PR #51863 完整报告

vllm-project/vllm

[Bugfix][Benchmark] Check readiness before tokenizer init in rust vllm-bench

合并时间：2026-08-19 10:52

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51863

执行摘要

PR #51863 修复 rust vllm-bench 在服务端尚未就绪时提前失败的问题。改动将 --ready-check-timeout-sec 应用到两条会先于就绪探测的启动依赖请求：省略 --model 时的 /v1/models 自动发现，以及 tokenizer 回退到服务端 /tokenize 时的连通探针。最终合并版本放弃第一版“把合成就绪探测提前到 tokenizer 初始化之前”的激进方案，改为提炼泛型 retry_with_timeout 助手让启动依赖请求自身带重试能力，同时保留基于数据集首条请求的原位就绪探测，从而规避 review 指出的合成探测 payload 失真（min_tokens 冲突、--skip-tokenizer-init 拒绝文本 prompt、multimodal / token-id 形状不兼容）与串行化等待副作用。影响范围仅限 rust/bench 工具，服务端行为零变化。

功能与动机

PR body 明确：Apply --ready-check-timeout-sec to the startup-dependent requests that can run before the benchmark readiness probe: retry /v1/models discovery when --model is omitted; retry the /tokenize probe when tokenizer loading falls back to the server; retain the dataset-derived readiness request immediately before tokenizer verification and the timed run.

背景痛点：vllm-bench 的完整就绪探测原本排在 tokenizer 加载与数据集生成之后，但模型自动发现（省略 --model）与服务端 tokenizer 回退探针都发生在更早阶段，且不带重试。服务端冷启动期间，这两类请求会直接以 connection refused 或 400 失败，导致 benchmark 立即退出，--ready-check-timeout-sec 形同虚设。修复目标是让“等待服务端就绪”的策略覆盖全流程，同时不牺牲数据集生成与服务端启动的并行重叠。

实现拆解

1. 提炼通用重试原语：在 rust/src/bench/src/ready_checker.rs 新增泛型函数 retry_with_timeout，接受 timeout_seconds、retry_interval 与闭包 operation，循环执行直到成功或超时，超时后返回 BenchError::EndpointTimeout。timeout_seconds == 0 时只执行一次并直接透传结果或错误，保证与旧行为兼容。同时把 wait_for_endpoint 的错误输出从 e.to_string() 改为 e.as_report()。
2. 收敛并重试模型发现：benchmark.rs 的 get_first_model_from_server 与 multi_turn.rs 的 get_first_model 两份私有实现被删除，统一为 ready_checker::get_first_model。新函数在 GET {base_url}/v1/models 外层包上 retry_with_timeout(timeout_seconds, 5, ...)，支持 extra_headers 与 OPENAI_API_KEY 注入，并在响应无 data[0] 时返回 BenchError::Config。

3. 服务端 tokenizer 探针接入重试：tokenizer.rs 的 ServerTokenizer::new 新增 timeout_seconds 参数，构造后把 /tokenize 探针 encode_async("test") 包进 retry_with_timeout；load_tokenizer 的 server_info 由 Option<(&str, &str)> 扩展为 Option<(&str, &str, u64)>，透传超时配置。
4. 编排器透传配置：run_benchmark 与 run_multi_turn_benchmark 在省略 --model 分支调用 get_first_model(..., config.ready_check_timeout_sec)，在加载 tokenizer 分支构造 server_info 时带上 config.ready_check_timeout_sec。原有的数据集派生就绪探测（携带真实首条请求的 token / multimodal / pooling 输入）保留在计时运行之前，未被改动。
5. 测试配套：在 ready_checker.rs 内新增 #[cfg(test)] 模块，包含 retry_succeeds_after_transient_failures（前两次失败、第三次成功，验证重试收敛）与 zero_timeout_does_not_retry（超时 0 时仅执行一次）两个单元测试。PR 说明中验证了 cargo test -p vllm-bench（150 passed）、cargo clippy、cargo fmt 与 pre-commit。

