PR #35275 完整报告

sgl-project/sglang

[Bug][Spec] fix startup crash and reduce CUDA graph memory usage for speculative adaptive

合并时间：2026-08-27 15:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35275

执行摘要

- 一句话：修复 adaptive 投机解码启动崩溃并削减 CUDA graph 内存
- 推荐动作：值得精读。该 PR 展示了三个可借鉴的设计决策：① max_decode_logits_rows() 从「单宽度静态定容」改为「遍历全部 candidate 宽度取最大值」，消除配置解析与 buffer 分配之间的不一致；② warmup hook 必须从 capture 实际使用的 backend 解析，任何依赖 model_runner 全局状态的捕获逻辑在 adaptive 多 runner 场景下都不可靠；③ 利用 CUDA caching allocator 按 stream 分区 graph pool segments 的特性，通过进程级共享一条 capture stream 让后续 pass 复用前次留下的 inactive scratch，这是理解 CUDA graph 内存复用的关键机理。建议后续跟进两个点：共享 stream 的串行前提是否有运行时断言保护，以及 IMA 修复是否值得补一个端到端 GPU 回归用例。

功能与动机

PR body 明确列出三个独立问题并引用 issue #30549：① ModelRunner.max_decode_logits_rows() 用静态的 decode_num_tokens_per_req() 为共享 logits buffer 定容，但 adaptive speculative decoding 会重放为更宽 draft-token 宽度构建的 runner，buffer 过小导致服务器启动失败，对应报错 shared logits buffer holds 128 rows but caller needs 256；② DecodeCudaGraphRunner.capture_one_shape 从 self.model_runner.attn_backend 查找 on_after_cuda_graph_warmup，而 adaptive 额外 runner 绑定的是自己新建的 backend，错误 hook 把 warmup 升级后的元数据（raw→full、FlashMLA scheduler meta）冻结进 graph，首次重放触发非法内存访问；③ 每次 capture pass 未显式指定 stream，缓存分配器按 stream 分区 graph pool segments，导致每个 pass 重复预留池内 scratch，adaptive 场景下产生数十 GB 只读回 inactive 的 segments。

实现拆解

按 4 步拆解实现过程：

1. 共享 logits buffer 按全部 candidate 宽度定容（python/sglang/srt/model_executor/model_runner.py） - 重写 max_decode_logits_rows()：先调用 get_spec() 读取运行时解析后的 spec；收集 max_speculative_num_draft_tokens() 作为默认宽度，若 spec.speculative_adaptive 为真，再追加 resolve_candidate_steps_from_config(spec.speculative_adaptive_config) 得到的每个 steps + 1 宽度。 - 对每个宽度调用 decode_num_tokens_per_req(num_draft_tokens=...) 得到该宽度下每请求 token 数，经

get_batch_sizes_to_capture 计算最大行数，最终取所有宽度的最大值。 - 原因：adaptive 会为更宽的 candidate 重放 decode graph，buffer 必须按最宽形状分配；同时从 runtime context（bags）读取有效值，避免启动记录与运行时解析结果不一致。 - 配套：新增 import max_speculative_num_draft_tokens 与 resolve_candidate_steps_from_config。

