PR #35154 完整报告

sgl-project/sglang

fix(unified-memory): four boot/correctness fixes on the hybrid model paths

合并时间：2026-09-01 06:08

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35154

执行摘要

- 一句话：统一内存混合模型路径 4 项启动与正确性修复
- 推荐动作：值得精读。这是 unified-memory 混合模型路径上线后第一批系统性缺陷修复，对以下读者尤其有价值：① 维护 mem_cache / 分配器相关代码的工程师——释放路径的 host 同步分析（标量 RHS H2D 阻塞、torch.unique D2H 计数）与 free_segment 步长推导是通用 GPU 内存管理经验；② 关注内核契约与 matcher 关系的读者——「matcher 强于内核真实契约」的判断方法可迁移到其他 JIT 内核；③ 测试设计爱好者——AST 源码扫描 + mock.patch 拦截 torch.unique + 功能测试三层护栏的组合非常值得借鉴。建议重点阅读 multi_ended_allocator.py 的 free/free_segment/_page_reps_pieces 与 swa_component.py 的 _page_pairs/_transfer_swa_pages，以及两个对应测试文件的 docstring（它们本身就是完整的设计文档）。

功能与动机

PR body 开宗明义："Four independent bugs on the hybrid model paths, each of which either prevents a model from booting or silently degrades behaviour." 四个 bug 分别是：① 既 mambaish 又 hybrid-SWA 的模型（Inkling 类）在 --attention-backend triton 下无法启动，TritonAttnBackend.__init__的中间分支调用 token_to_kv_pool.get_v_head_dim()，而 SWAKVPool 从未实现该方法，纯 hybrid-SWA 模型不是 mambaish 走不到该分支、mamba-hybrid 池自己有实现，所以一直未被发现；② conv-state matcher 默认 is_contiguous() 检查严格强于内核真实契约（内核体按 cache.stride(0)/stride(1) 索引，只需 channel 维连续），统一三池的 page-major 视图天然非连续；③ 统一池 free 路径两次 host 同步——t[idx] = -1 标量 RHS 触发 pageable H2D 阻塞拷贝，torch.unique 因数据依赖形状必须 D2H 读计数，测试 docstring 记录 gpt-oss/Qwen3.5 ps=256 上 77 次 _free_lazy 调用全部在同步；④ RecoverSWAWithLockedFull 假定静态分配器形状去索引统一池不存在的 full_to_swa_index_mapping 张量，而跳过恢复不可行：insert 路径的 new_prefix_len <= len(new_indices) 断言会把跳过变成正确性错误。

实现拆解

本 PR 按「一个 commit 一个修复」组织，5 个步骤拆解如下：

1. 启动修复（swa_memory_pool.py，+9 行）：新增 get_v_head_dim()，返回 full 侧 value buffer 在 full_kv_pool.start_layer 处的最后一维，与 HybridLinearKVPool 的既有实现对齐。读 start_layer 而非 layer 0 是正确性关键：第 0 层可能是 SWA 层（fixture 中

layers_mapping[0] 即 SWA），且 pipeline parallelism 下 start_layer > 0 也必须正确。UnifiedSWAKVPool 通过继承自动获得该方法，无需第二份实现。

