PR #26807 完整报告

sgl-project/sglang

Add the KV-canary plan JIT kernels

合并时间：2026-05-31 09:53

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26807

执行摘要

- 一句话：新增 KV-canary plan JIT 内核，编排 write/verify 计划
- 推荐动作：尽管 PR 已合并，但 Review 中提出的 CUDA Graph 内存损坏和输入验证问题尚未解决，建议后续提交及时修复。API 层增加边界检查和缓存机制（如类级持久化 scratch buffer 和 dummy LUT）将提升鲁棒性。对于阅读此 PR 的开发者，offsets_kernel.py 中 Triton 内核的块级 cumsum 实现和 SWA LUT 翻译逻辑值得学习。

功能与动机

KV-canary 系统需要通过 plan 内核来确定每个前向步骤中哪些 KV 槽需要写入或验证。PR body 指出该 op 是 write/verify 内核的顶级消费者，用于计算每请求的 canary 计划。

实现拆解

1. 入口 API(api.py)： launch_canary_plan_kernels 作为顶级函数，接收 VerifyPlan、WritePlan 以及请求元数据（req_pool_indices、prefix_lens 等），内部依次调用偏移量内核和条目内核填充输出计划。
2. 偏移量内核(offsets_kernel.py)： Triton JIT 内核 _plan_offsets_kernel 计算每个请求的验证偏移量、写入偏移量及种子槽索引，并累加总计数。 launch_plan_offsets_kernel 负责参数验证和内核启动，支持 SWA LUT 翻译。
3. 条目内核(entries_kernel.py)： 另一个 Triton 内核 _plan_entries_kernel 根据偏移量结果物化每个验证条目（slot index、前驱 slot、预期 token id）和写入元数据。
4. 工具函数(utils.py)： 提供 _resolve_swa_lut（处理 SWA 查找表）、输入验证辅助函数和两个 Triton JIT 函数 _compute_window_start 与 _swa_translate_tile，供偏移量内核使用。
5. Python 参考实现(plan_ref.py)： launch_canary_plan_kernels_torch_reference 在 CPU 上按相同的语义计算输出，用于差分测试验证字节等价。
6. 测试与基准： test_plan_hand.py 覆盖基本形状（单请求扩展 / 解码、多请求混合）、test_plan_fuzz.py 随机组合输入， bench_plan.py 在 NVIDIA GPU 上测量吞吐随变化（总 token 数、池容量）。所有测试均通过 Triton 结果与参考实现的字节级比较。

关键文件：

- python/sglang/jit_kernel/kv_canary/plan/offsets_kernel.py（模块 偏移内核；类别 source；类型 core-logic；符号 launch_plan_offsets_kernel, _validate_offsets_kernel_inputs, _plan_offsets_kernel, _exclusive_offsets_and_total）：

实现 plan 的核心 Triton 偏移量内核，计算每请求的 verify/write 偏移量和总计数，包含 SWA 支持。是 KV-canary 的关键计算模块。

- python/sglang/jit_kernel/kv_canary/plan/api.py（模块 计划入口；类别 source；类型 entrypoint；符号 launch_canary_plan_kernels）：顶层入口函数 launch_canary_plan_kernels 编排整个 plan 流程，协调偏移量内核与条目内核，对外提供统一接口。
- python/sglang/jit_kernel/kv_canary/plan/utils.py（模块 工具函数；类别 source；类型 utility；符号 _resolve_swa_lut, _require_dtype, _require_1d, _require_2d）：提供 SWA LUT 解析、输入验证辅助和 Triton JIT 工具函数，被偏移量内核和条目内核依赖，是计划模块的基础设施。
- python/sglang/jit_kernel/kv_canary/plan_ref.py（模块 参考实现；类别 source；类型 reference-implementation；符号 launch_canary_plan_kernels_torch_reference, _write_num_valid_and_enable, _swa_translate_slot, _materialize_verify_entries）：Python 参考实现，用于差分测试验证 Triton 内核的字节等价性，确保正确性。
- python/sglang/jit_kernel/benchmark/kv_canary/bench_plan.py（模块 性能基准；类别 source；类型 benchmark；符号 _TotalTokensBenchCase, _build_total_tokens_cases, _PoolCapacityBenchCase, _build_pool_capacity_cases）：性能基准测试，测量 plan 内核在不同负载（总 token 数、池容量、批次大小）下的吞吐，验证优化效果。
- python/sglang/jit_kernel/tests/kv_canary/test_plan_hand.py（模块 单元测试；类别 test；类型 test-coverage；符号 _tensor, _plan_pair, _alloc_for_inputs, _run_label）：手工测试用例，覆盖单请求扩展 / 解码、多请求混合等基本形状，与参考实现进行字节级比较。