2. 修复首次重放 IMA：warmup hook 改从 capture-local backend 解析（python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py） - capture_one_shape 内 post_warmup_hook 改为 getattr(attn_backend, "on_after_cuda_graph_warmup", None)，其中 attn_backend 来自 capture_prepare 返回的、本次 capture 实际使用的 backend。 - 原因：adaptive 为额外 runner 新建独立 backend，model_runner.attn_backend 仍指向活跃 backend；错挂 hook 会把 warmup 升级后的元数据冻结进 graph，首次重放时产生 Illegal Memory Access。
3. 引入进程级共享 capture stream，复用 graph pool scratch（runner_utils/pool.py + decode / prefill runner） - pool.py 新增 _CAPTURE_STREAM_NAME = "cuda_graph_capture" 与 get_or_create_global_graph_capture_stream()，内部通过 runtime_context.get_stream(name) 获取命名 stream lease，与现有 graph_memory_pool 同属 Resources 层。 - DecodeCudaGraphRunner.capture（非 pdmux 分支）与 PrefillCudaGraphRunner.capture 的 graph_capture() 显式传入该 stream。 - 原因：caching allocator 按 stream 分区 graph pool segments，共享一条 stream 后 scratch 只预留一次；capture 严格串行（Scheduler.init_all_cuda_graphs 逐个运行 target / draft worker，adaptive 控制器逐个构建 runtime state），单共享 stream 安全。 - 配套：runner_utils/__init__.py 增加导出；python/sglang/test/kits/attention_unittest/runner_modes/speculative_draft_runner.py 的 _single_rank_graph_capture shim 改为接受 stream=None 参数，保持与真实 graph_capture(stream=...) 两种调用形态兼容。
4. 新增 CPU 回归测试并演进 - 最初新增 test_graph_capture_resources.py 覆盖 pool 与 capture-stream lease，因依赖 CUDA 运行时被 review 指出（CI run #32329277658 失败），随后 patch torch.cuda.Stream 为 fake 使其在无 GPU 环境可跑。 - 最终提交用 test/registered/unit/model_executor/test_model_runner_decode_rows.py 替换 capture-resource 镜像测试，聚焦 adaptive logits sizing 回归：构造 _FakeModelRunner(initial_width=4, cuda_graph_bs=[4, 8, 12])、adaptive 配置 candidate_steps=[3, 5]、max_speculative_num_draft_tokens=6，断言 max_decode_logits_rows() == 72（bs 12 × 宽度 6），并注册到 CPU CI suite base-a-test-cpu。

关键文件：

- python/sglang/srt/model_executor/model_runner.py（模块 模型运行器；类别 source；类型 core-logic；符号 max_decode_logits_rows）：核心修复：重写 max_decode_logits_rows()，按 max_speculative_num_draft_tokens 与各 adaptive candidate 宽度（steps + 1）枚举取最大值，彻底解决启动时共享 logits buffer 定容不足的崩溃。
- python/sglang/srt/model_executor/runner_utils/pool.py（模块 资源池；类别 source；类型 core-logic；符号 get_or_create_global_graph_capture_stream）：新增进程级共享

graph capture stream（get_or_create_global_graph_capture_stream），利用缓存分配器按 stream 分区 graph pool segments 的特性，让所有 capture pass 复用同一份 scratch，是内存削减的关键机制。

- python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py（模块 解码图运行器；类别 source；类型 core-logic；符号 capture_one_shape, capture）：两处关键修复：capture_one_shape 的 warmup hook 改从 capture-local attn_backend 解析（修复首次重放 IMA），capture() 非 pdmux 分支改用共享 capture stream。
- python/sglang/srt/model_executor/runner/prefill_cuda_graph_runner.py（模块 预填充运行器；类别 source；类型 core-logic；符号 capture）：prefill capture 同样改用共享 capture stream，保证 prefill / decode 两个阶段共享同一进程级 stream 与 graph pool scratch，是内存复用策略的完整性配套。
- test/registered/unit/model_executor/test_model_runner_decode_rows.py（模块 单元测试；类别 test；类型 test-coverage；符号 TestModelRunnerDecodeRows, test_adaptive_sizing_covers_a_wider_candidate_width, _FakeModelRunner, _alignment_8_capture_bs）：新增 CPU 回归测试，锁定 adaptive 模式下共享 logits buffer 必须覆盖最宽 candidate 的 sizing 行为，作为启动崩溃修复的防护网。
- python/sglang/test/kits/attention_unittest/runner_modes/speculative_draft_runner.py（模块 测试工具；类别 test；类型 test-coverage；符号 _single_rank_graph_capture）：测试工具 shim _single_rank_graph_capture 适配 graph_capture(stream=...) 的新调用形态，保证 attention unittest 在共享 stream 下行为一致。
- python/sglang/srt/model_executor/runner_utils/__init__.py（模块 资源池；类别 source；类型 entrypoint）：导出新共享 capture stream 接口，保持 runner_utils 公共层的对外符号一致。

关键符号：max_decode_logits_rows, get_or_create_global_graph_capture_stream, DecodeCudaGraphRunner.capture_one_shape, DecodeCudaGraphRunner.capture, PrefillCudaGraphRunner.capture, _single_rank_graph_capture

