PR #26818 完整报告

sgl-project/sglang

Add token-id verification to the KV-canary

合并时间：2026-05-31 09:58

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26818

执行摘要

- 一句话：为 KV-canary 添加 token-id 验证能力
- 推荐动作：值得精读，尤其是 scatter kernel 的 tile 大小选择（平衡寄存器预算）以及 manager 如何与 forward batch 协同工作。对于需要调试 KV 缓存一致性的开发者，此功能提供了有价值的手段。

功能与动机

KV-canary 用于检测 KV 缓存中的异常，但无法直接检测是否将错误 token 的 KV 存放到了某个 slot。本 PR 通过引入 token-id 验证，可以在每次 forward 时计算期望的 token 序列并与 canary 存储的 token 对比，从而增强检测能力。PR body 描述：'cross-check each canary slot's stored input token id against an independently-computed expected token ... catching the model placing the wrong token's KV in a slot.'

实现拆解

1. 新增 Triton scatter 内核(python/sglang/jit_kernel/kv_canary/scatter_req_token_ids.py)：实现 launch_scatter_req_token_ids_kernel 和 _scatter_req_token_ids_kernel，将扁平化的 token-id 数组按照请求索引散布到二维 pool 中。内核设计考虑了寄存器预算（SCATTER_BATCH_BLOCK=512）。
2. 新增期望 token 管理器(python/sglang/srt/kv_canary/req_to_expected_token_ids_manager.py)：compute_req_all_ids_info 从 Req 对象采集 origin_input_ids + output_ids，生成 pinned CPU 张量；populate_req_to_expected_token_ids 在 forward 阶段将数据复制到设备端并通过 scatter 内核写入 req_to_verify_expected_tokens pool。
3. 修改核心数据结构：
   - forward_batch_info.py 增加 req_all_ids_flat 和 req_all_ids_lens 字段，用于传递 token-id 快照。
   - state.py 将 pool 指针传递给 forward 计划。
   - plan_input.py 和 config.py 添加环境变量 SGLANG_KV_CANARY_ENABLE_VERIFY_TOKEN_ASSERT 的解析与校验开关。
4. 添加测试覆盖：包括内核单元测试（与 PyTorch 参考对比）、manager 单元测试（快照与散射逻辑）、端到端回归测试（模拟 chunked-prefill + EAGLE 场景）。

关键文件：

- python/sglang/jit_kernel/kv_canary/scatter_req_token_ids.py（模块 JIT 内核；类别 source；类型 core-logic；符号 launch_scatter_req_token_ids_kernel, _scatter_req_token_ids_kernel, scatter_req_token_ids_torch_reference）：核心 Triton 内核，实现 token 散布逻辑，是验证功能的基础。
- python/sglang/srt/kv_canary/req_to_expected_token_ids_manager.py（模块 请求校验；类别 source；类型 core-logic；符号 compute_req_all_ids_info, populate_req_to_expected_token_ids）：负责采集请求的 token 序列并调用 scatter 内核，串联整个验证流程。
- python/sglang/srt/kv_canary/state.py（模块 状态管理；类别 source；类型 core-logic）：修改了 canary state，新增对 verify_expected_tokens pool 的管理，连接 plan 和 forward。
- python/sglang/srt/model_executor/forward_batch_info.py（模块 执行器；类别 source；类型 data-contract）：新增 req_all_ids_flat 和 req_all_ids_lens 字段，保存每个 forward batch 的 token 快照，是 manager 数据的载体。
- python/sglang/jit_kernel/tests/kv_canary/test_scatter_req_token_ids.py（模块 JIT 内核；类别 test；类型 test-coverage；符号 _build_pool, _build_offsets, _build_flat, TestScatterReqTokenIds）：内核单元测试，覆盖基本散布、空批次、截断等场景，与 PyTorch 参考实现比对。

关键符号：launch_scatter_req_token_ids_kernel, _scatter_req_token_ids_kernel, scatter_req_token_ids_torch_reference, compute_req_all_ids_info, populate_req_to_expected_token_ids

