

PR #56990 完整报告

vllm-project/vllm

[Rust Frontend] Add iteration token histogram

合并时间: 2026-09-15 20:12

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56990>

PR#56990 [Rust Frontend] Add iteration token histogram 分析报告

执行摘要

本 PR 在 Rust 前端补齐了 Python 侧已有的 `vllm:iteration_tokens_total` 直方图: 在 `rust/src/metrics/src/request.rs` 定义分桶与注册, 在 `rust/src/engine-core-client/src/metrics.rs` 新增按 (model_name, engine) 缓存的 `IterationMetricHandles`, 并在 `rust/src/engine-core-client/src/client/imp.rs` 的输出分发循环里, 对每个非空输出批次累计「新生成 token + prefill 计算 token」后观测一次; 配套让 `rust/src/mock-engine/src/engine.rs` 在首个输出上发送一次性 `PrefillStats`。改动 90 行、5 个文件, 属可观测性对齐类特性, 不触及推理核心路径, 主要风险在指标口径依赖与测试覆盖, 而非计算正确性。

功能与动机

PR body 的第一句就是目标陈述:

Expose `vllm:iteration_tokens_total` in the Rust frontend, including its `_count`, `_sum`, and `_bucket` series.

这意味着 Rust 前端要与 Python 前端输出同名的迭代级 token 吞吐指标。值得注意的历史包袱是, `rust/src/metrics/src/request.rs` 里原本留有这样一段结论:

We intentionally don't support iteration-level histograms for now, since it seems to make more sense if the engine maintains these metrics and frontend simply forwards.

本PR删除该注释并把统计放到前端, 口径被显式锚定为「frontend-visibleoutputbatches」。此外, 为了让两个前端能在同一台 mock 引擎上对账 prompt token, mock 也必须能产生 prefill 统计——这是第二个 commit 存在的原因。

实现拆解

1. 指标定义与注册 (`rust/src/metrics/src/request.rs`): 新增

`ITERATION_TOKENS_BUCKETS`, 取值 1、8、16 ... 16384 (13 个上界, 加 +Inf 共 14 个 bucket), 配套构造函数 `iteration_tokens_histogram()`。RequestMetrics 增加 `iteration_tokens_total: HistogramFamily` 字段, RequestMetrics::register 以 `vllm:iteration_tokens_total` 注册并使用 `Family::new_with_constructor` 保证每个 label 组

合共享同一套分桶。同时删除前述否定性注释与占位字段，完成设计反转。

2. 句柄缓存 (`rust/src/engine-core-client/src/metrics.rs`) : 新增 `IterationMetricHandles` , 通过 `get_or_create_owned(&EngineLabels { model_name, engine })` 为每个引擎预创建 `child series`。这一步同时承担两个目的: 热路径不再做 `Family` 查找; 零流量时 `/metrics` 也有 0 值 `series` 可供对账。
3. 客户端接入 (`rust/src/engine-core-client/src/client/imp.rs`) : `ClientInner` 增加 `iteration_metrics: BTreeMap<u32, IterationMetricHandles>`, 在 `new()` 中按 `engine.engine_id.engine_index()` 组装, `filter_map` 会跳过拿不到索引的引擎。
`run_output_dispatcher_loop` 在 `RequestBatch` 分支累加 `output.new_token_ids.len()` 与 `prefill_stats.num_computed_tokens`, 批次非空且引擎可查时 `observe`。累加发生在 `sender` 校验之后, 因此只统计本前端跟踪的活跃请求, 避免多前端重复计数。
4. mock 与测试配套 (`rust/src/mock-engine/src/engine.rs`、
`rust/src/mock-engine/src/tests.rs`) : `ActiveRequest::step` 在 `generated == 0` 时构造 `PrefillStats { num_prompt_tokens, num_computed_tokens, ..Default::default() }` (`mock` 不模拟前缀缓存, 两者都等于 `prompt` 长度), 后续步为 `None`;
`chunk_size_one_outputs_one_token_per_update` 增加三条断言, 把「`prefill stats` 只出现一次」固化成契约。
5. 验证方式 (未落库) : PR body 描述了完整的手工对账流程——用独立 Rust mock 引擎、
`output chunk size` 取 1/4/8, 发 14 次串行 `/v1/completions` (流式与非流式交替, `prompt 1-20000`、输出 1-17), 逐请求核对零初始化与 `sum/count/` 全部 `bucket`, 最终 `_sum` 均为 52,845, `_count` 分别为 96/32/21, 并自陈「`concurrent batching, cache simulation, and GPU model evaluation remain outside this check`」。

