PR #35375 完整报告

sgl-project/sglang

[Memory] Borrow CUDA graph pool storage for EAGLE sampling

合并时间：2026-08-19 07:19

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35375

执行摘要

- 一句话：借 CUDA 图池空闲块承载 EAGLE 采样内存，默认关闭
- 推荐动作：值得精读：pool.py 的借用、退役与互斥 scope 设计，以及 BumpArenaStub 的多 extent 改造是全文最有参考价值的部分；任何需要把步内临时张量塞进既有预留内存的场景都可借鉴该模式。对只关注 EAGLE 的用户，建议先跑 test_graph_pool_borrow.py 与 test_spec_eagle*.py 再决定是否开启。合入前请留意两个遗留问题：PR 级 CI 槽位失败原因未澄清、借用内存缺少逃逸检测的运行时断言。

功能与动机

PR body 的目标句是 reuse those runs for EAGLE's step-local full-vocabulary verification probabilities。EAGLE 在非贪心采样时需要构造 target_probs、draft_probs、coins 等形状为 (bs * num_draft_tokens, vocab_size) 的步骤内临时张量，正常情况下它们从通用缓存分配器申请，抬高 GPU reserved 水位；而共享 graph pool 在 prefill / decode 回放期间存在大量 idle 的空闲块。作者希望用内存借用让两者互补，同时保持 no silent fallback、capture 前退役借用池、CUDA-only 且默认关闭的安全边界。仓库未关联 Issue，动机完全来自 PR body。

实现拆解

1. 泛化 bump 分配器（python/sglang/srt/cuda_vmm_utils.py + python/sglang/srt/mem_cache/kv_vmm_backing.py） - 新增 BumpArenaStub 类与 _bump_arena_stub_source() 生成的 JIT C 源码，支持最多 BUMPARENA_MAX_EXTENTS = 64 个调用方提供的 (base, nbytes) extents；malloc 按 first-fit 分配、free 只记账不回收，块复用交给上层 torch.cuda.MemPool 缓存层。 - 每个实例使用 os.getpid()_实例序号 符号后缀、独立 build 目录与独立 .so，避免 co-located 引擎进程共享 tempdir 时发生编译或加载竞态。 - KvVmmArena 删除自带的 _stub_source / _build_stub（约 93 行），改为 self._stub = BumpArenaStub() 后 set_extents([(self.base, self.reserved)])，语义不变，让 KV 缓存与图池借用共用同一实现源。
2. 建立借用基础设施（python/sglang/srt/model_executor/runner_utils/pool.py，+179/-1） - 模块级状态：_borrow_stub、_borrow_mem_pool、_borrow_static_runs、_active_graph_pool_user 等。 - graph_pool_borrow_enabled()：要求 SGLANG_ENABLE_GRAPH_POOL_BORROW 开启、is_cuda() 且未被 disable_graph_pool_borrow() 禁用；若注册了静态 runs 则依其非空判断，否则依赖全局

图池句柄存在。 - find_free_graph_pool_runs()：对 torch.cuda.memory_snapshot(pool_id) 中 allocated_size == 0 的整段空闲 segment，把连续 inactive block 拼成 run 并按大小降序返回。 - borrow_graph_pool(user)：在互斥 scope 内懒建 torch.cuda.MemPool，把 torch.empty 等分配路由到图池空闲 run；当 freed_bytes 非零（empty_cache 释放了段）或 cursor 消耗超过总 extents 一半时重建池；_teardown_borrow_pool() 先 synchronize() 再触发一次小分配，排空跨流延迟释放。 - graph_pool_user_scope / graph_pool_replay_scope / graph_pool_capture_scope：三者互斥，冲突直接 RuntimeError 并报告冲突用户；capture 进入前退役借用池，因为捕获会重新雕刻图池空闲区。