关键源码片段

python/sglang/jit_kernel/kv_canary/scatter_req_token_ids.py

核心 Triton 内核，实现 token 散布逻辑，是验证功能的基础。

```
# python/sglang/jit_kernel/kv_canary/scatter_req_token_ids.py
# 关键入口函数：launch_scatter_req_token_ids_kernel
# 负责验证参数并启动 Triton 网格

def launch_scatter_req_token_ids_kernel(
    *,
    flat_in: torch.Tensor, # [total_tokens] int64 设备张量，扁平化的所有 token
    offsets: torch.Tensor, # [bs + 1] int64 设备张量，cumsum 边界
    req_pool_indices: torch.Tensor, # [bs] int64 设备张量，每个请求对应的 pool 行号
    pool_out: torch.Tensor, # [max_reqs, max_context_len] int32 设备张量，输出 pool
) -> None:
    # 严格的形状 / 类型校验，防止内核接收到非法输入
    if flat_in.dim() != 1:
        raise ValueError(...)
    if offsets.dim() != 1:
        raise ValueError(...)
    if req_pool_indices.dim() != 1:
        raise ValueError(...)
```

```
    if pool_out.dim() != 2:
        raise ValueError(...)
    # 类型校验：输入 int64，输出 int32（为节省显存）
    if flat_in.dtype != torch.int64:
        raise TypeError(...)
    if offsets.dtype != torch.int64:
        raise TypeError(...)
    if req_pool_indices.dtype != torch.int64:
        raise TypeError(...)
    if pool_out.dtype != torch.int32:
        raise TypeError(...)
    # 一致性校验：offsets 长度应为 bs+1
    bs = int(req_pool_indices.shape[0])
    if int(offsets.shape[0]) != bs + 1:
        raise ValueError(...)
    # BATCH_BLOCK 限制，确保 tile 在寄存器中
    if bs + 1 > _SCATTER_BATCH_BLOCK:
        raise ValueError(...)
    num_tokens = int(flat_in.shape[0])
    if num_tokens == 0:
        return # 空输入直接返回
    # 计算网格大小（token 维度）
    grid = (triton.cdiv(num_tokens, _SCATTER_TOKEN_BLOCK),)
    _scatter_req_token_ids_kernel[grid](
        flat_in, offsets, req_pool_indices, pool_out,
        num_tokens=num_tokens, num_batch=bs,
        pool_stride0=int(pool_out.stride(0)),
        pool_max_context_len=int(pool_out.shape[1]),
        TOKEN_BLOCK=_SCATTER_TOKEN_BLOCK,
        BATCH_BLOCK=_SCATTER_BATCH_BLOCK,
    )
```

