PR #26810 完整报告

sgl-project/sglang

Add KV-canary SWA + DeepSeek-V4 pool support

合并时间：2026-05-31 09:55

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/26810

执行摘要

- 一句话：扩展 KV-canary 支持 SWA 和 DeepSeek-V4 KV 池
- 推荐动作：建议在合并前处理导入兼容性问题（使用 try-except 包装 DeepSeekV4TokenToKVPool 的导入）并修复拼写错误。该 PR 的适配器模式设计清晰，值得后续扩展时参考。

功能与动机

PR body 中提到：'Layer sliding-window-attention (SWA) and DeepSeek-V4 KV-pool support onto the MHA-only canary core' 目的是扩展 KV-canary 的支持范围，使其能监视 SWA 和 DeepSeek-V4 模型的 KV 缓存。

实现拆解

1. 新增适配器模块：在 swa.py 中实现 attach_swa，通过 _build_subpool_group 分别为 full 和 SWA 子池创建 CanaryBufferGroup；在 dsv4.py 中实现 attach_dsv4，仅处理 SWA 子池（其他子池未覆盖）。
2. 注册适配器：更新 api.py，导入新的适配器函数，并在 _POOL_ATTACHERS 字典中添加 SWAKVPool: attach_swa 和 DeepSeekV4TokenToKVPool: attach_dsv4。
3. 测试支撑：在 fixtures.py 中添加 FakeSWAPool 和 FakeSwaSubPool 模拟类，以及 make_swa_pool 工厂方法；在 consts.py 中定义 SWA 和 DSV4 的服务器参数和环境变量。
4. 单元测试：在 test_self_unit_pool_patcher.py 中新增 test_canary_buffer_group_allocate_full_and_swa 和 test_swa_attach_splices_full_into_contiguous_and_swa_into_state，验证缓冲区组分配和打补丁后的 buf_info。
5. 端到端测试：修改 e2e_base.py 和 mode_config.py，添加 'swa' 和 'dsv4' 模式；新增 test_self_e2e_baseline.py 中的 TestBaselineSwa 类和 test_self_e2e_baseline_dsv4.py。

关键文件：

- python/sglang/srt/kv_canary/pool_patcher/adapters/swa.py（模块 KV-canary；类别 source；类型 core-logic；符号 attach_swa, _build_subpool_group）：核心适配器，实现 SWA 池的 canary 缓冲区附加逻辑
- python/sglang/srt/kv_canary/pool_patcher/adapters/dsv4.py（模块 KV-canary；类别 source；类型 core-logic；符号 attach_dsv4）：核心适配器，实现 DeepSeek-V4 池的

canary 缓冲区附加（仅覆盖 SWA 子池）

- python/sglang/test/kv_canary/fixtures.py（模块 测试工具；类别 test；类型 test-coverage；符号 FakeSwaSubPool, FakeSWAPool, get_contiguous_buf_infos, get_state_buf_infos）：测试 fixtures，提供 FakeSWAPool 等模拟类，支持单元测试