3. 接入全部 CUDA graph 后端 - full_cuda_graph_backend.py、breakable_cuda_graph_backend.py、compilation/cuda_piecewise_backend.py 的 capture 与 replay 分别包上 graph_pool_capture_scope() / graph_pool_replay_scope()。 - multimodal/kimi_k3_vit_cuda_graph_runner.py 的命中回放与新增捕获同样包上对应 scope，并把 self.graphs[key] = entry 之后的首次 replay 移入 capture scope 内。
4. EAGLE 采样接入（python/sglang/srt/speculative/eagle_utils.py） - eagle_sample() 的非贪心分支整体包进 with borrow_graph_pool(user="EAGLE probability borrow")，expanded_temperature / target_probs / draft_probs / coins / coins_for_final_sampling 全部在 scope 内分配，并在 scope 结束前显式 del，确保借用块不活过当前 step。
5. 配置与测试配套 - python/sglang/srt/environ.py 新增 SGLANG_ENABLE_GRAPH_POOL_BORROW = EnvBool(False)。 - 新增 test/registered/unit/model_executor/runner_utils/test_graph_pool_borrow.py（306行），覆盖：借用期间 replay 报错、外部托管图存储可禁用借用、EAGLE 概率计算确实处于借用 scope、借用分配落在图池 run 上且可回收复用、静态 runs 无需池快照、跨流延迟释放不冲突。 - test/registered/spec/eagle/test_spec_eagle.py 增加用该 env 开启 feature 的变体执行。

关键文件：

- python/sglang/srt/model_executor/runner_utils/pool.py（模块 图池管理；类别 source；类型 data-contract；符号 disable_graph_pool_borrow, set_graph_pool_borrow_runs, graph_pool_borrow_enabled, graph_pool_user_scope）：本 PR 的核心：新增借用池基础设施，包括启用判断、互斥 scope、空闲 run 扫描、持久 MemPool 借用与退役逻辑，并定义了整个 feature 的数据契约。
- python/sglang/srt/cuda_vmm_utils.py（模块 虚拟内存；类别 source；类型 core-logic；符号 _bump_arena_stub_source, BumpArenaStub, BumpArenaStub.init, BumpArenaStub._build）：新增 BumpArenaStub：把 KV VMM 原有的单 extent bump 分配器泛化为多 extent、可被外部调用方设置地址区间的通用 JIT 分配器，是借用池与 KV VMM arena 共用的底层机制。
- test/registered/unit/model_executor/runner_utils/test_graph_pool_borrow.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestGraphPoolBorrow, TestGraphPoolBorrow.setUp, TestGraphPoolBorrow.tearDown, TestGraphPoolBorrow._reset_borrow_state）：新增 306 行专项测试，覆盖互斥、禁用、

