PR #25098 完整报告

sgl-project/sglang

perf: migrate Req token-id storage to array.array('q') in Scheduler

合并时间：2026-05-23 01:51

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25098

执行摘要

- 一句话：Req token-id 存储迁移至 array.array('q') 优化长 prompt 性能
- 推荐动作：建议 PR 评审者仔细阅读，尤其是 flatten_arrays_to_int64_tensor 的实现和 Req 字段迁移的处理方式，可为类似数据类型迁移提供参考。

功能与动机

PR body 指出，切换至 PyArray 可显著提升长 prompt 场景效率（主要来自更高效的 Tensor 构建），同时为未来将 Radix Cache 迁移至 Rust 后降低 PyO3 通信开销做准备。当前 Scheduler 和 Radix Cache 使用 PyList 通信，迁移到 Rust 时列表转 Vec 的 PyO3 开销会成为瓶颈，而 array 的连续内存布局可降至微秒级。

实现拆解

1. 核心数据结构迁移：在 schedule_batch.py 中将 Req 的 origin_input_ids、output_ids、fill_ids 类型从 list[int] 改为 array('q')，调整相关属性（如 output_ids_through_stop）并移除过渡用的 setter。
2. 工具函数提取：在 utils/common.py 中新增 flatten_arrays_to_int64_tensor，通过 np.frombuffer 零拷贝视图避免逐元素装箱，高效将 array 列表转为 pinned CUDA 张量。
3. 模型接口适配：更新 moss_vl.py、whisper.py、llavavid.py、llava.py、mllama.py 等多个多模态模型的 pad_input_ids 方法，使其接受并返回 array('q') 以保持数据类型一致。
4. 下游调用方适配：调整 output_streamer.py、disaggregation 模块、RadixKey 等中的写操作和类型注解，确保所有赋值都使用 array('q') 而非 list，同时修复 prepare_encoder_info_extend 使用新工具函数。
5. 测试与基准：新增 bench_token_storage.py 基准脚本对比 list 与 array 性能；新增 test_common.py 单元测试覆盖 flatten_arrays_to_int64_tensor 的 CPU/CUDA 路径；调整现有 radix cache 测试以适应新的 RadixKey 类型要求。

关键文件：

- python/sglang/srt/managers/schedule_batch.py（模块 调度器；类别 source；类型 core-logic；符号 output_ids_through_stop, prepare_encoder_info_extend）：核心变更文件：将 Req 的 token-id 存储从 list[int] 迁移至 array('q')，并调整所有相关属性和方法，是本次 PR 的主要改动点。

- python/sglang/srt/utils/common.py（模块 工具函数；类别 source；类型 core-logic；符号 flatten_arrays_to_int64_tensor）：新增核心工具函数 flatten_arrays_to_int64_tensor，用于高效将 array 列表转为张量
- benchmark/scheduler/bench_token_storage.py（模块 基准测试；类别 source；类型 benchmark；符号 _ingest_list, _ingest_pyarray, _empty_list, _empty_pyarray）：新增基准测试脚本，模拟请求生命周期对比 list 与 array 各阶段性能
- test/registered/unit/utils/test_common.py（模块 测试；类别 test；类型 test-coverage；符号 TestFlattenArraysToInt64Tensor, _check, test_single_part, test_multiple_parts）：新增 flatten_arrays_to_int64_tensor 的单元测试，覆盖 CPU/CUDA 的多种组合
- python/sglang/srt/models/moss_vl.py（模块 模型适配；类别 source；类型 data-contract；符号 _build_encoder_prefix_pad_ids, pad_input_ids）：代表多模态模型适配，其 pad_input_ids 方法从 List[int] 迁移至 array('q')

关键符号：flatten_arrays_to_int64_tensor, output_ids_through_stop, pad_input_ids, _build_encoder_prefix_pad_ids