关键符号：attach_swa, attach_dsv4, _build_subpool_group, make_swa_pool

关键源码片段

python/sglang/srt/kv_canary/pool_patcher/adapters/swa.py

核心适配器，实现 SWA 池的 canary 缓冲区附加逻辑

```
from __future__ import annotations
from typing import Optional
import torch
from sglang.srt.kv_canary.buffer_group import CanaryBufferGroup, PoolKind
from sglang.srt.kv_canary.pool_patcher.buf_info_splice import patch_buf_info_method
from sglang.srt.kv_canary.pool_patcher.buffer_alloc import alloc_canary_buf

def attach_swa(
    *,
    pool: object,
    device: torch.device,
    kv_token_id_vs_position_offset: int,
) -> tuple[CanaryBufferGroup, ...]:
    # 为 full 子池构建金丝雀缓冲区组（无 swa_index_lut）
    full_group = _build_subpool_group(
        sub_pool=pool.full_kv_pool,
        kind=PoolKind.FULL,
        device=device,
        swa_lut=None,
        kv_token_id_vs_position_offset=kv_token_id_vs_position_offset,
    )
    # 为 SWA 子池构建金丝雀缓冲区组（携带 swa_index_lut）
    swa_group = _build_subpool_group(
        sub_pool=pool.swa_kv_pool,
        kind=PoolKind.SWA,
        device=device,
        swa_lut=pool.full_to_swa_index_mapping,
        kv_token_id_vs_position_offset=kv_token_id_vs_position_offset,
    )
    # 打补丁 full 组的 buf_info 到 get_contiguous_buf_infos
    patch_buf_info_method(
        pool,
        method_name="get_contiguous_buf_infos",
        group=full_group,
        has_v_half=True,
        page_size=pool.page_size,
    )
```

```
    # 打补丁 SWA 组的 buf_info 到 get_state_buf_infos
    patch_buf_info_method(
        pool,
        method_name="get_state_buf_infos",
        group=swa_group,
        has_v_half=True,
        page_size=pool.page_size,
    )
    return (full_group, swa_group)

def _build_subpool_group(
    *,
    sub_pool: object,
    kind: PoolKind,
    device: torch.device,
    swa_lut: Optional[torch.Tensor],
    kv_token_id_vs_position_offset: int,
) -> CanaryBufferGroup:
    # 从子池的 k_buffer 第一维获取 slot 数量
    num_slots = int(sub_pool.k_buffer[0].shape[0])
    # 分配 4 个金丝雀缓冲区：k_head, k_tail, v_head, v_tail
    k_head = alloc_canary_buf(num_slots=num_slots, device=device)
    k_tail = alloc_canary_buf(num_slots=num_slots, device=device)
    v_head = alloc_canary_buf(num_slots=num_slots, device=device)
    v_tail = alloc_canary_buf(num_slots=num_slots, device=device)
    return CanaryBufferGroup(
        kind=kind,
        k_head=k_head,
        k_tail=k_tail,
        v_head=v_head,
        v_tail=v_tail,
        swa_index_lut=swa_lut,
        kv_token_id_vs_position_offset=kv_token_id_vs_position_offset,
    )
```

python/sglang/srt/kv_canary/pool_patcher/adapters/dsv4.py

核心适配器，实现 DeepSeek-V4 池的 canary 缓冲区附加（仅覆盖 SWA 子池）

```
from __future__ import annotations
import torch
from sglang.srt.kv_canary.buffer_group import CanaryBufferGroup, PoolKind
from sglang.srt.kv_canary.pool_patcher.buf_info_splice import patch_buf_info_method
from sglang.srt.kv_canary.pool_patcher.buffer_alloc import alloc_canary_buf

def attach_dsv4(
    *,
    pool: object,
    device: torch.device,
    kv_token_id_vs_position_offset: int,
```

```
) -> tuple[CanaryBufferGroup, ...]:
    """Attach canary buffers to a DeepSeekV4TokenToKVPool.
    TODO: only the swa_kv_pool sub-pool is wired; c4_kv_pool / c128_kv_pool /
    c4_indexer_kv_pool / compress state pools are left uncovered.
    """
    # 目前仅处理 SWA 子池
    sub_pool = pool.swa_kv_pool
    num_slots = int(sub_pool.size)
    # 分配 K 的金丝雀缓冲区（V 暂不分配，has_v_half=False）
    k_head = alloc_canary_buf(num_slots=num_slots, device=device)
    k_tail = alloc_canary_buf(num_slots=num_slots, device=device)
    # 创建 SWA 类型的金丝雀缓冲区组，无 V 缓冲区
    group = CanaryBufferGroup(
        kind=PoolKind.SWA,
        k_head=k_head,
        k_tail=k_tail,
        v_head=None,
        v_tail=None,
        swa_index_lut=pool.full_to_swa_index_mapping,
        kv_token_id_vs_position_offset=kv_token_id_vs_position_offset,
    )
    # 打补丁到 get_state_buf_infos
    patch_buf_info_method(
        pool,
        method_name="get_state_buf_infos",
        group=group,
        has_v_half=False,
        page_size=sub_pool.page_size,
    )
    return (group,)
```