EAGLE scope 接入、地址落在图池 run、静态 runs、跨流延迟释放等关键安全边界，是本 feature 可信度的重要支撑。

- python/sglang/srt/mem_cache/kv_vmm_backing.py（模块 缓存分配；类别 source；类型 core-logic；符号 KvVmmArena.init, KvVmmArena.cursor_bytes, BumpArenaStub）：把 KvVmmArena 自带的单 extent bump 实现替换为共享的 BumpArenaStub，删除约 93 行私有 JIT 代码，是 ' 泛化现有 KV VMM 分配器 ' 这一目标的具体落地。
- python/sglang/srt/speculative/eagle_utils.py（模块 推测解码；类别 source；类型 dependency-wiring；符号 eagle_sample）：EAGLE 是借用机制的消费者：非贪心分支的全词表概率计算整体包进 borrow_graph_pool scope，是本 feature 的实际收益点。
- python/sglang/srt/multimodal/kimi_k3_vit_cuda_graph_runner.py（模块 视觉图执行；类别 source；类型 core-logic；符号 KimiK3ViTCudaGraphRunner.run）：Kimi-K3 ViT 图执行器新增 capture / replay scope 包裹，保证多模态图执行路径也遵守互斥协议。
- python/sglang/srt/model_executor/runner_backend/full_cuda_graph_backend.py（模块 图后端；类别 source；类型 data-contract；符号 FullCudaGraphBackend.capture_one, FullCudaGraphBackend.replay）：主图后端 capture_one / replay 接入互斥 scope，是借用机制在图回放路径上的关键护栏。
- python/sglang/srt/compilation/cuda_piecewise_backend.py（模块 图后端；类别 source；类型 dependency-wiring；符号 CudaPiecewiseBackend.call）：piecewise 图编译执行路径同样接入 scope，与标签 piecewise-cuda-graph 对应，保证该后端在借用期不回放。
- python/sglang/srt/model_executor/runner_backend/breakable_cuda_graph_backend.py（模块 图后端；类别 source；类型 data-contract；符号 BreakableCudaGraphBackend.capture_one, BreakableCudaGraphBackend.replay）：breakable 图后端同样接入 scope，防止可打断图路径在借用期间回放造成覆盖。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 configuration；符号 SGLANG_ENABLE_GRAPH_POOL_BORROW）：新增 SGLANG_ENABLE_GRAPH_POOL_BORROW 环境变量开关，默认 False，保证功能 opt-in 且默认 no-op。
- test/registered/spec/eagle/test_spec_eagle.py（模块 推测测试；类别 test；类型 test-coverage）：在 spec/eagle 主测试中增加开启借用 feature 的变体，验证 EAGLE 采样在默认场景外的新路径下仍正确。

关键符号：borrow_graph_pool, find_free_graph_pool_runs, graph_pool_user_scope, graph_pool_capture_scope, graph_pool_replay_scope, graph_pool_borrow_enabled, disable_graph_pool_borrow, set_graph_pool_borrow_runs, _teardown_borrow_pool, BumpArenaStub.init, BumpArenaStub._build, BumpArenaStub.set_extents, KvVmmArena.init, eagle_sample, KimiK3ViTCudaGraphRunner.run, FullCudaGraphBackend.capture_one, FullCudaGraphBackend.replay, BreakableCudaGraphBackend.replay