python/sglang/srt/kv_canary/req_to_expected_token_ids_manager.py

负责采集请求的 token 序列并调用 scatter 内核，串联整个验证流程。

```
# python/sglang/srt/kv_canary/req_to_expected_token_ids_manager.py
# 核心函数：compute_req_all_ids_info 和 populate_req_to_expected_token_ids

def compute_req_all_ids_info(
    reqs: "list[Req]",
) -> tuple[torch.Tensor, torch.Tensor]:
    """采集每个请求的 origin_input_ids + output_ids，返回扁平化 pinned CPU 张量。"""
    # 将所有请求的输入输出 id 按顺序拼接
    parts = [arr for req in reqs for arr in (req.origin_input_ids, req.output_ids)]
    req_all_ids_flat = flatten_arrays_to_int64_tensor(
        parts, device=torch.device("cpu"), pin=True
    )
    # 记录每个请求的总 token 数（用于 scatter 时的长度边界）
    req_all_ids_lens = torch.tensor(
```

```
        [len(req.origin_input_ids) + len(req.output_ids) for req in reqs],
        dtype=torch.int64,
        pin_memory=True,
    )
    return req_all_ids_flat, req_all_ids_lens


def populate_req_to_expected_token_ids(
    *,
    forward_batch: "ForwardBatch",
    req_to_verify_expected_tokens: Optional[torch.Tensor],
) -> None:
    """将 forward batch 中的 token 快照散布到设备端 pool，供后续对比。"""
    req_all_ids_flat_cpu = forward_batch.req_all_ids_flat
    req_all_ids_lens_cpu = forward_batch.req_all_ids_lens
    # 当 snapshot 未就绪（如 CUDA Graph capture 阶段）或 pool 未创建时直接返回
    if req_all_ids_flat_cpu is None or req_all_ids_lens_cpu is None:
        return
    if req_to_verify_expected_tokens is None:
        return
    bs = int(forward_batch.req_pool_indices.shape[0])
    if bs == 0:
        return
    # 一致性校验：快照长度应与 batch size 一致
    if int(req_all_ids_lens_cpu.shape[0]) != bs:
        raise RuntimeError(...)
    # 计算 offsets（cumsum）并校验总 token 数
    offsets_cpu = torch.zeros(bs + 1, dtype=torch.int64, pin_memory=True)
    offsets_cpu[1:] = torch.cumsum(req_all_ids_lens_cpu, dim=0)
    total_tokens = int(offsets_cpu[bs].item())
    if total_tokens != int(req_all_ids_flat_cpu.shape[0]):
        raise RuntimeError(...)
    if total_tokens == 0:
        return
    # 异步拷贝到设备端 scatter 内核所在设备
    device = req_to_verify_expected_tokens.device
    req_all_ids_flat_dev = req_all_ids_flat_cpu.to(device, non_blocking=True)
    offsets_dev = offsets_cpu.to(device, non_blocking=True)
    req_pool_indices_dev = forward_batch.req_pool_indices.to(
        device=device, dtype=torch.int64
    )
    # 调用 Triton 内核完成散布
    launch_scatter_req_token_ids_kernel(
        flat_in=req_all_ids_flat_dev,
        offsets=offsets_dev,
        req_pool_indices=req_pool_indices_dev,
        pool_out=req_to_verify_expected_tokens,
    )
```

评论区精华

该 PR 无公开 review 评论，暂无争议点。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 新内核正确性风险：scatter 内核使用 Triton 编写，若边界条件处理不当（如空批次、超长序列截断）可能导致内存越界。单元测试覆盖了空批次、单请求、多请求以及截断场景，并提供了 PyTorch 参考实现比对。
  2. 性能开销：该功能默认关闭，启用后每次 forward 会增加一次 H2D 拷贝和 kernel 启动，但只对 canary 启用的请求生效，影响有限。
  3. 数据契约变更：forward_batch_info.py 新增字段，若其他模块未同步更新可能引发属性缺失。PR 通过类型检查和运行时断言防范。 - 影响：对用户：默认无感知，仅当设置环境变量 SGLANG_KV_CANARY_ENABLE_VERIFY_TOKEN_ASSERT 且 canary 模式为 LOG 时启用验证，增加调试能力。对系统：增强了 KV-canary 的检测能力，有助于早期发现模型将错误 token 的 KV 缓存存入 slot 的 bug。对团队：提供了一套可复用的 token 散布与验证基础设施。 - 风险标记：新 Triton 内核，核心路径变更，默认关闭，测试覆盖完善

关联脉络

- PR #26821 Add periodic KV-canary stats logging and kernel-run-counter health check: 同属 kv-canary 可观测性增强系列，共享部分基础设施。
- PR #26820 Add a sliding-window-attention divergence reporter for the KV-canary: 同系列 PR，增加 SWA divergence 报告。
- PR #26819 Add the KV-canary perturb modes and PD-disaggregation e2e tests: 同系列 PR，增加扰动模式和 PD 拆分端到端测试。
- PR #26329 Fix routed-experts device buffer overflow under DP attention: 本 PR 的端到端测试中涉及 revert PR #26329 的场景，用于回归验证。