python/sglang/test/kv_canary/fixtures.py

测试 fixtures，提供 FakeSWAPool 等模拟类，支持单元测试

```
@dataclass
class FakeSwaSubPool:
    k_buffer: List[torch.Tensor]
    v_buffer: List[torch.Tensor]

@dataclass
class FakeSWAPool:
    full_kv_pool: object
    swa_kv_pool: object
    full_to_swa_index_mapping: torch.Tensor
    page_size: int = 1

    def get_contiguous_buf_infos(self):
        # 返回 full_kv_pool 的缓冲区信息
        return _kv_buf_infos(
```

```
            k_buffer=self.full_kv_pool.k_buffer,
            v_buffer=self.full_kv_pool.v_buffer,
            page_size=self.page_size,
        )

    def get_state_buf_infos(self):
        # 返回 swa_kv_pool 的缓冲区信息
        return _kv_buf_infos(
            k_buffer=self.swa_kv_pool.k_buffer,
            v_buffer=self.swa_kv_pool.v_buffer,
            page_size=self.page_size,
        )

def _kv_buf_infos(*, k_buffer, v_buffer, page_size) -> tuple:
    # 通用函数：计算指针、nbytes、item_lens
    ptrs = [b.data_ptr() for b in k_buffer] + [b.data_ptr() for b in v_buffer]
    lens = [b.nbytes for b in k_buffer] + [b.nbytes for b in v_buffer]
    # 注意：使用 b.shape[1:].numel() 而非 b[0] 以支持空张量
    item_lens = [b.shape[1:].numel() * b.element_size() * page_size for b in k_buffer] + \
                [b.shape[1:].numel() * b.element_size() * page_size for b in v_buffer]
    return ptrs, lens, item_lens

def make_swa_pool(
    device: torch.device = DEFAULT_DEVICE,
    *,
    full_slots: int = 16,
    swa_slots: int = 8,
    dim: int = 8,
    layer_num: int = 1,
) -> FakeSWAPool:
    # 创建 full 和 swa 子池的模拟数据
    full = FakeSwaSubPool(
        k_buffer=[torch.zeros(full_slots, dim, dtype=torch.float16, device=device) for _ in
        range(layer_num)],
        v_buffer=[torch.zeros(full_slots, dim, dtype=torch.float16, device=device) for _ in
        range(layer_num)],
    )
    swa = FakeSwaSubPool(
        k_buffer=[torch.zeros(swa_slots, dim, dtype=torch.float16, device=device) for _ in
        range(layer_num)],
        v_buffer=[torch.zeros(swa_slots, dim, dtype=torch.float16, device=device) for _ in
        range(layer_num)],
    )
    # LUT: 前 swa_slots 映射，其余为 -1
    lut = torch.full((full_slots + 1,), -1, dtype=torch.int64, device=device)
    lut[:swa_slots] = torch.arange(swa_slots, dtype=torch.int64, device=device)
    return FakeSWAPool(full_kv_pool=full, swa_kv_pool=swa, full_to_swa_index_mapping=lut)
```

评论区精华

gemini-code-assist[bot] 提出了若干建议：

- 在 api.py 中应将 DeepSeekV4TokenToKVPool 的导入放在 try-except 中，以避免在不支持 DeepSeek 的环境中启动崩溃。
- mode_config.py 中的模型路径拼写错误（gemma-4-E2B-it 应改为 gemma-2-2b-it）。
- dsv4.py 中应添加对 pool.swa_kv_pool 为 None 的防御性检查。
- swa.py 中应检查 sub_pool 和 k_buffer 的有效性。
- fixtures.py 中应使用 b.shape[1:].numel() 代替 b[0] 以避免空张量时的 IndexError。
- e2e_base.py 中的注释引用了错误的模型名。 截止 PR 合并时，这些建议是否被采纳尚不明确（diff 中未体现修改）。
- 顶级导入 DeepSeekV4TokenToKVPool 可能导致启动失败 (security): PR 已合并但 diff 中未体现修改，状态未解决。
- 模型路径拼写错误 (correctness): 未修改，状态未解决。
- 缺少子池 None 检查 (correctness): 未修改，状态未解决。
- 空张量索引风险 (testing): 未修改，状态未解决。

风险与影响

- 风险：
  1. api.py 中顶级导入 DeepSeekV4TokenToKVPool 可能导致在不支持 DeepSeek 的平台上启动失败。
  2. dsv4.py 和 swa.py 缺少对子池（swa_kv_pool、k_buffer）为 None 或空时的防御性检查，可能引发 AttributeError 或 IndexError。
  3. 测试配置中的模型路径错误（gemma-4-E2B-it 不存在）会导致端到端测试失败。
  4. 测试 fixtures 中使用 b[0] 在 full_slots 或 swa_slots 为 0 时会触发 IndexError。 - 影响：对用户：若使用 SWA 或 DeepSeek-V4 模型并启用 KV-canary，将自动获得 KV 缓存监控，无需额外配置。对系统：仅在启动时注册适配器，运行时无性能影响。对团队：需维护适配器代码，确保与底层池接口同步。 - 风险标记：核心路径变更，环境兼容性，缺少防御检查，测试配置错误

关联脉络

- PR #26811 Add the KV-canary mock-model end-to-end test harness: 提供了 Mock 模型端到端测试框架，本 PR 的端到端测试依赖此框架。
- PR #26812 Add a periodic full-radix-tree KV-canary sweep: 增加定期全 radix 树扫描，与 SWA/DSV4 同属 KV-canary 的可观测性增强。
- PR #26813 Support EAGLE speculative decoding in the KV-canary: 扩展 KV-canary 到 EAGLE 推测解码，与本 PR 扩展池类型属于同一功能线。