文件	角色
<code>rust/src/engine-core-client/src/client/imp.rs</code>	指标唯一记录点 (dispatcher 循环)
<code>rust/src/metrics/src/request.rs</code>	指标名、描述与分桶定义
<code>rust/src/engine-core-client/src/metrics.rs</code>	per-engine 句柄缓存与零值预初始化
<code>rust/src/mock-engine/src/engine.rs</code>	mock 侧产生一次性 <code>prefill</code> 统计
<code>rust/src/mock-engine/src/tests.rs</code>	<code>prefill stats</code> 出现一次的断言

rust/src/engine-core-client/src/client/imp.rs

指标的唯一记录点: `ClientInner` 新增 `iteration_metrics` 缓存, 并在输出分发循环中按批次累加生成 `token` 与 `prefill` 计算 `token` 后 `observe` 一次, 是整条链路的收口处, 也是正确性风险 (`has_outputs` 与 `sender` 过滤错位) 所在。

```
// rust/src/engine-core-client/src/client/imp.rs
// ClientInner 构造阶段: 按已连接引擎预创建指标句柄,
// 这样即使还没有任何请求, /metrics 中也会出现 0 count / 0 sum 的 series。
let iteration_metrics = engines
    .iter()
    .filter_map(|engine| {
```

```

    // engine_index() 返回 Option: 拿不到索引的引擎直接跳过, 不参与指标打点。
    let engine = engine.engine_id.engine_index()?;
    let handles = IterationMetricHandles::new(&METRICS.request, &model_name, engine);
    Some((engine, handles))
  })
  .collect();

// 输出分发循环: 每个 batch 只做一次 observe。
EngineCoreOutputs::RequestBatch(batch) => {
  // 空批次 (只带 scheduler_stats) 不计入; 注意这里用的是原始 outputs 判断。
  let has_outputs = !batch.outputs.is_empty();
  let mut iteration_tokens = 0_u64;
  let senders = inner.take_senders_for_outputs(&batch.outputs);
  for (output, sender) in batch.outputs.into_iter().zip(senders) {
    let request_id = output.request_id.clone();
    // sender 为 None 表示该请求不由本 frontend 跟踪, 直接丢弃且不计入统计,
    // 避免与其它 frontend 对同一批次重复计数。
    let Some(sender) = sender else {
      debug!(request_id, "dropping output for inactive request");
      continue;
    };

    // 本次迭代的 token 数 = 本步解码出的新 token + 首批输出携带的 prefill 计算量。
    iteration_tokens += output.new_token_ids.len() as u64;
    if let Some(prefill) = &output.prefill_stats {
      // 引擎约定只在首个输出上发一次 prefill_stats;
      // 用 num_computed_tokens 而非 num_prompt_
      // tokens, 前缀缓存命中的部分不计入计算量。
      iteration_tokens += u64::from(prefill.num_computed_tokens);
    }

    let wrapped_output = EngineCoreStreamOutput {
      engine_index: batch.engine_index,
      timestamp: batch.timestamp,
      output,
    };
    if sender.send(Ok(wrapped_output)).is_err() {
      debug!(request_id, "request output stream receiver dropped");
    }
  }
}

// 只要批次非空且该 engine 有缓存句柄就记一次观测。
if has_outputs
  && let Some(handles) = inner.iteration_metrics.get(&batch.engine_index)
  {
    handles.iteration_tokens_total.observe(iteration_tokens as f64);
  }
// ... 后续 finished_requests / scheduler_stats / lora 统计保持不变
}