2. 内核契约放松（inkling 目录 6 个 .cuh 文件）：7 处 TensorMatcher 的 .verify(cache) 全部追加 .with_strides({-1, -1, 1})——slot/window stride 通配、channel 维固定为 1，这是向量化 state 加载唯一依赖的契约。涉及 update_sconv_cache.cuh、causal_conv1d.cuh、draft_extend_sconv.cuh、fused_decode_update.cuh、gather_scatter_sconv.cuh、inkling_ar_fused_decode.cuh。配套测试 test_inkling_sconv_strided_conv_state.py 用两层护栏：CPU 可跑的源码扫描保证任意站点回归即失败（含已知站点数完整性检查、channel 维禁止全通配检查），CUDA + JIT 功能测试用 page-major strided 视图驱动真实 update_sconv_cache 内核并断言与 contiguous 克隆 bit 级一致。
3. 释放路径去 host 同步（multi_ended_allocator.py，+181/-40，本 PR 最大改动）：free() 新增 pages 关键字参数携带位置推导好的 page id 以跳过 torch.unique；新增 _page_reps_pieces() 与 free_segment()，按 start_pos % page_size 推导代表元切片（头部不满页取 free_index[:1]，其余按 stride 步进）；新增 free_page_reps_group 缓冲，free-group 内缓存 page 代表元而非原始 token，避免 free_group_end 拼接时破坏每段形状；墓碑写入在 free、_free_lazy、_commit_move_batch 三处统一改为 index_fill(0, idx, -1)；删除 SGLANG_SORT_FREE_LIST_AFTER_MERGE 开关（environ.py 同步转入 _DeprecatedEnv）。base 与 Mamba/SWA 两个复合分配器全部覆写 free_segment，测试逐一验证三个类未继承 base 的丢弃 start_pos 版本。
4. SWA locked-full 恢复（swa_component.py，+85/-1）：apply_component_action 的 RecoverSWAWithLockedFull 分支先经 unified_allocator() 判断分配器类型。unified 分支用 _page_pairs 按「首次出现」掩码配对两个区间的 page（刻意不用 torch.unique，它按 id 值排序会配错页），_transfer_swa_pages 做 v2p rebind——节点 virtual page 绑定 incoming 的物理页、incoming 表项 index_fill 置 -1、清 inverse history，再经复合池 free() 释放时其 swa_v2p_pages > 0 过滤只退还 full 侧；并断言 incoming 页全部存活（physical > 0），防止把 padding sink 交给节点。静态分支保持原配方，但映射写入统一走 set_full_to_swa_mapping / clear_full_to_swa_mapping API 而非直接索引张量。
5. 配套清理：flashattention_backend.py 与 aiter_backend.py 删除 _apply_cuda_graph_metadata 两个从未被读取的参数（seq_lens_sum、encoder_lens）；test_unified_radix_cache_unittest.py 补 3 行。新增 5 个测试文件全部注册 CPU CI，其中 test_swa_locked_full_recover_unified.py 用真实 UnifiedSWATokenToKVPoolAllocator + stub TreeCore 做端到端断言，并用 _StaticAllocRecorder 验证静态池路径未被 unified 分支劫持。

关键文件：

- python/sglang/srt/mem_cache/multi_ended_allocator.py（模块 分配器；类别 source；类型 core-logic；符号 free, _free_lazy, _page_reps_pieces, free_segment）：统一池释放路径核心：free/_free_lazy/free_segment/_page_reps_pieces/free_group 重构，移除 torch.unique 与标量墓碑两次 host 同步，是调度线程停顿问题的根治点。
- python/sglang/srt/mem_cache/unified_cache/components/swa_component.py（模块 SWA 组件；类别 source；类型 core-logic；符号 _unified_allocator, _page_pairs,

_transfer_swa_pages, apply_component_action）：SWA locked-full 恢复在统一池下的崩溃修复：以 v2p 页属主重绑定替代静态映射张量配方，并用首次出现掩码保证配对位置语义。

