

PR #54646 完整报告

vllm-project/vllm

[Core][MRV2] Freeze gc during V2 CG capture; skip per-descriptor cleanup

合并时间: 2026-09-01 10:37

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54646>

执行摘要

- 一句话: 冻结捕获期 GC, V2 捕获提速约 6 倍
- 推荐动作: 值得精读。核心设计决策有三点: 一是把 V1 的 `_freeze_gc` 抽取为公共 context manager 而非复制, 避免两套 runner 行为漂移; 二是以 `gc.isenabled()` 作为跳过清理的条件, 巧妙复用 `freeze` 的副作用传递状态, 但这也构成隐式契约, 阅读时需注意; 三是保留 `VLLM_ENABLE_CUDAGRAPH_GC` opt-out, 兼顾依赖 GC 行为的特殊场景。建议关注后续是否补充 `freeze` 恢复语义的测试。

功能与动机

PR body 指出 V2 runner 的 `capture_model` 没有像 V1 runner 的 `_freeze_gc` 那样在 CUDA graph 捕获期间冻结 / 禁用 GC, 导致捕获暴露在 gc 周期风险下 (例如 Triton kernel 在捕获中途被 `finalize` 并卸载 module, 使捕获中的图失效); 同时 `breakable wrapper` 的 per-descriptor `gc.collect() + empty_cache()` 既必要又昂贵: 每个 descriptor 约 0.3s, 整体捕获时间被放大约 10 倍 (PR #54127 的 CI 上捕获从 8s 膨胀到 70s, H200 MIG, 且 memory profiling 阶段每个 engine 支付两次)。

实现拆解

1. 抽取公共 GC 冻结工具 (`vllm/utils/gc_utils.py`): 新增 `freeze_gc_for_cudagraph_capture()` context manager, 逻辑与 V1 原 `_freeze_gc` 完全一致——进入时先执行一次 `gc.collect()`, 然后 `gc.freeze() + gc.disable()` 冻结堆并禁用自动 GC; 退出时 `gc.unfreeze()` 并统一 `gc.collect()`, 最后按进入前的 `gc.isenabled()` 状态恢复。`VLLM_ENABLE_CUDAGRAPH_GC` 环境变量继续作为 opt-out, 未新增配置项。
2. V2 runner 接入 (`vllm/v1/worker/gpu/model_runner.py`): `capture_model()` 用 `with freeze_gc_for_cudagraph_capture()`: 包裹从 `empty_cache()`、encoder/decoder 捕获到读取 `end_free_gpu_memory` 的完整流程, 原本的显式 `gc.collect()` 被移除 (`freeze` 内部已覆盖), 内存统计与计时逻辑保持不变。
3. V1 runner 去重 (`vllm/v1/worker/gpu_model_runner.py`): 删除 23 行内联 `_freeze_gc` 实现, 替换为 `_freeze_gc = staticmethod(freeze_gc_for_cudagraph_capture)`, 既有调用点零改动, 行为等价。
4. `breakable wrapper` 条件化清理 (`vllm/compilation/breakable_cudagraph.py`): `_capture()` 中的 `gc.collect() + torch.accelerator.empty_cache()` 改为仅在 `gc.isenabled()` 时执行——bulk capture 期间 GC 已被禁用, `gc.isenabled()` 返回 False 从

而跳过每个 descriptor 约 0.3s 的清理；freeze 之外的懒捕获（lazy capture）仍保留完整清理，内存回收语义不变。

5. 测试与配置配套：无测试文件变更；未新增配置或 schema 改动。现有 Buildkite CI（#86452）覆盖回归验证。

关键文件：

- vllm/utils/gc_utils.py（模块 工具层；类别 source；类型 core-logic；符号 freeze_gc_for_cudagraph_capture）：新增 freeze_gc_for_cudagraph_capture context manager，是本次变更的核心逻辑载体，从 V1 runner 抽取并供 V2 runner 与 breakable wrapper 共用，决定捕获期间 GC 冻结与恢复的语义。
- vllm/v1/worker/gpu/model_runner.py（模块 模型运行；类别 source；类型 core-logic；符号 capture_model）：V2 runner 的 capture_model 是整个捕获流程的入口，本 PR 用 freeze 上下文包裹它，消除捕获中 GC 周期导致图失效的风险，并移除显式 gc.collect()。
- vllm/v1/worker/gpu_model_runner.py（模块 模型运行；类别 source；类型 refactor；符号 _freeze_gc）：V1 runner 的原 _freeze_gc 定义被删除，改为指向公共函数的静态别名，保证两代 runner 行为一致并消除重复实现。
- vllm/compilation/breakable_cudagraph.py（模块 图编译；类别 source；类型 core-logic；符号 _capture）：per-descriptor 清理是捕获时间被放大 10 倍的主因，条件化跳过是该性能回归的直接修复点，且需与 freeze 上下文配合保持语义。