关键符号：launch_canary_plan_kernels, launch_plan_offsets_kernel, _plan_offsets_kernel, launch_plan_entries_kernel, _resolve_swa_lut, _compute_window_start, _swa_translate_tile, launch_canary_plan_kernels_torch_reference, run_plan_diff, TestBasicShape, test_plan_fuzz_full_combo, fn

关键源码片段

python/sglang/jit_kernel/kv_canary/plan/offsets_kernel.py

实现 plan 的核心 Triton 偏移量内核，计算每请求的 verify/write 偏移量和总计数，包含 SWA 支持。是 KV-canary 的关键计算模块。

```
def launch_plan_offsets_kernel(
    *,
    req_pool_indices: torch.Tensor,
    prefix_lens: torch.Tensor,
    extend_seq_lens: torch.Tensor,
    req_to_token: torch.Tensor,
    full_to_swa_index_mapping: Optional[torch.Tensor],
    out_verify_offsets_scratch: torch.Tensor,
    out_write_offsets: torch.Tensor,
    out_write_seed_slot_indices: torch.Tensor,
    out_verify_num_valid: torch.Tensor,
```

```
    out_verify_enable: torch.Tensor,
    out_write_num_valid_reqs: torch.Tensor,
    swa_window_size: int,
    verify_capacity: int,
) -> None:
    # 提取批次大小与步长信息
    bs = int(req_pool_indices.shape[0])
    lut_tensor, lut_len, has_swa_lut = _resolve_swa_lut(
        full_to_swa_index_mapping, out_verify_offsets_scratch.device
    )
    req_to_token_stride0 = int(req_to_token.stride(0))
    write_offsets_len = int(out_write_offsets.shape[0])
    write_req_capacity = int(out_write_seed_slot_indices.shape[0])

    # 执行输入验证（形状、数据类型、连续性）
    _validate_offsets_kernel_inputs(
        req_pool_indices=req_pool_indices,
        prefix_lens=prefix_lens,
        extend_seq_lens=extend_seq_lens,
        req_to_token=req_to_token,
        lut_tensor=lut_tensor,
        out_verify_offsets_scratch=out_verify_offsets_scratch,
        out_write_offsets=out_write_offsets,
        out_write_seed_slot_indices=out_write_seed_slot_indices,
        out_verify_num_valid=out_verify_num_valid,
        out_verify_enable=out_verify_enable,
        out_write_num_valid_reqs=out_write_num_valid_reqs,
        bs=bs,
        req_to_token_stride0=req_to_token_stride0,
        lut_len=lut_len,
        has_swa_lut=has_swa_lut,
        write_offsets_len=write_offsets_len,
        write_req_capacity=write_req_capacity,
        verify_capacity=verify_capacity,
    )

    # 启动单块 Triton 内核，通过 constexpr 参数传递形状信息
    _plan_offsets_kernel[(1,)](
        req_pool_indices,
        prefix_lens,
        extend_seq_lens,
        req_to_token,
        lut_tensor,
        out_verify_offsets_scratch,
        out_write_offsets,
        out_write_seed_slot_indices,
        out_verify_num_valid,
        out_verify_enable,
        out_write_num_valid_reqs,
```

```
        bs,
        req_to_token_stride0,
        lut_len,
        BS_BLOCK=_PLAN_BS_BLOCK_SIZE, # 块内 cumsum 上限
        SWA_WINDOW=int(swa_window_size), # SWA 窗口大小，0 表示全池
        HAS_SWA_LUT=has_swa_lut, # 是否启用 LUT 翻译
        WRITE_OFFSETS_LEN=write_offsets_len,
        WRITE_REQ_CAPACITY=write_req_capacity,
        VERIFY_CAPACITY=verify_capacity,
        REQ_POOL_IDX_PADDING=REQ_POOL_IDX_PADDING,
        TOKEN_TO_KV_SLOT_PADDING=TOKEN_TO_KV_SLOT_PADDING,
    )
```