- python/sglang/srt/mem_cache/swa_memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 get_v_head_dim）：启动崩溃修复点：为 SWAKVPool 增加 get_v_head_dim，从 full 子池的 start_layer 读取维度，UnifiedSWAKVPool 继承复用。
- test/registered/unit/mem_cache/test_unified_free_no_host_sync.py（模块 缓存测试；类别 test；类型 test-coverage；符号 _paged_allocator, _scalar_index_assignments, _is_scalar_literal, TestTombstonesDoNotCrossTheBus）：释放路径去同步的回归护栏：AST 源码扫描拦截标量墓碑赋值，mock.patch 拦截 torch.unique，验证三个分配器类均覆写 free_segment 且 group 缓冲的是代表元。
- test/registered/unit/mem_cache/test_swa_locked_full_recover_unified.py（模块 缓存测试；类别 test；类型 test-coverage；符号 _build_swa_composite, _StubTreeCore, _Probe, _StaticAllocRecorder）：SWA locked-full 恢复的端到端回归测试：真实 UnifiedSWATokenToKVPoolAllocator + stub TreeCore，断言属主转移不改容量、回收值指向存活页、静态池路径未被劫持。
- test/registered/unit/mem_cache/test_inkling_sconv_strided_conv_state.py（模块 缓存测试；类别 test；类型 test-coverage；符号 _cache_matcher_lines, TestConvStateMatchersAcceptStrided, TestUpdateSconvCacheStridedFunctional）：内核契约放松的回归护栏：源码扫描保证 7 处 conv-state matcher 全部携带 stride 放松且 channel 维不失控，CUDA 功能测试验证 strided 视图与 contiguous 结果 bit 级一致。
- test/registered/unit/mem_cache/test_swa_pool_v_head_dim.py（模块 缓存测试；类别 test；类型 test-coverage；符号 _swa_pool, TestSWAPoolVHeadDim）：get_v_head_dim 修复的回归测试：钉住返回值来自 full 子池 start_layer 而非 layer 0、UnifiedSWAKVPool 继承同一实现、签名与 HybridLinearKVPool 前例一致。
- python/sglang/kernels/jit/csrc/inkling/update_sconv_cache.cuh（模块 JIT 内核；类别 other；类型 core-logic）：7 处 TensorMatcher 放松的代表文件（其余 5 个 .cuh 为同型一行改动）：.with_strides({-1, -1, 1}) 使 page-major 非连续 conv-state 视图通过校验，与已 stride-aware 的内核体契约对齐。
- python/sglang/srt/environ.py（模块 环境配置；类别 source；类型 configuration）：删除不再可达的 SGLANG_SORT_FREE_LIST_AFTER_MERGE 配置键并转入 _DeprecatedEnv，标记统一池释放列表默认不排序的设计决策。
- python/sglang/srt/layers/attention/flashattention_backend.py（模块 注意力后端；类别 source；类型 refactor；符号 _apply_cuda_graph_metadata, _init_full_cg_decode_metadata）：维护者清理 commit：删除 _apply_cuda_graph_metadata 未被读取的 seq_lens_sum 参数及调用处计算（aiter_backend.py 同类删除 encoder_lens），降低 CUDA graph 元数据签名噪音。

关键符号：get_v_head_dim, free, _free_lazy, free_segment, _page_reps_pieces, free_group_begin, free_group_end, _release_phys_pages_batch, clear, _unified_allocator, _page_pairs, _transfer_swa_pages, apply_component_action

关键源码片段

python/sglang/srt/mem_cache/multi_ended_allocator.py

统一池释放路径核心：free/_free_lazy/free_segment/_page_reps_pieces/free_group 重构，移除 torch.unique 与标量墓碑两次 host 同步，是调度线程停顿问题的根治点。

```
def free(self, free_index: torch.Tensor, *, _pages: Optional[torch.Tensor] = None) -> None:
    """释放 virtual TOKEN id，回收 virtual PAGE id 并解除 v2p / p2v 映射。

    `_pages` 携带 `free_segment` 已按 `start_pos` 步长推导好的 page id，
    提供时跳过依赖数据形状的 `torch.unique` 去重；free-group 路径有独立
    的代表元缓冲，不接收该参数。
    """
    with record_function("MultiEndedAlloc.free"):
        if free_index is None or free_index.numel() == 0:
            return
        if self.free_group is not None:
            self.free_group.append(self._copy_for_free_group(free_index))
            return
        if self.lazy_compaction:
            self._free_lazy(free_index, pages=_pages)
            return
        # --- EAGER 路径 ---
        if self.forward_stream is not None:
            torch.cuda.current_stream().wait_stream(self.forward_stream)
        with record_function("MultiEndedAlloc.free.v2p_lookup"):
            free_v_pages = (
                _pages
                if _pages is not None
                else torch.unique(free_index.detach().to(torch.int64) // self.page_size)
            )
            freed_p_pages = self.virtual_to_physical[free_v_pages]
        if bool((freed_p_pages < 0).any().item()):
            self._raise_stale_slot_assertion(free_v=free_v_pages, freed_p=freed_p_pages)
        # 墓碑写入用 `index_fill_` 而不是 `t[idx] = -1`：标量 RHS 会让 torch 在
        # CPU 物化 -1 张量并走 pageable H2D 拷贝，而 pageable H2D 拷贝是 host
        # 阻塞的 —— 调度线程会停在在飞 forward 后面（8192-token prefill 场景
        # 约 16 ms/ 次）。`index_fill_` 走 ATen Scalar 重载：单个设备内核，无
        # host 同步。
        self.virtual_to_physical.index_fill_(0, free_v_pages, -1)
        if self.is_id_owner:
            self.free_virtual_ids = torch.cat([self.free_virtual_ids, free_v_pages])
        self._compact_pending(freed_p_pages)


def _page_reps_pieces(
    self, free_index: torch.Tensor, start_pos: int
) -> Tuple[torch.Tensor, ...]:
    """取一条 kv-row 段内代表各 page 的 token 切片。
```

```
    与 `PagedTokenToKVPoolAllocator.free_segment` 同理：一个 page 的 token
    在 kv row 中连续存放，host 侧已知 `start_pos` 时代表元就是 stride 切片，
    无需 `torch.unique` —— 后者输出形状依赖数据，必须先 D2H 读计数再同步。
    对任意段形状精确：不满页的头部是 `[:1]` 项，不满页的尾部是最后一个
    stride 步。
    """
    ps = self.page_size
    offset = start_pos % ps
    if offset == 0:
        return (free_index[::ps],)
    return (free_index[:1], free_index[ps - offset :: ps])


def free_segment(self, free_index: torch.Tensor, *, start_pos: int) -> None:
    """`free()` 的固定形状版本，见 `_page_reps_pieces`；每页在每个 group 内
    只能由一个调用释放。
    """
    if free_index is None or free_index.numel() == 0:
        return
    if self.page_size == 1:
        self.free(free_index) # token == page，无需去重，普通路径已精确
        return
    pieces = self._page_reps_pieces(free_index.detach().to(torch.int64), start_pos)
    if self.free_page_reps_group is None:
        reps = pieces[0] if len(pieces) == 1 else torch.cat(pieces)
        self.free(reps, _pages=reps // self.page_size)
    else:
        # free-group 内缓冲 PAGE 代表元而非原始 token：`torch.cat` 拼接原始
        # token 会破坏每段形状，group end 时只能退回带同步的去重路径
        self.free_page_reps_group.extend(pieces)
```