关键符号：freeze_gc_for_cudagraph_capture, capture_model, _capture, _freeze_gc

关键源码片段

vllm/utils/gc_utils.py

新增 freeze_gc_for_cudagraph_capture context manager，是本次变更的核心逻辑载体，从 V1 runner 抽取并供 V2 runner 与 breakable wrapper 共用，决定捕获期间 GC 冻结与恢复的语义。

freeze_gc_for_cudagraph_capture (vllm/utils/gc_utils.py)

```
import gc
from contextlib import contextmanager

import vllm.envs as envs

@contextmanager
def freeze_gc_for_cudagraph_capture():
    """Freeze and disable gc for the duration of bulk CUDA graph capture.

    A gc cycle during stream capture can invalidate the captured graph,
    e.g. a finalized Triton kernel unloads its module. Opt out with
    VLLM_ENABLE_CUDAGRAPH_GC=1.
    """
    # 记住进入前的开关状态，退出时按原状恢复，而不是无条件 enable
```

```

gc_was_enabled = gc.isenabled()
# 进入捕获前先做一次完整回收，保证后续冻结时堆是干净的
gc.collect()
# 环境变量提供 opt-out，某些场景可能依赖捕获期间维持 GC 行为
should_freeze = not envs.VLLM_ENABLE_CUDAGRAPH_GC
if should_freeze:
    gc.freeze()
    # gc.disable 能防止捕获中途触发 gc 周期，避免 Triton kernel
    # finalize 时卸载 module 导致已捕获的图失效
    gc.disable()
try:
    yield
finally:
    if should_freeze:
        try:
            gc.unfreeze()
            # 退出后统一回收一次，替代 bulk capture 期间逐 descriptor 的清理
            gc.collect()
        finally:
            if gc_was_enabled:
                gc.enable()
            else:
                gc.disable()

```

vllm/v1/worker/gpu/model_runner.py

V2 runner 的 `capture_model` 是整个捕获流程的入口，本 PR 用 `freeze` 上下文包裹它，消除捕获中 GC 周期导致图失效的风险，并移除显式 `gc.collect()`。

capture_model 捕获入口 (vllm/v1/worker/gpu/model_runner.py)

```

@torch.inference_mode()
def capture_model(self) -> int:
    assert self.cudagraph_manager is not None
    capture_encoder = (
        self.model_state.supports_mm_inputs
        and self.model_state.encoder_runner.has_cudagraph()
    )
    capture_decoder = self.cudagraph_manager.needs_capture()
    if not capture_encoder and not capture_decoder:
        return 0

    start_time = time.perf_counter()
    # 用公共 freeze 上下文包住整个 bulk capture：期间的 gc 周期
    # 可能 finalize Triton kernel 并卸载其 module，使捕获中的图失效
    with freeze_gc_for_cudagraph_capture():
        torch.accelerator.empty_cache()
        start_free_gpu_memory = torch.accelerator.get_memory_info()[0]

    with self.maybe_setup_dummy_loras(self.lora_config):

```

```

if capture_encoder:
    self.model_state.encoder_runner.capture()
if capture_decoder:
    input_buffers = self.input_buffers
    if self.pcp_manager is not None:
        input_buffers = self.pcp_manager.input_buffers
    self.cudagraph_manager.capture(
        self.model,
        self.model_state,
        input_buffers,
        self.intermediate_tensors,
        self.block_tables,
        self.attn_groups,
        self.kv_cache_config,
        pcp_manager=self.pcp_manager,
        has_lora=self.lora_config is not None,
        use_aux_hidden_state_outputs=self.use_aux_hidden_state_outputs,
        lora_capture_hook=create_lora_capture_hook(
            self.lora_config, self
        ),
    )
if self.speculator is not None:
    with use_workspace_lane(self._draft_workspace_lane):
        self.speculator.capture()
if self.adaptive_verification is not None:
    with self.step_timing.collect() as timings:
        for batch in self.adaptive_verification.batches_to_profile(
            self.cudagraph_manager.captured_token_counts()
        ):
            self._dummy_run(**batch)
        self.adaptive_verification.set_initial_cost_curves(timings)

end_free_gpu_memory = torch.accelerator.get_memory_info()[0]

elapsed_time = time.perf_counter() - start_time
cuda_graph_size = start_free_gpu_memory - end_free_gpu_memory
logger.info(
    "Graph capturing finished in %.0f secs, took %.2f GiB",
    elapsed_time,
    cuda_graph_size / (1 << 30),
)
return cuda_graph_size