关键源码片段

python/sglang/srt/managers/schedule_batch.py

核心变更文件：将 Req 的 token-id 存储从 list[int] 迁移至 array('q')，并调整所有相关属性和方法，是本次 PR 的主要改动点。

```
# python/sglang/srt/managers/schedule_batch.py（片段）

class Req(ReqDllmMixin):
    def __init__(
        self,
        rid: str,
        origin_input_text: str,
        origin_input_ids: array[int], # 类型从 List[int] 迁移为 array('q')
        sampling_params: SamplingParams,
        return_logprob: bool = False,
        top_logprobs_num: int = 0,
        dllm_config: Optional[DllmConfig] = None,
        token_ids_logprob: List[int] = None,
        stream: bool = False,
        origin_input_ids_unpadded: Optional[array[int]] = None, # 同样迁移
        lora_id: Optional[str] = None,
        input_embeds: Optional[List[List[float]]] = None,
        positional_embed_overrides: Optional[PositionalEmbeds] = None,
        token_type_ids: List[int] = None,
        session: Optional[Session] = None,
        custom_logit_processor: Optional[str] = None,
        require_reasoning: bool = False,
        return_hidden_states: bool = False,
        return_routed_experts: bool = False,
        routed_experts_start_len: int = 0,
```

```
        return_indexer_topk: bool = False,
        eos_token_ids: Optional[Set[int]] = None,
        bootstrap_host: Optional[str] = None,
        bootstrap_port: Optional[int] = None,
        ...
    ):
        # ... 其他初始化
        self.origin_input_ids = origin_input_ids
        # 输出 ID 和填充 ID 现在初始化为空 array('q') 而非空列表
        self.output_ids = array("q")
        self.fill_ids = array("q")
        self.session = session
        # ...

    @property
    def output_ids_through_stop(self) -> array[int]:
        """获取通过 stop 条件后的输出 ID（包含 stop 位置）"""
        if self.finished_len is not None:
            return self.output_ids[:self.finished_len]
        return self.output_ids
```

python/sglang/srt/utils/common.py

新增核心工具函数 flatten_arrays_to_int64_tensor，用于高效将 array 列表转为张量

```
# python/sglang/srt/utils/common.py（新增片段）

def flatten_arrays_to_int64_tensor(
    parts: List[array],
    device: str = "cpu",
    pin_memory: bool = False,
) -> torch.Tensor:
    """将 array('q') 列表扁平化为 int64 张量。
    利用 np.frombuffer 零拷贝视图避免逐元素装箱。
    """
    if not parts:
        return torch.empty(0, dtype=torch.int64, device=device)
    # 零拷贝视图，无需逐元素转换
    views = [np.frombuffer(p, dtype=np.int64) for p in parts]
    # 拼接：多个时 concat，单个时直接使用
    combined = np.concatenate(views) if len(views) > 1 else views[0]
    tensor = torch.from_numpy(combined)
    if pin_memory:
        tensor = tensor.pin_memory()
    return tensor.to(device, non_blocking=True)
```

评论区精华

审查者 merrymercy 要求将 flatten_arrays_to_int64_tensor 从内联函数移到 utils/common.py，并将类型注解从 array 改为 array[int]；还要求清理所有仍为

origin_input_ids 等字段赋 list 的调用点，最终作者移除了过渡用的 setter 并强制使用 array。另外，关于 TokenizedGenerateReqInput.input_ids 为何成为 Optional[array]，作者解释在 input_embeds 路径下输入 ID 可能为 None，本变更恰好修正了原有的错误类型注解。审查者还建议移动 benchmark 文件到合适目录，作者已执行。

- flatten 函数移动和重命名 (design): Jialin 已移动至 utils/common.py 并重命名，回复 'Move to utils/common.py'。
- 类型注解应明确元素类型 array[int] (style): Jialin 更新了所有类型注解，回复 'Updated.'。
- 清理所有赋值 list 的调用点，移除过渡 setter (correctness): Jialin 彻底清理了所有调用点并删除了过渡 setter，回复 'Removed, and update all callers instead.'。
- TokenizedGenerateReqInput.input_ids 为何标记为 Optional (question): Jialin 提供推理，merrymercy 接受该解释。

风险与影响

- 风险：
  - 兼容性风险：Req 字段类型变更可能影响未同步的外部代码或钩子，但已通过强制 array 并调整所有调用点降低风险。
  - 性能风险：short prompt 场景可能存在轻微性能回退，但已在 benchmark 中确认可接受。
  - 类型强制风险：RadixKey 要求 array 后，任何仍传入 list 的调用会因类型不匹配导致 TypeError，但已通过全面审计修复所有生产调用点，并在测试中覆盖。
  - 数据契约变更：pad_input_ids 接口签名改变，但所有模型实现均已更新。
- 影响：对用户基本透明，对内部开发者需要适应新类型；系统性能在长 prompt 场景有明显提升，并为后续 Rust 迁移提供基础；团队需在后续开发中注意保持 array('q') 习惯。
- 风险标记：核心路径变更，数据契约变更，兼容性风险，类型强制风险

关联脉络

- PR #26085 drop FutureIndices wrapper class: 同为性能重构，降低 Python 到 C++ 桥接开销；类似的数据结构简化风格。