PR #48107 完整报告

vllm-project/vllm

[Rust][Benchmark] Port in vllm-bench

合并时间：2026-07-17 14:25

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/48107

执行摘要

PR #48107 将原先独立仓库的 Rust 基准测试工具 vllm-bench 整体移植到 vLLM 主仓库的 rust/src/bench/ 下，新增约 17k 行 Rust 代码，提供与 Python 版 vllm bench serve 对等的在线吞吐测试能力，支持多后端、多数据集、多轮对话、SLO 验证等特性。该 PR 为后续深度集成和统一 CLI 奠定了基础。

功能与动机

PR body 直接指向 https://github.com/vllm-project/vllm-bench，表明其目的就是将已经独立开发的 vllm-bench 工具迁入主仓库，以实现代码统一管理、简化 CI 流程、降低双份维护成本。

实现拆解

1. 项目结构创建：在 rust/ 工作空间中新增 bench 子 crate，配置 Cargo.toml 依赖（tokio、reqwest、clap、indicatif、tiktoken-rs 等）并注册为工作空间成员；更新 .pre-commit-config.yaml 以包含 Rust 格式化检查。
2. 核心模块实现：
   - cli.rs：通过 clap 定义 Cli 结构体，枚举 BackendKind（vllm/openai/openai-chat/embeddings/rerank 等）、DatasetName（random/sharegpt/sonnet/hf/custom 等）、LoraAssignment 等。
   - config.rs：BenchConfig 从 Cli 构建并验证，RangeRatio 支持两种格式的随机范围（单一浮点或 JSON 对象），并增加与 Python 版语义对齐的错误提示。
   - benchmark.rs：run_benchmark 函数驱动完整基准周期，包含 DNS 预解析（避免高并发下 DNS 失败）、SpecDecode 指标采集、速率控制、进度条显示和“稳态”检测。
   - multi_turn.rs：支持多轮对话评估，可自动从服务器获取模型并截断历史到 max_model_len。
   - datasets/：包括 HF 数据集在线下载（hf_dataset.rs）、随机生成（random.rs/random_mm.rs）、ShareGPT/Sonnet 解析等，其中 hf_dataset.rs 可自动检测列格式（chat/text/combined）并支持分页。
   - output/json.rs：build_result_json 严格对齐 Python 输出 schema，确保结果通用。
   - metrics/：calculator.rs 计算 TTFT/TPOT/ 吞吐量等指标，steady_state.rs 通过滑动窗口自动选择稳定测量区。
   - sweep.rs：支持并发度和请求率的自动扫描（sweep）。
   - tiktoken.rs：封装 tiktoken-rs 实现 token 化，支持文件加载和内置 BPE。

3. 构建与代码风格修复：BugenZhao 提交了 fmt with nightly config、fix lint & tweak deps、Add missing SPDX header 等，使项目满足仓库 CI 要求。
4. 内联测试：各关键模块附有少量 #[cfg(test)] 测试，如 benchmark.rs 中 test_filter_requests_by_max_model_len_* 和 multi_turn.rs 中 test_valid_prefix_len_for_max_model_len_with_history。

rust/src/bench/src/benchmark.rs

基准测试核心入口，包含 DNS 预解析、请求调度、进度跟踪、SpecDecode 统计等关键逻辑。

```
/// Pre-resolve the hostname in `base_url` and pin all resolved IPs on the
/// client builder via [`reqwest::ClientBuilder::resolve_to_addrs`]. This
/// avoids repeated DNS lookups under high concurrency which can cause
/// transient "Temporary failure in name resolution" errors while preserving
/// happy-eyeballs and multi-A failover.
///
/// Skipped when:
/// - URL parse fails
/// - host is already an IP
/// - host is a loopback name (resolved from `/etc/hosts`, no DNS pressure
///   and dual-stack ambiguity between `127.0.0.1` and `::1` breaks
///   IPv4-only servers like vLLM)
/// - resolution fails
pub fn pre_resolve_dns(
    base_url: &str,
    mut builder: reqwest::ClientBuilder,
) -> reqwest::ClientBuilder {
    let parsed = match url::Url::parse(base_url) {
        Ok(u) => u,
        Err(_) => return builder, // 无法解析 URL 时跳过
    };
    let host = match parsed.host_str() {
        Some(h) => h,
        None => return builder, // 无 host 时跳过
    };
    // 如果已经是 IP，跳过预解析
    if host.parse::<std::net::IpAddr>().is_ok() {
        return builder;
    }
    // 避免解析 localhost（IPv4/IPv6 双栈问题）
    let host_lower = host.to_ascii_lowercase();
    if host_lower == "localhost"
        || host_lower == "ip6-localhost"
        || host_lower.ends_with(".localhost")
    {
        return builder;
    }
    let port = parsed.port_or_known_default().unwrap_or(80);
    let addr_str = format!("{host}:{port}");
```

```
    match std::net::ToSocketAddrs::to_socket_addrs(&addr_str) {
        Ok(addrs) => {
            // 分离 IPv4 和 IPv6，优先 IPv4
            let mut v4 = Vec::new();
            let mut v6 = Vec::new();
            for addr in addrs {
                if addr.is_ipv4() {
                    v4.push(addr);
                } else {
                    v6.push(addr);
                }
            }
            v4.extend(v6); // IPv4 排前
            if !v4.is_empty() {
                let ips: Vec<_> = v4.iter().map(|a| a.ip()).collect();
                println!("Pre-resolved {host} -> {ips:?}");
                builder = builder.resolve_to_addrs(host, &v4);
            }
        }
        Err(e) => {
            eprintln!("Warning: DNS pre-resolution for '{host}' failed: {e}");
        }
    }
    builder
}
```