关键源码片段

python/sglang/srt/model_executor/runner_utils/pool.py

本 PR 的核心：新增借用池基础设施，包括启用判断、互斥 scope、空闲 run 扫描、持久 MemPool 借用与退役逻辑，并定义了整个 feature 的数据契约。

```
# ===== pool.py 借用池核心 =====
# 图回放会静默覆盖其分配器空闲块上的别名数据，因此 capture / replay / borrow
# 必须互斥；冲突时直接抛错并报告当前占用者
@contextmanager
def graph_pool_user_scope(user: str) -> Iterator[None]:
    global _active_graph_pool_user
    if _active_graph_pool_user is not None:
        raise RuntimeError(
            f"graph pool already has live user {_active_graph_pool_user!r}; "
            f"cannot use it for {user!r}"
        )
    _active_graph_pool_user = user
    try:
        yield
    finally:
        _active_graph_pool_user = None


# 只扫描 allocated_size 为 0 的整段空闲 segment，再把其中连续 inactive 块拼成
# run，按大小降序返回；绝不去碰仍被图占用的内存
# 返回的 run 即安全暴露给 eager 分配的空闲地址区间
# 返回的 run 即安全暴露给 eager 分配的空闲地址区间
def find_free_graph_pool_runs(pool_id: Any) -> list[tuple[int, int]]:
    runs: list[tuple[int, int]] = []
    for segment in torch.cuda.memory_snapshot(pool_id, include_traces=False):
        if segment["allocated_size"] != 0:
            continue
        run_address = 0
        run_bytes = 0
        for block in segment["blocks"]:
            if block["state"] == "inactive":
                if run_bytes == 0:
                    run_address = block["address"]
                run_bytes += block["size"]
                continue
            if run_bytes:
                runs.append((run_address, run_bytes))
            run_bytes = 0
        if run_bytes:
            runs.append((run_address, run_bytes))
    runs.sort(key=lambda run: run[1], reverse=True)
    return runs


@contextmanager
def borrow_graph_pool(user: str) -> Iterator[None]:
```

```
    # scope 内分配的张量不能活过当前 step，下一次图回放可能覆盖其字节；
    # 没有任何 run 能容纳的分配走普通 OOM，不静默回退
    global _borrow_stub, _borrow_mem_pool, _borrow_extents_total
    if not graph_pool_borrow_enabled():
        yield
        return
    with graph_pool_user_scope(user):
        if _borrow_mem_pool is not None:
            # 推进一次小分配，让缓存分配器处理跨流延迟释放的事件
            torch.empty(1, device="cuda")
            if (
                _borrow_stub.freed_bytes
                or _borrow_stub.cursor_bytes > _borrow_extents_total // 2
            ):
                # empty_cache 释放了段，或未决释放吃掉一半 extents：重建借用池
                _teardown_borrow_pool()
        if _borrow_mem_pool is None:
            if _borrow_stub is None:
                _borrow_stub = BumpArenaStub()
            if _borrow_static_runs is not None:
                runs = _borrow_static_runs # 外部托管且地址稳定的固定 extents
            else:
                runs = find_free_graph_pool_runs(get_global_graph_memory_pool())[
                    : BumpArenaStub.MAX_EXTENTS
                ]
            _borrow_stub.set_extents(runs)
            # 保留一层缓存：普通块复用与流序延迟释放仍由分配器管理；
            # 捕获时因为底层空闲段变了，所以要整体退役
            _borrow_mem_pool = torch.cuda.MemPool(_borrow_stub.allocator)
            _borrow_extents_total = sum(run_bytes for _, run_bytes in runs)
        with torch.cuda.use_mem_pool(_borrow_mem_pool):
            yield
```

python/sglang/srt/cuda_vmm_utils.py

新增 BumpArenaStub：把 KV VMM 原有的单 extent bump 分配器泛化为多 extent、可被外部调用方设置地址区间的通用 JIT 分配器，是借用池与 KV VMM arena 共用的底层机制。

```
// ===== cuda_vmm_utils.py 内嵌 JIT C stub：多 extent bump 分配器 =====
// malloc 按 first-fit 遍历调用方注册的 extents，返回 base + cursor；
// 上限是各 extent 的 RESERVED 大小而非已提交水位，便于上界张量先分配后提交
void* bumparena_malloc_{sfx}(size_t size, int device, void* stream){
  std::lock_guard<std::mutex> lk(g_mu);
  for (size_t i = 0; i < g_num_extents; ++i) {
    size_t need = g_cursors[i] + align_up(size, g_align);
    if (need > g_reserved[i]) continue; // 当前 extent 放不下，换下一个
    void* p = reinterpret_cast<void*>(g_bases[i] + g_cursors[i]);
    g_cursors[i] = need;
    return p;
  }
```

```
  return 0; // 没有任何 extent 能容纳，表现为上层普通 OOM
}

// free 只记账：块复用交给 torch.cuda.MemPool 的缓存层，
// 这里统计 freed_bytes 供上层判断是否需要重建借用池
void bumparena_free_{sfx}(void* ptr, size_t size, int device, void* stream){
  std::lock_guard<std::mutex> lk(g_mu);
  g_freed_bytes += size;
}

// set_extents 重置所有 cursor，让同一个 stub 服务连续的若干组区域
void bumparena_set_extents_{sfx}(const uintptr_t* bases, const size_t* sizes, size_t n){
  std::lock_guard<std::mutex> lk(g_mu);
  if (n > BUMPARENA_MAX_EXTENTS) n = BUMPARENA_MAX_EXTENTS;
  g_num_extents = n;
  g_freed_bytes = 0;
  for (size_t i = 0; i < n; ++i) {
    g_bases[i] = bases[i];
    g_reserved[i] = sizes[i];
    g_cursors[i] = 0;
  }
}
```