python/sglang/srt/mem_cache/unified_cache/components/swa_component.py

SWA locked-full 恢复在统一池下的崩溃修复：以 v2p 页属主重绑定替代静态映射张量配方，并用首次出现掩码保证配对位置语义。

```
def _page_pairs(
    self, full_value: torch.Tensor, incoming_full_value: torch.Tensor
) -> tuple[torch.Tensor, torch.Tensor]:
    """取两个 token 区间中指向同一逻辑 token 的 page id 对。

    用「首次出现」掩码去重而不是 `torch.unique`：unique 按 id 值排序，
    而分配器发放的 virtual id 并无顺序，排序会把两个区间里互不相关的
    page 配成一对 —— 静默写错 KV。单个共享掩码保证配对是位置性的，
    因而是逻辑性的。
    """
    page_size = self.tree_core.page_size
    kept = full_value.detach().to(torch.int64) // page_size
    incoming = incoming_full_value.detach().to(torch.int64) // page_size
```

```
    assert kept.numel() == incoming.numel(), (
        f"locked-full recovery 需要 1:1 的 token 对应关系，"
        f"实际 kept={kept.numel()} vs incoming={incoming.numel()}"
    )
    # 相邻不等检测每个区间的 page 起点，两个区间必须按同一偏移断页
    starts = torch.ones_like(kept, dtype=torch.bool)
    starts[1:] = kept[1:] != kept[:-1]
    incoming_starts = torch.ones_like(incoming, dtype=torch.bool)
    incoming_starts[1:] = incoming[1:] != incoming[:-1]
    assert torch.equal(starts, incoming_starts), (
        "两个区间断页偏移不同，page 粒度的属主转移无法表达该 token 映射"
    )
    return kept[starts], incoming[starts]


def _transfer_swa_pages(
    self,
    allocator,
    full_value: torch.Tensor,
    incoming_full_value: torch.Tensor,
) -> None:
    """把 swa page 属主从 incoming id 转移到节点自己的 id 上。

    静态池配方通过 `full_to_swa_index_mapping` 重指向；unified 池下 swa
    子池的 v2p 表本身就是这份映射，所以同样一次移动就是一次 rebind：
    节点 virtual page 绑定 incoming 的物理页，再把 incoming 表项置为
    tombstone。不分配也不释放任何 page，容量不变，只有属主变化。
    """
    swa = allocator.swa_attn_allocator
    kept_pages, incoming_pages = self._page_pairs(full_value, incoming_full_value)
    physical = swa.virtual_to_physical[incoming_pages]
    # `> 0` 严格判断：-1 = 已 tombstone，0 = padding sink。incoming id 刚被
    # 在飞请求分配，每页必须存活；违反即相当于把 sink 交给节点去服务
    # 全零数据，值得硬失败而不是静默损坏。
    assert bool((physical > 0).all()), (
        f"incoming swa pages 必须全部存活，实际得到 {physical.tolist()}"
    )
    swa.bind(kept_pages, physical)
    swa.virtual_to_physical.index_fill_(0, incoming_pages, -1)
    swa.clear_inverse_history()

if isinstance(action, RecoverSWAWithLockedFull):
    # 保留锁定的 full；把 incoming id 的 swa page 交给节点，只释放
    # incoming 的 full 侧，再把 swa 值写回节点。
    unified = self._unified_allocator()
    if unified is not None:
        # unified 复合池没有 `full_to_swa_index_mapping`：swa 子池的 v2p
        # 就是映射。先 rebind 页属主，再经复合池 free —— 其
        # `swa_v2p_pages > 0` 过滤会跳过刚被 tombstone 的 swa 侧，只释放
```

```
            # full 侧。
            self._transfer_swa_pages(
                unified, action.kept_full, action.incoming_full
            )
            unified.free(action.incoming_full)
            self.tree_core.set_component_device_value(
                action.node_id,
                self.component_type,
                self._translate_full_to_swa(action.kept_full),
            )
            return
        # 静态池保持原配方，映射写入一律走分配器 API 而非直接索引张量
        swa_value = self._translate_full_to_swa(action.incoming_full)
        alloc.set_full_to_swa_mapping(action.kept_full, swa_value)
        alloc.clear_full_to_swa_mapping(action.incoming_full)
        alloc.free_full(action.incoming_full)
        self.tree_core.set_component_device_value(
            action.node_id, self.component_type, swa_value
        )
        return
```