关键源码片段

python/sglang/srt/model_executor/model_runner.py

核心修复：重写 max_decode_logits_rows()，按 max_speculative_num_draft_tokens 与各 adaptive candidate 宽度（steps + 1）枚举取最大值，彻底解决启动时共享 logits buffer 定容不足的崩溃。

```
def max_decode_logits_rows(self) -> int:
    """Rows the shared logits buffer needs."""
    # 关键：adaptive 模式可能被运行时解析关闭，因此以 runtime context
    #  （bags）中的有效值为准，而不是启动参数中记录的 CLI 输入。
    spec = get_spec()
    # 默认收集当前激活的 draft token 上限对应的宽度
    draft_token_counts = [max_speculative_num_draft_tokens()]
    if spec.speculative_adaptive:
        # adaptive 会为每个 candidate step 构建更宽的图，
        # 每步需要的行数为 steps + 1（验证 token + steps 个 draft token）
```

```
            draft_token_counts.extend(
                steps + 1
                for steps in resolve_candidate_steps_from_config(
                    spec.speculative_adaptive_config
                )
            )

        max_rows = 0
        for draft_tokens in draft_token_counts:
            num_tokens_per_req = self.decode_num_tokens_per_req(
                num_draft_tokens=draft_tokens
            )
            # 对该宽度可捕获的 batch size 求最大行数，取所有宽度中的最大值
            capture_bs, _ = get_batch_sizes_to_capture(self, num_tokens_per_req)
            max_rows = max(max_rows, max(capture_bs) * num_tokens_per_req)
        return max_rows
```

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

新增进程级共享 graph capture stream（get_or_create_global_graph_capture_stream），利用缓存分配器按 stream 分区 graph pool segments 的特性，让所有 capture pass 复用同一份 scratch，是内存削减的关键机制。

```
_CAPTURE_STREAM_NAME = "cuda_graph_capture"


def get_or_create_global_graph_capture_stream() -> Any:
    """Return the shared graph capture stream, creating it on first use so every
    capture pass reserves the pool's scratch once instead of per stream.

    CUDA only — the NPU / XPU / CPU graph runners keep their own streams.
    """
    # 缓存分配器按 stream 分区 graph pool 的 segments：
    # 若每次 capture 都用新 stream，MoE / DeepEP 等 scratch 会反复预留；
    # 所有 capture pass 共享一条 stream 后，后续 pass 可复用前次留下的
    # inactive segments，显著降低 adaptive speculative decoding 内存占用。
    return get_stream(_CAPTURE_STREAM_NAME)
```

python/sglang/srt/model_executor/runner/decode_cuda_graph_runner.py

两处关键修复：capture_one_shape 的 warmup hook 改从 capture-local attn_backend 解析（修复首次重放 IMA），capture() 非 pdmux 分支改用共享 capture stream。

```
# 关键：adaptive runner 绑定的是 capture 时新建的 attn_backend，
# 而 model_runner.attn_backend 仍指向当前活跃的 backend。
# 若从错误的 backend 查找 hook，会把 warmup 升级后的元数据
# （raw → full、FlashMLA scheduler meta）冻结进本次 graph，
# 首次重放时产生 Illegal Memory Access。
post_warmup_hook = getattr(
    attn_backend,
    "on_after_cuda_graph_warmup",
```

```
    None,
)

with (
    graph_capture(
        stream=get_or_create_global_graph_capture_stream()
    ) as graph_capture_context,
    profile_context as prof,
):
    # 所有 capture pass 共享同一条 stream,
    # 保证 graph pool 的 scratch 只预留一次
    self.stream = graph_capture_context.stream
    with self.backend.capture_session(self.stream):
        self._capture_one_stream()
```