python/sglang/srt/speculative/eagle_utils.py

EAGLE 是借用机制的消费者：非贪心分支的全词表概率计算整体包进 borrow_graph_pool scope，是本 feature 的实际收益点。

```
# ===== eagle_utils.py 非贪心分支：借用图池空闲 run 承载步骤内概率张量 =====
# 这些全词表矩阵只在本 step 内被采样 kernel 消费；predict 等返回张量
# 在 scope 之前就已分配，因此下一次图回放可以安全回收这些借用字节
with borrow_graph_pool(user="EAGLE probability borrow"):
    expanded_temperature = torch.repeat_interleave(
        sampling_info.temperatures, verify_input.draft_token_num, dim=0
    ) # (bs * num_draft_tokens, 1)

    target_probs = F.softmax(
        next_token_logits / expanded_temperature, dim=-1
    ) # (bs * num_draft_tokens, vocab_size)
    maybe_detect_nan(target_probs, "v2 verify: target_probs after softmax")
    if sampling_info.need_top_k_sampling:
        target_probs = top_k_renorm_prob(
            target_probs,
            torch.repeat_interleave(
                sampling_info.top_ks, verify_input.draft_token_num, dim=0
            ),
        )
    if sampling_info.need_top_p_sampling:
        target_probs = top_p_renorm_prob(
            target_probs,
```

```
            torch.repeat_interleave(
                sampling_info.top_ps, verify_input.draft_token_num, dim=0
            ),
        )
    target_probs = target_probs.reshape(bs, verify_input.draft_token_num, -1)
    draft_probs = (
        verify_input.draft_probs
        if use_rejection_sampling
        else torch.zeros_like(target_probs)
    )
    # 防御性校验：拒绝采样要求 draft 提议分布与目标词表一致
    if use_rejection_sampling and (
        draft_probs is None or draft_probs.shape[-1] != target_probs.shape[-1]
    ):
        raise ValueError(
            "Rejection sampling requires a target-vocab draft proposal distribution"
        )

    sampling_fn(
        predicts=predict,
        accept_index=accept_index,
        accept_token_num=num_correct_drafts,
        candidates=candidates,
        target_probs=target_probs,
        draft_probs=draft_probs,
        uniform_samples=coins,
        uniform_samples_for_final_sampling=coins_for_final_sampling,
        threshold_single=get_spec().speculative_accept_threshold_single,
        threshold_acc=get_spec().speculative_accept_threshold_acc,
        deterministic=True,
    )
    # 显式释放，让借用块回到 MemPool 缓存，绝不活过当前 step
    del (expanded_temperature, target_probs, draft_probs, coins, coins_for_final_sampling)
```

评论区精华

本 PR 的 review_comments_count 为 0，没有代码评审交锋；6 条 issue 评论均为 /rerun-group 命令与 CI bot 结果。作者先后三次重跑 unit/model_executor/runner_utils 与 spec/eagle 组：1-gpu-5090（6 个）、1-gpu-h100（5 个）、4-gpu-b200（1 个）用例全部通过。值得注意的是 PR body 中的 Latest PR Test (Base) / (Extra) 两个 CI 槽位显示失败（x），但任何评论都未解释失败根因，安全性论证完全依赖作者在 PR body Safety 一节中声明的设计约束：只暴露整段空闲 segment 中的 inactive run、无 run 可满足时正常 OOM 且不静默回退、capture 前退役借用池、capture / replay / borrow 互斥并报告冲突用户。

- EAGLE 与 runner_utils 用例在 CI 中反复重跑 (testing): 受影响回归用例在 5090 / H100 / B200 上均全绿；PR 级默认 CI 槽位的失败未被澄清，可能仅靠定向重跑确认后合入。
- 借用机制的安全边界设计（无 review 评论，作者在 body 中声明） (design): 这些约束已由 graph_pool_user_scope、graph_pool_capture_scope、_teardown_borrow_pool 等实现