rust/src/bench/src/config.rs

配置解析核心，定义了 BenchConfig、RangeRatio、GoodputConfig 等关键类型，实现 CLI 到配置的转换与验证。

```
/// Range ratio for sampling input/output lengths, matching Python
/// `vllm bench serve`: lengths are drawn uniformly from [len*(1-r), len*(1+r)].
/// A single float applies to both; the JSON form '{"input": r1, "output": r2}'
/// controls them independently. Each ratio must be in [0, 1).
#[derive(Debug, Clone, Copy, PartialEq)]
pub struct RangeRatio {
    pub input: f64,
    pub output: f64,
}

impl RangeRatio {
    /// Parse `--random-range-ratio`: a bare float or a JSON object with
    /// "input" and "output" keys.
    pub fn parse(raw: &str) -> Result<Self> {
        let trimmed = raw.trim();
        let (input, output) = if let Ok(v) = trimmed.parse::<f64>() {
            (v, v) // 单个浮点数，输入输出共用
        } else {
            // 尝试解析 JSON 对象
```

```
            let v: serde_json::Value = serde_json::from_str(trimmed).map_err(|_| {
                BenchError::Config(format!(
                    "Invalid --random-range-ratio '{raw}': expected a float or \
                     '{{\"input\": r1, \"output\": r2}}'"
                ))
            })?;
            let obj = v.as_object().ok_or_else(|| {
                BenchError::Config(
                    "--random-range-ratio JSON form must be an object with \
                     'input' and 'output' keys"
                        .into(),
                )
            })?;
            // 从对象中提取 input 和 output
            let get = |key: &str| -> Result<f64> {
                obj.get(key).and_then(|v| v.as_f64()).ok_or_else(|| {
                    BenchError::Config(format!(
                        "--random-range-ratio JSON form must contain a numeric '{key}' key"
                    ))
                })
            };
            (get("input")?, get("output")?)
        };
        // 验证范围在 [0,1) 内
        for (name, r) in [("input", input), ("output", output)] {
            if !(0.0..1.0).contains(&r) {
                let hint = if r == 1.0 {
                    " NOTE: semantics now match Python vllm bench serve — lengths are \
                     sampled from [len*(1-r), len*(1+r)] and 0.0 means fixed length. \
                     The old Rust-only default 1.0 ([len*r, len]) is no longer valid."
                } else {
                    ""
                };
                return Err(BenchError::Config(format!(
                    "--random-range-ratio {name} ratio must be in [0, 1), got {r}.{hint}"
                )));
            }
        }
        Ok(Self { input, output })
    }
    // ... input_bounds, output_bounds, is_fixed 等方法省略
}
```

评论区精华

BugenZhao 在审批时肯定了这个移植，并指出后续改进方向：

> “Great to see this! We can make some follow-up changes to integrate the bench entry point more tightly with the existing vllm-rs, and try to reuse more components

> between them." 这表明当前为初步移植，后续将推动与现有 Rust 模块（vllm-rs）的深度融合和组件复用。

风险与影响

风险：

- 新增大量 Rust 代码（~17k 行）增加构建时间和 CI 缓存压力。
- 与 Python 版基准测试的功能对等需要持续维护，输出 JSON schema 的同步更新可能出现疏漏。
- 多后端适配可能随 vLLM API 演进而滞后。

影响：

- 用户：获得一个与 Python 版功能一致且性能更优的基准测试工具，支持多轮对话和多种数据集。
- 系统：扩展了 Rust 代码库，增加编译依赖，但工具高度模块化，风险可控。
- 团队：需要 Rust 维护能力，但统一了基准测试的实现入口，减少双份维护。

关联脉络

本 PR 是 vllm-bench 从独立仓库迁入的第一步，后续可能通过关联 PR 合并 CLI 入口、复用 vllm-rs 的 HTTP 客户端和 tokenizer 组件。历史 PR 中未发现直接关联项，但仓库中已有 rust/ 模块（如 vllm-rs），未来可能演进为统一的 Rust 工具集。