评论区精华

review 的核心交锋集中在注释密度与测试可移植性两方面：

- hnyls2002 两次指出新增源码注释过多：在 pool.py 第 87 行评论 "Too many comments. Did you review every line of the comments manually?"，又在 decode_cuda_graph_runner.py 的单行改动处评论 "Only one line change; normally, our comments require only a few lines."。作者以 "clean redundant comments" 提交响应并精简注释。
- alphabetc1 在最初的 test_graph_capture_resources.py 上指出 "It looks like it still relies on the CUDA runtime so can't run on CPU-only test?"，并附上 CI 失败链接。Deleter-D 回复已将 torch.cuda.Stream patch 为 fake，断言只覆盖 pool.py 的 lease 逻辑，且本地 CUDA_VISIBLE_DEVICES=99 下 6 个用例全部通过、不初始化 CUDA context。
- 最终测试演进为 test_model_runner_decode_rows.py（提交 "replace the capture-resource mirrors with an adaptive logits sizing guard"），彻底移除对 CUDA 运行时的依赖，两位 reviewer 均 APPROVED。
- 新增注释密度过高 (style): 作者以 "clean redundant comments" 提交精简注释，最终合入版本注释收敛到关键缘由说明。
- 新测试依赖 CUDA 运行时，CPU-only CI 无法运行 (testing): 测试改为无 CUDA 依赖；后续提交进一步用 test_model_runner_decode_rows.py 替换 capture-resource 镜像测试，聚焦 adaptive sizing 回归并注册到 CPU suite。

风险与影响

- 风险：
  1. 共享 capture stream 是全局行为变化：所有 CUDA 平台的 decode / prefill graph capture（非 pdmux 分支）现在共用一个进程级 stream，非 adaptive 场景同样受影响。此设计依赖「capture 严格串行」这一不变式（Scheduler.init_all_cuda_graphs 逐个执行）；若未来引入并行 capture 或多 stream group 并发捕获，单一共享 stream 可能成为正确性隐患或性能瓶颈。

2. IMA 修复缺少端到端 GPU 回归：capture_one_shape 的 warmup hook 修复依赖 attention backend（FlashInfer / FlashMLA）的具体行为，现有 CPU 单元测试无法覆盖首次重放的 IMA 场景，属于高风险低覆盖区域。
3. max_decode_logits_rows() 与配置解析强耦合：新逻辑依赖 get_spec() 的运行时值和 resolve_candidate_steps_from_config 对 config 文件的解析；若解析结果与实际构建 runner 的宽度不一致（例如配置在 capture 前被运行时调整），buffer 仍可能偏小。好在测试覆盖了典型 geometry（bs 12、宽度 6、72 行）。
4. pdmux 多流分支未走共享 stream：decode_cuda_graph_runner.py 的 pdmux 分支仍使用 stream_groups 中 sg[1] 的 stream，该场景下内存复用收益会打折。
5. 进程级共享资源新增状态：命名 stream lease 与既有 graph_pool_borrow / disable_graph_pool_borrow 机制的交互未见冲突，但 Resources 层新增常驻 stream 的生命周期管理值得持续观察。

- 影响：
  - 用户影响：启用 --speculative-adaptive 的部署可正常启动（消除 #30549 崩溃与首次重放 IMA）；CUDA graph 显存显著下降（默认配置 14.4 GB→10.13 GB，作者自定义配置 44.3 GB→16 GB），释放的显存可直接转化为 KV cache 容量，提升并发与长上下文能力；三项精度基准（GPQA Diamond 88.13→88.04、AIME25 98.33→97.29、GSM8k 96.89→96.97）波动在误差范围内，无精度回退。
  - 系统影响：共享 capture stream 对全体 CUDA 用户生效，为进程级共享资源（graph memory pool、shared read event 之外）新增一名成员，后续 capture 流程演进必须维持「capture 串行」约束。
  - 团队影响：改动横跨 model_runner.py、两个 CUDA graph runner 与 runner_utils 公共层，speculative adaptive 相关的 capture 改动都需回归这几处；测试策略从「测试专用镜像」转向「复用生产路径 + patch 依赖」，更贴近真实调用链，也更容易被 CPU-only CI 覆盖。
  - 风险标记：核心路径变更：全部 CUDA graph capture 共用流，共享流依赖 capture 串行不变式，IMA 修复缺少端到端 GPU 回归测试，跨模块引入进程级共享资源

关联脉络

- PR #30554 [Spec] Size shared logits buffer for adaptive candidates: 同一缺陷（#30549）的独立修复，PR body 明确标注 "Same fix as #30554"；本 PR 采用更完整的遍历 candidate 宽度方案，两者共同决定共享 logits buffer 的定容逻辑。
- PR #35947 Publish gated DSV4 DFLASH-family target-prefill read completion: 同属 CUDA graph 捕获 / 回放基础设施演进：同样修改 prefill_cuda_graph_runner.py，并在 runner_utils 层引入进程级共享资源（shared read event 与本 PR 的共享 capture stream 并行），体现 runner_utils 从单进程内存池向多类共享资源演进的方向。