```

vllm/compilation/breakable_cudagraph.py

per-descriptor 清理是捕获时间被放大 10 倍的主因，条件化跳过是该性能回归的直接修复点，且需与 freeze 上下文配合保持语义。

_capture 条件化清理 (vllm/compilation/breakable_cudagraph.py)

```

def _capture(
    self,
    entry: _BreakableEntry,
    args: tuple[Any, ...],
    kwargs: dict[str, Any],
) -> Any:
    validate_cudagraph_capturing_enabled()
    entry.input_addresses = self._collect_tensor_addresses(args, kwargs)

    if self.graph_pool is not None:
        set_graph_pool_id(self.graph_pool)
    else:
        set_graph_pool_id(current_platform.graph_pool_handle())

    # 手动驱动 capture_begin/end 绕过了 torch.cuda.graph() 自带的
    # gc + empty_cache, 因此这里原本要模拟它的 pre-capture 清理。
    # 但 bulk capture 已由 freeze_gc_for_cudagraph_capture 统一清理过,
    # 每个 descriptor 重复执行约耗时 0.3s, 成为捕获时间被放大 10 倍的主因,
    # 所以在 gc 被禁用 (冻结) 时跳过; lazy capture 仍保留完整清理。
    if gc.isenabled():
        gc.collect()
        torch.accelerator.empty_cache()
    # 同步 offloader 的 copy stream, 避免 pre-capture 的预取泄漏进图
    get_offloader().sync_prev_onload()

    capture = BreakableCUDAGraphCapture(pool=self.graph_pool)
    with capture:
        output = self.runnable(*args, **kwargs)
        # 在最后一个 segment 关闭前 join offloader 的 copy stream,
        # 否则后续 forward 会报 "unjoined stream" 错误
        get_offloader().join_after_forward()
        # 在 capture 上下文内转弱引用, 让 cudagraph pool 立即复用内存
        output = weak_ref_tensors(output)

    entry.capture = capture
    entry.output = weak_ref_tensors(output)
    return output

```

评论区精华

benchislett 在 `vllm/compilation/breakable_cudagraph.py:374` 提出疑问: GC 禁用时 `empty_cache` 是否也应省略? 为什么? njhill 确认: 是的, 应同时省略, 以避免承受不必要的 per-descriptor 开销。该讨论澄清了冻结清理与逐 descriptor 清理的职责边界——bulk capture 下前者已统一完成, 后者只服务于 freeze 之外的懒捕获路径。两位 reviewer (khluu、benchislett) 均批准, 无未解决疑虑。

- GC 禁用时是否应省略 `empty_cache` (performance): njhill 确认应同时省略, 目的是避免不必要的 per-descriptor 开销; `empty_cache` 与 `gc.collect` 一起由 freeze 上下文的统一清

理覆盖。

风险与影响

- 风险：正确性风险低：冻结逻辑与 V1 原实现逐行等价，V1 路径行为不变，且 `VLLM_ENABLE_CUDAGRAPH_GC=1` 可完全退出冻结行为。主要风险是 `gc.isenabled()` 作为跳过 per-descriptor 清理的条件，隐含“bulk capture 一定运行在 freeze 下”的契约；当前由 `capture_model` 唯一入口保证，但若未来新增批量捕获调用路径且未处于 freeze 上下文，可能跳过必要的内存回收。此外 `gc.freeze()/disable()` 是全局状态修改，若捕获中途异常，context manager 的 finally 仍会恢复状态，但该路径没有专门测试覆盖。本 PR 未附测试文件，`VLLM_ENABLE_CUDAGRAPH_GC` 开关与冻结恢复的状态组合建议补一条回归测试。
- 影响：影响所有启用 CUDA graph（含 MRV2/breakable 路径）的用户：引擎启动与 memory profiling 阶段显著提速，H200 上初始化 28.9s -> 8.2s，且消除了捕获中途被 GC 周期失效的潜在不稳定。对 V1 runner 无行为变化（逻辑等价）；breakable 懒捕获路径保持原有清理语义，不影响在线捕获正确性。团队层面，公共工具落入 `vllm.utils.gc_utils`，后续断点捕获、profiling 等场景可复用。影响程度中等偏上，主要在启动路径而非在线推理路径。
- 风险标记：核心路径变更，缺少测试覆盖，全局 GC 状态修改，环境变量可退出

关联脉络

- PR #54127（标题未在上下文中提供）：PR body 明确提到在其 CI 上观察到 breakable 默认开启时捕获时间从 8s 膨胀到 70s（H200 MIG，memory profiling 阶段每个 engine 支付两次），是本 PR 优化动机的直接来源；材料未提供该 PR 标题。