python/sglang/srt/mem_cache/swa_memory_pool.py

启动崩溃修复点：为 SWAKVPool 增加 get_v_head_dim，从 full 子池的 start_layer 读取维度，UnifiedSWAKVPool 继承复用。

```
    def get_v_head_dim(self):
        # 返回 FULL 侧的维度，与 `HybridLinearKVPool.get_v_head_dim()` 对齐：
        # 向池子询问 v_head_dim 的调用方要的是 full-attention 几何
        # （`TritonAttnBackend.__init__` 的 mambaish 分支）。
        # 读 `start_layer` 而不是 layer 0：第 0 层不一定是 full-attention 层，
        # 且 pipeline parallelism 下 start_layer > 0 也必须正确。
        return self.full_kv_pool.get_value_buffer(self.full_kv_pool.start_layer).shape[-1]
```

评论区精华

该 PR 全程没有 inline review 评论（comments_count = 0，review_comments_count = 0），由 ch-wan 合入，并在作者 4 个修复 commit 之外追加了 3 个清理 commit：移除 FlashAttention/Aiter 后端 _apply_cuda_graph_metadata 的两个无效参数、压缩两处过长注释、删除 inkling matcher 上 7 份重复注释。真正有分量的设计论证沉淀在 PR body 与测试 docstring 中，几个关键判断：

- 「The matcher was strictly stronger than the kernels' real contract」——matcher 默认 is_contiguous() 比内核体（按 stride 索引）的实际约束更严，这是内核契约分析的标准范例。
- 「torch.unique sorts by id value, and allocation hands out virtual ids in no particular order, so sorting would pair page k of one range with an unrelated page of the other — silent wrong-KV」——这是 _page_pairs 必须用首次出现掩码而非 torch.unique 的根本原因。

- 「a pageable H2D copy BLOCKS the host until the stream drains」——标量 RHS 赋值与 index_fill_ 的设备行为差异是释放路径去同步的核心洞察，约 16 ms/ 次。
- 测试自检：「-1 parses as UnaryOp(USub, Constant(1)), NOT Constant」——AST 扫描测试专门钉住了负数字面量的解析坑，避免扫描静默漏检所有墓碑。
- 取舍原则：「Plain free() keeps torch.unique — without a position there is nothing to derive from」，即无位置信息时正确性优先于速度。
- 暂无高价值评论线程