评论区精华

Review 由 gemini-code-assist[bot] 提出三个高优先级问题：

1. 输入验证缺失： api.py 中未检查 prefix_lens、extend_seq_lens 及 req_pool_indices 不超出 req_to_token 维度，可能导致越界内存访问。
2. CUDA Graph 内存损坏风险（scratch buffer）： api.py 每次调用通过 torch.empty 分配临时张量 verify_offsets_scratch；若被 CUDA Graph 捕获，此张量会在函数返回后被回收，造成图节点中的指针悬空。
3. CUDA Graph 内存损坏风险与分配开销（dummy LUT）： utils.py 的 _resolve_swa_lut 在非 SWA 模式下每次创建新零张量，同样存在图捕获后的指针回收风险，且引入不必要的分配开销。

- 输入验证缺失：prefix_lens 等可能越界 (correctness): 未收到作者回复，PR 已合并但问题未解决。
- CUDA Graph 内存损坏风险：动态分配 scratch buffer (performance): 未收到作者回复，PR 已合并但问题未解决。
- CUDA Graph 内存损坏风险与分配开销：dummy LUT 每次创建 (performance): 未收到作者回复，PR 已合并但问题未解决。

风险与影响

- 风险：主要风险集中在 CUDA Graph 兼容性：
  - api.py 每次调用动态分配 scratch buffer 和 utils.py 动态创建 dummy LUT 张量，若被 CUDA Graph 捕获会导致内存损坏（指针悬空）。当前实现未预留缓存机制，一旦在生产环境中启用 CUDA Graph（如 Eagle 推测解码），可能引发难以调试的随机错误。
  - 输入验证不足：缺少对 req_to_token、prefix_lens 等参数的边界检查，异常输入会造成内核越界访问，可能无声地破坏 KV 池。
  - 影响：对 KV-canary 子系统是核心组件，所有 write/verify 操作都依赖 plan 内核输出的偏移量与条目。对 模型推理路径无直接影响（仅当启用 KV-canary 时注入调用）。对 团队维护增加了 Triton JIT 内核和大量测试的维护成本，但片元级差分测试确保了正确性。对 性能影响由基准表明覆盖。

- 风险标记：CUDA Graph 兼容性风险，缺少输入验证，动态分配开销

关联脉络

- PR #26808 Add the KV-canary core: data layer, MHA KV-pool patcher, and per-forward runner: 同属 KV-canary 功能线，该 PR 添加核心数据层和运行时，本 PR 为其提供 plan 内核，两者紧密依赖。
- PR #26809 Add the KV-canary install API and forward-path wiring: 引入了 KV-canary 的安装 API 和前向路径接线，本 PR 的 plan 内核将被其消费。
- PR #26816 Add the KV-canary perturb framework for fault-injection self-tests: 添加扰动框架用于自测试，测试中会使用 plan 内核构建注入场景。
- PR #26820 Add a sliding-window-attention divergence reporter for the KV-canary: 添加 SWA divergence reporter，与本 PR 的 SWA 支持共享 LUT 处理逻辑。