PR #35125 完整报告

sgl-project/sglang

[Rust Server] Add e2e latency metadata and fix Sarashina import

合并时间：2026-08-18 17:08

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35125

执行摘要

- 一句话：Rust /generate 增加 e2e_latency 元数据，修 Sarashina 导入
- 推荐动作：值得精读 rust/sglang-server/src/api_server/native_api.rs，重点看 RequestTiming 的设计——把 API 层计时与 scheduler 内部消息解耦，并利用 join_all 在保持顺序的同时并发观察终端输出。这个模式对后续 Rust server 增加 TTFT、ITL 等指标有直接参考价值。建议后续补充针对 e2e_latency 的端到端测试，覆盖 unary、batch、流式终止帧三种形态。

功能与动机

PR body 明确要求“add Python-compatible meta_info.e2e_latency to terminal Rust /generate responses”，并“preserve per-request latency semantics for unary, batch, cumulative streaming, and incremental streaming”。Rust server 作为 Python 后端的数据面替代，需要在对齐响应结构的同时补齐端到端延迟指标；Sarashina2 Vision 部分则是因为多模态请求类型已从 mm_utils 迁往 schedule_batch，旧导入路径已失效。

实现拆解

实现分三步完成：

1. 新增 API 层计时结构：在 rust/sglang-server/src/api_server/native_api.rs 中引入 RequestTiming，记录 created_at、time_to_first_token 和 e2e_latency 三个字段，并提供 observe_first_output、finish、terminal_latencies 方法。generate 在 into_requests 完成 body 归一化后、prefetch 之前创建计时实例，与 Python 的 APIServerReqTimeStats 起始点对齐，注释中明确这是为避免把 API 专属时间戳塞入 scheduler 消息。
2. 把计时贯穿 unary 与 batch 两条响应路径：generate_single、generate_batch、drain_unary 均接收 RequestTiming。drain_unary 在收到首个 Frame 或 Done 时调用 observe_first_output，在 Done 或 Error 时调用 finish，并在正常终态通过 add_e2e_latency 把延迟写入 meta_info。batch unary 从原来的“按序逐个 drain”改为 futures::future::join_all 并发 drain，join_all 保持返回顺序，同时让每个请求的终端输出能被及时观察，保证 e2e_latency 准确。流式路径则通过 generation_event_stream 携带 timing，在终端帧的 terminal_stream_frame_string 中注入延迟字段。

3. 修复 Sarashina2 多模态导入：python/sglang/srt/models/sarashina2_vision.py 将 MultimodalDataItem、MultimodalInputs 的导入从 sglang.srt.managers.mm_utils 改到 sglang.srt.managers.schedule_batch，与类型新归属地保持一致。

测试配套：PR 仅依赖 cargo test（241 passed）与 cargo clippy 验证，没有新增独立测试文件对 e2e_latency 做显式断言，属于本次变更的覆盖缺口。

关键文件：

- rust/sglang-server/src/api_server/native_api.rs（模块 API 入口；类别 source；类型 entrypoint；符号 new, observe_first_output, finish, terminal_latencies）：Rust native API 入口的核心改动文件：新增 RequestTiming 计时结构、贯穿 generate_single/generate_batch/drain_unary 全部响应路径，把 e2e_latency 写入 meta_info，并将 batch unary 改为 join_all 并发 drain。这是该 PR 的技术主体。
- python/sglang/srt/models/sarashina2_vision.py（模块 模型层；类别 source；类型 data-contract）：修复 Sarashina2Vision 的多模态类型导入：MultimodalDataItem 与 MultimodalInputs 改从 schedule_batch 导入，避免因 mm_utils 中类型迁移导致的导入失败。

关键符号：new, observe_first_output, finish, terminal_latencies, generate_single, generate_batch, drain_unary, add_e2e_latency, terminal_stream_frame_string, timed_receiver