```

rust/src/metrics/src/request.rs

定义了 `vllm:iteration_tokens_total` 的名字、描述、分桶与注册方式，并删除了原先「不在前端维护 iteration 级直方图」的注释——这是本次设计反转的物证。分桶（2 的幂次到 16384）直接决定 `_bucket` 序列的可读性。

```
// rust/src/metrics/src/request.rs
// 迭代 token 数按 2 的幂次分桶：下界覆盖 decode 单步的 1 token，
// 上界 16384 足以覆盖长 prompt 的 prefill chunk，超过则落入 +Inf 桶。
const ITERATION_TOKENS_BUCKETS: [f64; 13] = [
    1.0, 8.0, 16.0, 32.0, 64.0, 128.0, 256.0, 512.0, 1024.0, 2048.0, 4096.0, 8192.0, 16384.0,
];

fn iteration_tokens_histogram() -> Histogram {
    Histogram::new(ITERATION_TOKENS_BUCKETS.iter().copied())
}

/// Request and output-batch Prometheus families.
pub struct RequestMetrics {
    /// Computed prompt and generated tokens per nonempty engine output batch,
    /// recorded by the engine-core client for requests tracked by this frontend.
    pub iteration_tokens_total: HistogramFamily,

    // Request-derived counters.
    pub num_preemptions: Family<EngineLabels, U64Counter>,
    pub prompt_tokens: Family<EngineLabels, U64Counter>,
    pub prompt_tokens_by_source: PromptTokenSourceCounterFamily,
    pub prompt_tokens_cached: Family<EngineLabels, U64Counter>,
    pub generation_tokens: Family<EngineLabels, U64Counter>,

    // 原本这里写着「前端暂不支持 iteration 级直方图，交给引擎维护更合理」，
    // 本 PR 删除该结论并改为前端侧观测，口径是 frontend-visible output batch.
    // Request lifecycle counters and histograms.
    pub request_success: FinishedReasonCounterFamily,
    // ... 其余请求生命周期指标不变
}

impl RequestMetrics {
    /// Register the request-oriented metric families into the shared registry.
    pub(crate) fn register(registry: &mut Registry) -> Self {
        // Family::new_with_constructor 保证每个新 label 组合都拿到同一套分桶。
        let iteration_tokens_total =
            Family::new_with_constructor(iteration_tokens_histogram as fn() -> Histogram);
        registry.register(
            "vllm:iteration_tokens_total",
            "Histogram of computed prompt and generated tokens per nonempty engine output batch.",
            iteration_tokens_total.clone(),
        );
    }
}
```

```

    // ... 其余 counter / histogram 注册保持不变
    Self { iteration_tokens_total, /* ... */ }
}
}

```

rust/src/engine-core-client/src/metrics.rs

新增 IterationMetricHandles，把 RequestMetrics 的直方图族在 (model_name, engine) 标签下物化成可复用的 owned 句柄，既避免热路径重复做 Family 查找，也实现「零值 series 预初始化」的设计目标。

```

// rust/src/engine-core-client/src/metrics.rs
/// Cached output-batch metric handles for one model and engine index.
pub(crate) struct IterationMetricHandles {
    pub iteration_tokens_total: HistogramMetric,
}

impl IterationMetricHandles {
    pub(crate) fn new(metrics: &RequestMetrics, model_name: &str, engine: u32) -> Self {
        let labels = EngineLabels {
            model_name: model_name.to_string(),
            engine,
        };
        Self {
            // get_or_create_owned 在此刻就把 child 建出来：
            // 1) 后续 observe 不再经过 Family 的哈希查找，热路径更轻；
            // 2) 未产生任何流量的 engine 也会以 0 count / 0 sum 暴露，便于告警与对账。
            iteration_tokens_total: metrics.iteration_tokens_total.get_or_create_owned(&labels),
        }
    }
}

```

评论区精华

本 PR 的 `review_comments_count` 为 0，讨论信息几乎全部来自 review 状态、CI 评论与代码自身的注释：

```

claude[bot]: This pull request is from a fork — automated review is disabled. A
repository maintainer can comment @claude review to run a one-time review.

esmeetu: APPROVED (无正文)

github-actions[bot]: 🔄 This PR is 2 commits behind upstream main. ... No new CI
build was started.

github-actions[bot]: 🔄 Triggered Buildkite CI #89074 for commit cf5b630ed1f7. ⚠️
🔄 This PR is 4 commits behind upstream main. Running CI at your own risk
because --allow-stale was requested.

