CPU 拷贝：
    # 特征张量搬离 GPU 后，池内 scratch 显存一次性释放。
    with self._temporary_fast_processor_cuda_pool(processor_device):
        result = processor.__call__(
            text=[input_text],
            padding=True,
            return_tensors="pt",
            **kwargs,
        )
        # 默认路径：特征张量立即搬到 CPU，便于后续调度与复用；
        # 搬移完成后退出上下文，池内 CUDA 内存归还驱动。
        if (
            not self.keep_mm_features_on_device
            and not self.precompute_hash_before_cpu_transfer
        ):
            for feature_name in self.FEATURE_NAMES:
                if feature_name in result and isinstance(
                    result[feature_name], torch.Tensor
                ):
                    result[feature_name] = result[feature_name].to("cpu")
```

test/registered/unit/multimodal/test_processor_device_selection.py

新增 TestFastImageProcessorMemoryPool 验证池启用条件与调用顺序，防止回归。

```
  def test_processor_call_uses_private_pool_until_cpu_copy_finishes(self):
        """验证池的进入/退出时机包裹了处理器调用与 CPU 拷贝。"""

        class Processor:
            image_processor = ImageProcessor()
            tokenizer = SimpleNamespace(bos_token=None)

            def __call__(self, **kwargs):
                events.append(("call", kwargs["device"]))
                return {"pixel_values": feature}

        events = []
        processor = self._processor()
        processor._processor = Processor()
        processor._tokenizer = processor._processor.tokenizer
        processor._tokenizer_auto_adds_specials = False
        processor.disable_fast_image_processor = False
        processor.image_config = {}
        processor.video_config = {}
        processor.audio_config = {}
        processor.FEATURE_NAMES = ["pixel_values"]

        # 用 PoolContext 模拟 MemPool 上下文，记录进入 / 退出事件，
```

```
    # 断言事件顺序为：进入池 -> 处理器调用 -> 拷贝到 CPU -> 退出池。
    with patch(f"{BASE}.torch.cuda.use_mem_pool", return_value=PoolContext()):
        processor.process_mm_data("test", images=["image"])

    self.assertEqual(
        events,
        ["enter", ("call", "cuda:0"), ("copy", "cpu"), "exit"],
    )
```

评论区精华

本 PR 无 review 评论。技术决策集中在 PR body 的实测数据：fast-path 答案摘要与 PIL 一致，且第二次重放仍为 956 MiB，证明显存有界。作者明确保留了 CUDA IPC、CUDA VMM 和预哈希场景的行为不变。

- 暂无高价值评论线程

风险与影响

- 风险：主要风险来自 torch.cuda.MemPool 与 use_mem_pool 的版本兼容性，较旧 PyTorch 可能不支持该 API；请求级池每次创建 / 销毁有微小开销，但对预处理时长影响可忽略（实测中位数 0.545s vs 0.548s）；若未来新增需要跨请求存活的特征缓存（即 keep_mm_features_on_device 为真），该池会自动跳过，需注意新增路径是否也满足该条件。
- 影响：影响所有启用 fast image processor 的 VLM 服务（如 Qwen3-VL、GLM-4.6V），显存占用从随请求数线性增长变为有界，显著提升长跑服务稳定性并降低 OOM 概率。对 PIL 路径、CUDA IPC/VMM 传输及双遍精度校验路径无行为变化。
- 风险标记：核心路径变更，依赖新版 PyTorch MemPool API

关联脉络

- PR #35646 fix: detect cross-node multimodal transport by nnodes: 同属多模态特征从 GPU 到 CPU 的传输链路，本 PR 关注该链路上的显存生命周期。
- PR #36232 Refactor HiCache host pool management: 同样围绕显存 / 内存池生命周期管理，体现仓库内对临时显存占用的系统性收紧。