关键源码片段

rust/sglang-server/src/api_server/native_api.rs

Rust native API 入口的核心改动文件：新增 RequestTiming 计时结构、贯穿 generate_single/generate_batch/drain_unary 全部响应路径，把 e2e_latency 写入 meta_info，并将 batch unary 改为 join_all 并发 drain。这是该 PR 的技术主体。

```
/// API 本地单请求计时。
///
/// Python 在首个输出 batch 记录 time-to-first-token，在请求结束时记录
/// end-to-end 延迟。这里保留两个测量值，即使 `/generate` 目前只暴露
/// `e2e_latency`——避免把 API 专属时间戳塞进 scheduler 消息。
#[derive(Clone, Debug)]
struct RequestTiming {
    // TODO: 将请求生命周期计时抽到独立的 tracing/metrics 模块，
    // 并与 Python 的 APIServerReqTimeStats 对齐设计。
    created_at: Instant,
    time_to_first_token: Option<Duration>,
    e2e_latency: Option<Duration>,
}

impl RequestTiming {
    fn new() -> Self {
        Self {
            created_at: Instant::now(),
            time_to_first_token: None,
```

```
            e2e_latency: None,
        }
    }

    // 只在首个输出到达时记录一次 TTFT，后续帧不会覆盖。
    fn observe_first_output(&mut self) {
        self.time_to_first_token
            .get_or_insert_with(|| self.created_at.elapsed());
    }

    // 请求结束时记录 e2e，同样只保留第一次值。
    fn finish(&mut self) {
        self.e2e_latency
            .get_or_insert_with(|| self.created_at.elapsed());
    }

    // 两个指标都齐了才返回，供终端帧写入 meta_info。
    fn terminal_latencies(&self) -> Option<(Duration, Duration)> {
        Some((self.time_to_first_token?, self.e2e_latency?))
    }
}

// drain_unary 中关键的计时挂载点：
while let Some(item) = rx.recv().await {
    match item {
        EgressItem::Frame(out) => {
            // 首个帧触发 TTFT 观测，与 Python 在首个输出 batch 记录一致。
            timing.observe_first_output();
            acc.fold(&out);
        }
        EgressItem::Done(out) => {
            // 终帧也算首个输出（单 token 请求场景），并结束 e2e 计时。
            timing.observe_first_output();
            timing.finish();
            acc.fold(&out);
            let final_out = acc.into_output();
            // 校验类中止会携带自己的 HTTP 状态码与诊断信息。
            if let Some((code, message)) = final_out
                .finish_reason
                .as_ref()
                .and_then(|f| f.abort_status())
            {
                let status = StatusCode::from_u16(code)
                    .unwrap_or(StatusCode::INTERNAL_SERVER_ERROR);
                return (status, error_value(code, message), true);
            }
            // 正常终态：把计时写入 meta_info.e2e_latency 后返回。
            let mut value = frame_value(&final_out, rid_str);
            add_e2e_latency(&mut value, &timing);
```

```
            return (StatusCode::OK, value, true);
        }
        EgressItem::Error(e) => {
            // 错误路径也结束计时，避免 e2e_latency 悬挂为 None。
            timing.finish();
            let code = e.http_status();
            let status = StatusCode::from_u16(code)
                .unwrap_or(StatusCode::INTERNAL_SERVER_ERROR);
            return (status, error_value(code, e.message()), true);
        }
    }
}
```

python/sglang/srt/models/sarashina2_vision.py

修复 Sarashina2Vision 的多模态类型导入：MultimodalDataItem 与 MultimodalInputs 改从 schedule_batch 导入，避免因 mm_utils 中类型迁移导致的导入失败。

```
# 多模态请求类型已从 mm_utils 迁往 schedule_batch，
# 这里同步更新导入来源，避免 ImportError。
from sglang.srt.managers.mm_utils import (
    MultiModalityDataPaddingPatternMultimodalTokens,
    general_mm_embed_routine,
)
from sglang.srt.managers.schedule_batch import MultimodalDataItem, MultimodalInputs
```

评论区精华

该 PR 没有任何 review 评论或讨论线程，作者 merrymercy 以 3 个连续 commit 完成变更并自行合并。因此没有可提炼的争议点或已解决 / 未解决疑虑。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 响应契约变化：/generate 的终端响应 meta_info 新增 e2e_latency 字段，对严格 schema 校验的客户端是兼容的增量，但依赖字段枚举的客户端可能告警；native_api.rs 是 Rust server 的入口层，属于数据面核心路径。
  2. batch 并发 drain 的资源开销：join_all 会同时 poll 所有 receiver，大批量请求（例如数百个）会同时持有多个 drain_unary 任务，虽不阻塞调度，但会在 API 层增加并发 task 和内存占用。
  3. 计时语义对齐风险：created_at 在 prefetch 之前开始，因此 e2e_latency 包含媒体下载 / 文件读取等 I/O 时间，与 Python 起点的对齐依赖注释中的约定，若后续调整 generate 入口顺序可能破坏一致性。
  4. 测试缺口：本次没有新增任何测试文件，cargo test 仅能保证编译与既有行为不回归，无法验证 e2e_latency 数值正确性和四种响应形态下的字段存在性。

5. Sarashina 导入修复风险低：仅调整 import 来源，但若 schedule_batch 与 mm_utils 之间存在循环依赖，未来重构时需留意该文件的导入稳定性。 - 影响：影响面集中在 Rust server 数据面：所有 /generate 调用方都会在终端响应的 meta_info 中看到新增的 e2e_latency，与 Python 端行为对齐；batch unary 的并发 drain 对大 batch 场景能缩短整体响应延迟，同时保持输出顺序不变。对 Sarashina2Vision 用户，修复了因类型迁移导致的加载 / 推理报错。对团队而言，该 PR 进一步加固了 Rust server 与 Python 后端在响应元数据上的兼容性，为后续运维指标采集提供了基础。影响程度中等，主要是入口层契约增强与一个模型文件的导入修正。 - 风险标记：Rust 入口层变更，无新增测试文件，batch 并发 drain 资源开销，响应 schema 新增字段

关联脉络

- 暂无明显关联 PR