```

最实质的「讨论」其实是作者在代码里删掉的那段旧结论，以及 PR body 里对口径和验证边界的自我辩护：

Retain per-request output processing; the histogram measures frontend-visible output batches.

Validation uses mock outputs; concurrent batching, cache simulation, and GPU model evaluation remain outside this check.

整体看，这是一次在自动 review 缺席、仅单人 approve、作者自合并的情况下落地的设计反转，缺少对「前端记账 vs 引擎记账」的第三方确认。

风险与影响

- 计数口径错位（正确性）：has_outputs 取自原始 batch.outputs，而计数只累加 sender 有效的输出。若某批次全部输出都因请求不被本前端跟踪而被跳过，仍会 observe(0)，给 le="1" 桶加一次 count 而 _sum 不变，使 _count/_sum 的可解释性下降。
- 协议契约写死在客户端：计数依赖 prefill_stats.num_computed_tokens，注释「The engine emits prefill_stats once, on the first output」把约定固化在 Rust 侧。引擎若改为 chunked prefill 多次上报，或改用 num_prompt_tokens 口径，前端计数会立即与 Python 侧分叉，且当前无断言保护。
- 热路径与基数：run_output_dispatcher_loop 是每引擎每 step 的必经路径，新增了逐 output 的 len()/Option 检查与逐批次的 map 查找 + observe，绝对开销小但频率高；get_or_create_owned 使 series 按 (model_name, engine) 预建，大 DP 部署下标签基数随引擎数线性增长。
- mock 行为变更外溢：mock 输出现在可能携带 prefill_stats，任何假设「mock 只产出 token id 与 finish_reason」的下游测试或脚本需同步适配，本 PR 只更新了 rust/src/mock-engine/src/tests.rs。
- 测试覆盖缺口：直方图的 sum/count/bucket 正确性仅由 PR body 中的 84 次手工 HTTP 请求验证，没有落成仓库内自动化测试；rust/src/mock-engine/src/tests.rs 只覆盖 prefill_stats 出现一次。

影响面方面：对 SRE 是纯增益——Rust 前端首次能提供与 Python 前端同名的迭代吞吐指标，可直接做双前端对照面板；对系统是低成本采样；对团队则意味着「前端维护 iteration 级指标」成为新的既定模式，后续同类指标很可能沿用同一实现套路。

关联脉络

- Rust 前端与 Python 前端的持续对齐：#56593 修复 Redis 前端 DeepSeek 渲染器的 add_generation_prompt 行为，本 PR 继续在指标语义维度拉平两个前端，二者同属 rust/ 目录下的 parity 工作线。
- Prometheus 指标口径治理：#55283 澄清 ITL 与 TPOT 指标语义并补 Grafana 面板，本 PR 为 vllm:iteration_tokens_total 提供了明确的 bucket 与口径注释，属于同一类「让指标可被正确解读」的努力。
- 前缀缓存与 prompt token 记账：#56794 处理 KDA 中段 checkpoint 挤占前缀缓存的问题；本 PR 之所以用 num_computed_tokens 而非 num_prompt_tokens，正是因为前缀缓存命中会改变实际计算量，两者共享同一套 prompt token 语义语境。

- 演进方向：仓库近期大量 PR 集中在 DeepSeek V4.1、MoE 融合算子与 CUDA 图等性能方向（#56962、#56903、#56893），本 PR 属于少数面向可观测性的基础补强，是 Rust 前端从「功能可用」走向「运维可观测」的一步。