，并有对应单测覆盖；但借用容量受整段空闲 segment 大小限制这一点仍是潜在 OOM 来源。

风险与影响

- 风险：
  1. 别名 / 生命周期风险（高）：借用块与图池内存地址重叠，若任何借用张量逃逸出当前 step，下一次图回放会静默覆盖其字节，产生难以定位的正确性损坏。目前仅靠 eagle_utils.py 中 scope 内显式 del 与注释约束，缺少运行时断言。
  2. OOM 无回退（中）：借用容量受整段空闲 segment 大小限制（find_free_graph_pool_runs 只接受 allocated_size == 0 的 segment），大 batch 尖峰时 EAGLE 概率矩阵需求可能超过空闲段容量，直接以分配器 OOM 中断服务，没有软降级路径。
  3. 互斥保护完整性（中）：保护依赖所有 CUDA graph 后端都接入 graph_pool_replay_scope / graph_pool_capture_scope；未来若新增后端或 torch.compile / XLA 路径漏包 scope，借用期间回放不会报错而是默默覆盖。
  4. JIT 构建风险（低 - 中）：BumpArenaStub 每次实例化都在 tempdir 编译 .so，首次构建有启动开销；虽已用 per-process / per-instance 后缀与独立 build 目录缓解竞态，但多进程并发首启仍可能触发。
  5. 重构回归（低 - 中）：kv_vmm_backing.py 改用共享 BumpArenaStub 后语义应完全一致（对齐、per-buffer slack、commit_range 逻辑未变），但属于核心缓存分配路径，需依赖既有 KV / KV-VMM 测试兜底。 - 影响：用户侧：默认无感，未设置 SGLANG_ENABLE_GRAPH_POOL_BORROW 时整条路径是 no-op；开启后 EAGLE 非贪心采样的 GPU reserved 内存有望下降（幅度取决于图池空闲 run 与概率矩阵大小之比），对 piecewise CUDA graph 共享池场景收益最明显。系统侧：共享图池的空闲段成为第二分配源，与 KV VMM arena 共用同一套 bump 机制，cuda_vmm_utils.py 成为 bump 分配器的唯一实现源。团队侧：所有 CUDA graph 后端（full / breakable / piecewise / kimi-k3）都必须遵守 capture / replay scope 契约，新增后端时需注意接入；pool.py 的进程级借用状态（_borrow_stub、_borrow_mem_pool 等）成为新的全局约束点。 - 风险标记：借用内存与图回放存在别名覆盖风险，无 run 可满足时直接 OOM 而无回退，互斥保护依赖所有图后端接入 scope，JIT stub 构建在 tempdir 存在竞态隐患，PR 级 CI 槽位失败未被澄清

关联脉络

- PR #35382 [Refactor] Share the page-aligned decode alloc lens between EAGLE and DFLASH: 同属 speculative decoding 内存分配共享化方向，且都涉及 eagle_utils.py 与 allocation_sizing.py，本 PR 进一步把 KV VMM bump 分配器抽为共享组件。
- PR #35265 [Spec] Page-align the DFLASH decode KV reservation: 同一 speculative decoding + KV 分配领域，关注解码期 KV 预留的页对齐与记账；本 PR 泛化的 BumpArenaStub 正是此类分配机制的底层实现。
- PR #35164 Refactor kv cache event mixin into a recorder: 同属 mem_cache 基础设施重构家族，趋势是把散落的缓存 / 分配逻辑收敛为统一组件，与本次 KvVmmArena 改用共

享 stub 方向一致。

- PR #34890 [Perf] Hoist DSv4 draft-extend SWA write locs; unify SWA graph buffer naming: 同属 speculative decoding 对 CUDA graph 内临时内存的精细管理，说明 EAGLE / DSv4 路径的内存布局一直是性能优化重点。