风险与影响

- 风险：
  1. 释放路径重构（multi_ended_allocator.py）：free_segment 的步长推导依赖调用方保证 segment 来自同一 kv-row 且 start_pos 准确。若未来新增分配器子类未覆写 free_segment，base 版本会丢弃 start_pos 静默退回 torch.unique 同步路径——测试 TestEveryUnifiedAllocatorOverridesFreeSegment 已用完整性守卫覆盖现有三个类，但新类仍可能遗漏。另外普通 free() 与 free-group 内的无位置 free 仍保留 torch.unique，属有意的正确性优先。
  2. SWA 恢复重绑定（swa_component.py）：_page_pairs 要求两个区间按同一偏移断页，否则断言失败（不会静默错配）；physical > 0 断言防止把 padding sink 交给节点服务全零数据。风险在于断言失败即调度器报错，依赖 _page_pairs 的前置条件（incoming 与 kept 1:1 token 对应）在调用链上始终成立。
  3. 内核契约放松（inkling .cuh）：slot/window stride 通配后，若未来某个内核体实际依赖完整连续性（例如引入按 slot 连续假设的向量化），将静默接收 strided 数据。测试通过固定 {-1, -1, 1} 并禁止 {-1, -1, -1} 来守住 channel 维契约，但功能测试仅覆盖 update_sconv_cache 一个内核的真实执行。
  4. 环境变量删除：SGLANG_SORT_FREE_LIST_AFTER_MERGE 被移除并转入 _DeprecatedEnv，外部显式设置该变量的部署会收到废弃告警，但不会硬失败。
  5. 无效参数删除（flashattention/aiter）：无行为影响，仅签名清理；PR 声明 Musa FA3 子类未覆盖这两个方法，风险极低，但未附测试证明。 - 影响：影响范围集中在 unified-memory 子系统与混合模型（mamba/SWA/diffusion 系）路径：① Inkling 类（mambaish + hybrid-SWA 且 full/SWA value head dim 相等）模型现在可在 triton 后端正常启动，静态池与统一池同时受益；② 统一池释放路径不再发生调度线程阻塞，实测 gpt-oss、Qwen3.5 ps=256 上原本 77/77 次 _free_lazy 同步降至 0，长 prefill 场景每次释放可省约 16 ms；③ SWA locked-full 恢复在 --enable-unified-memory 下不再崩溃调度器，radix 缓存插入 / 匹配的一致性断言（new_prefix_len <= len(new_indices)）得以维持；④ 对静态池用户，行为保持不变（映射 API 路由相同、性能开关默认关闭）。PR body 给出的 GSM8K n=200 对比：各模型 mean Δ 在 -0.31 ~ +0.06 pt，TPOT 相对静态池 -0.7% ~ +2.4%，精度无回退、性能基本持平。团队层面，5 个新测试文件 +929 行为该子系统建立了可持续回归护栏，特别是 AST 源码扫描测试为同类「防止再引入 host 同步 / 契约回退」的问题提供了可复用的测试方法论。 - 风险标记：统一池释放路径重构，调度关键路径，内核契约放松，位置语义前置条件，AST 扫描回归护栏

关联脉络

- PR #35177 feat(unified-memory): three sub-pools for mamba + hybrid-SWA models: 同一功能线：三子池把 mamba + SWA 混合模型引入统一池，本 PR 正是该功能落地后暴露的启动、释放与恢复缺陷的定点修复。
- PR #35158 feat(unified-memory): byte-budget sizing, feasibility floor, and a conservation verifier: 同批 unified-memory 演进，改动 multi_ended_allocator / unified_memory_pool / kv_cache_configurator，与本 PR 的释放路径重构直接相邻。
- PR #34613 feat(unified-memory): read unified pool from attention backends fa3/flashinfer/trtllm_mha/flashmla: 统一池读路径迁移至注意力后端，conv-state 的 page-major 视图与注意力后端 v_head_dim 分支问题同属此脉络的上游。
- PR #37167 [mem_cache] Make release, row-reuse asserts, and presence checks read the KV record: mem_cache 释放与一致性系列的延续，与本 PR 的释放路径与 v2p 映射正确性关注点一脉相承。