rust/src/bench/src/tokenizer.rs

ServerTokenizer::new 新增 timeout_seconds 参数，/tokenize 探针接入 retry_with_timeout；load_tokenizer 的 server_info 由二元组扩展为三元组，是本 PR 覆盖的第二条启动依赖请求。

```
// rust/src/bench/src/tokenizer.rs —— 服务端 tokenizer 探针接入同一重试助手
impl ServerTokenizer {
    /// 创建服务端 tokenizer，并在 `timeout_seconds` 内重试 `/tokenize` 探针。
    ///
    /// 探针同时承担“验证连通性 + 估算 vocab size”两个职责；
    /// 服务端可能仍在加载模型，因此必须复用 `retry_with_timeout`。
    pub async fn new(base_url: &str, model: &str, timeout_seconds: u64) -> Result<Self> {
        let client = reqwest::Client::builder()
            .timeout(std::time::Duration::from_secs(30))
            .build()
            .map_err(|e| BenchError::Tokenizer(format!("Failed to build HTTP client: {e}")))?;

        let tokenize_url = format!("{base_url}/tokenize");
        let detokenize_url = format!("{base_url}/detokenize");

        let st = Self {
            client,
            runtime: tokio::runtime::Handle::current(),
            tokenize_url,
            detokenize_url,
            model: model.to_string(),
            cached_vocab_size: 0,
        };

        // Probe the endpoint to verify it works and discover vocab size
        let test_tokens = crate::ready_checker::retry_with_timeout(timeout_seconds, 5, || {
            st.encode_async("test")
        })
        .await?;
```

```
        // 由最大 token id 反推 vocab size 下限，供 `get_allowed_tokens` 采样使用
        let max_id = test_tokens.iter().copied().max().unwrap_or(0);
        let estimated_vocab = (max_id * 2).max(131072);

        Ok(Self {
            cached_vocab_size: estimated_vocab,
            ..st
        })
    }

    // ... 其余编码逻辑保持不变 ...
}
```

评论区精华

esmeetu 用 Claude Code 对第一版提交做了 AI 辅助审查，核心交锋如下，这些发现直接促成了第二版重构。

> “Model auto-resolution still hits the server before this ready check... the benchmark exits immediately with a connection-refused error instead of waiting `--ready-check-timeout-sec`.”
>
> “The synthetic probe hardcodes `output_len: 1`... so `--extra-body '{"min_tokens": 32}'` makes every probe fail with 400 `min_tokens must be less than or equal to max_tokens` against a perfectly healthy server.”
>
> “replacing the dataset-derived probe with a synthetic one (`"test"`, `output_len: 1`) makes the check fail forever against healthy servers with `--extra-body min_tokens` or `--skip-tokenizer-init`, while also passing trivially when the real request shape (multimodal / token-id prompts) would be rejected — converting an upfront abort into a full run that saves `completed=0` garbage results.”
>
> “This serializes the server-boot wait before tokenizer load and dataset generation... now wall-clock startup is `boot_time + generation_time` instead of roughly `max` of the two.”

结论：第二版 commit “Retry startup-dependent requests” 放弃合成探测、回归“数据集派生探测 + 启动依赖请求自带重试”的路线，回应了上述正确性与效率问题；esmeetu 最终 APPROVED（LGTM）。review 中标注为 confirmed 的若干正确性发现均针对第一版实现，在合并版本中已不适用。

风险与影响

风险：

- 重试间隔在 get_first_model 与 ServerTokenizer::new 内硬编码为 5 秒，与 wait_for_endpoint 调用方传参一致，但未来若需可配置需统一处理。
- 新增单测只覆盖 retry_with_timeout 本身，未覆盖 get_first_model 的 header / API key 注入与 ServerTokenizer::new 的网络路径；这两处是否有回归依赖 CI 集成测试兜底。

- timeout_seconds == 0 的语义是“单次执行、失败即返回”，虽然与旧的“无重试”行为一致，但未来复用者可能误认为 0 表示无限等待，需要留意语义说明。
- 错误日志从 *e*.to_string() 改为 *e*.as_report()，输出格式有细微变化，不影响行为。

影响：

- 用户侧：rust vllm-bench 在服务端慢启动、省略 --model、本地无 tokenizer 文件需回退服务端 /tokenize 的场景不再提前崩溃，--ready-check-timeout-sec 真正生效。
- 系统侧：纯客户端工具改动，不影响 vLLM 服务端任何路径。
- 团队侧：retry_with_timeout 成为 rust/bench 可复用的等待原语，消除两份重复的模型发现代码。

关联脉络

本次改动没有直接的前序 PR，但同属 vLLM 近期“rust 侧正确性打磨”主题：PR #52671 让 Rust 前端等待所有 utility 调用收敛后再报错，与本次“等待所有启动依赖请求就绪”的语义一致；PR #52703 在 rust 协议层新增字段并配套 cargo 测试，说明 rust frontend / bench 工具链正在持续补齐边界行为。后续方向可预见：将 retry_with_timeout 复用到更多启动阶段的 HTTP 调用（如 /v1/health 探活），或把重试间隔抽为配置项。