# PR #28744 完整报告

sgl-project/sglang

[router] Tokenize prompt once at ingress; forward input_ids to the engine (all policies)

合并时间：2026-06-20 07:07

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28744

## 执行摘要

本 PR 在 sgl-router 入口处统一 tokenize 请求 prompt，将生成的 token ids 传递给后端引擎，消除了引擎侧重复 tokenization。对于长上下文请求（55k-125k tokens），idle TTFT 降低 29%-41%，负载下降低 34%-49%。所有路由策略（cache-aware、sticky、round-robin）共享同一份预计算 tokens，并配有安全守卫确保仅在模型有 chat encoder 且无 tools/multimodal 等干扰时转发。

## 功能与动机

PR 作者在描述中指出：cache-aware 策略会 tokenize prompt 以计算哈希进行路由，引擎随后又会 tokenize 相同 prompt 用于推理，对于 55k-125k token 长上下文，双重 tokenization 带来 100-580ms 的冗余延迟。本 PR 旨在消除该冗余，降低 TTFT 并减少路由器 CPU 开销，同时为优化所有策略而非仅 cache-aware。

## 实现拆解

1. ingress tokenization 决策（chat.rs）：在 chat_completions 中，根据模型是否有 chat encoder（用于 input_ids 转发）或所选策略是否需要 tokens（cache-aware 需要）来决定是否将请求体完整解析为 JSON 并调用 tokenization。仅当任一条件满足时执行完整 parse，避免对短请求的不必要开销。
2. 共享 tokenization 函数（policies/mod.rs）：新增 RequestTokens { ids, engine_equivalent } 结构体和 request_tokens_for() 公共函数。该函数先检查 chat encoder 路径（messages 字段 -> chat encoder 产生 engine-equivalent ids），失败或不存在则退化到 raw prompt 提取（prompt/text）后 tokenize，此时 engine_equivalent 为 false。这一步使得 tokenization 逻辑集中，所有策略共享。
3. 策略侧适配（cache_aware_zmq.rs）：CacheAwareZmqPolicy::select 优先使用 SelectionContext 中的 request_tokens()，若无预计算 tokens 则自行 tokenize（向后兼容）。同时实现了 needs_request_tokens() 方法，让入口决策层知道该策略需要 tokens。
4. outgoing body 构建（chat.rs）：build_outgoing_body 新增 input_ids_safe_to_forward() 安全守卫，确保仅当 ids 是 engine-equivalent 且请求不包含 tools、multimodal、chat_template、reasoning/reasoning_effort、continue_final_message 等会改变 prompt 渲染的字段时才注入 input_ids。同时保留了 messages 字段供引擎推导 stop tokens。
5. 可观测性（metrics.rs）：新增 sgl_router_ingress_tokenize_errors_total{model_id} 计数器，在 chat encoder 路径 tokenize 失败时递增（预期极少），辅助运维发现

tokenization offload 异常。

6. 测试覆盖：新增三个集成测试文件，分别针对 cache-aware、sticky、round-robin 策略，验证正常转发、工具请求阻断、多模态阻断、思考请求阻断等场景。

## experimental/sgl-router/src/server/routes/chat.rs

入口处理函数，新增 ingress tokenization 决策、build_outgoing_body 和 input_ids_safe_to_forward 守卫逻辑，改动量最大（+536/-57）

```
// Tokenize once at ingress whenever it can pay off — decoupled from the
// routing policy, because forwarding `input_ids` is a property of the
// MODEL (does it have a chat encoder so the router can produce
// engine-equivalent tokens?), not of how we pick the worker. Two gates:
//
// * `has_chat_encoder` → a chat request on this model yields
// engine-equivalent ids we can forward as `input_ids` so the engine
// skips re-tokenizing. This enables the offload for EVERY policy —
// sticky and round-robin included — not just cache-aware.
// * `needs_request_tokens()` → the cache-aware policy ALSO wants the
// raw-prompt path tokenized for tree matching even on a model with no
// chat encoder (`/v1/completions` / `text`), which the first gate
// alone wouldn't trigger.
//
// When neither holds, `parse_probe`'s minimal probe is enough, so we keep
// avoiding the full `serde_json::Value` allocation over a (up to 1 MiB)
// body. When parsed, this single value is reused for the routing
// tokenization and the outgoing-body injection below (and PD bootstrap
// injection). `parse_probe` already validated the object shape.
let want_tokens = ctx.tokenizers.has_chat_encoder(&model_str)
    || policy.needs_request_tokens();
let request_value: Option<serde_json::Value> = if want_tokens {
    Some(serde_json::from_slice(&body)
        .map_err(|_| {
            ApiError::BadRequest("invalid request: body must be a JSON object".into())
        })?)
} else {
    None
};

// The ids feed both the routing decision (cache-aware consumes them; other
// policies ignore them) and — when engine-equivalent — the engine itself,
// forwarded as `input_ids` so it skips re-tokenizing the same prompt. The
// ingress owns the tokenize via the shared registry, so the choice of
// policy never changes whether we tokenize.
let request_tokens = request_value
    .as_ref()
    .and_then(|v| request_tokens_for(&ctx.tokenizers, &model_id, v));
```

## experimental/sgl-router/src/policies/mod.rs

新增 RequestTokens 结构体和 request_tokens_for、tokenize_text 等核心函数，作为全局共享的 tokenization 入口

```
/// Produce the routing tokens — and whether they are engine-equivalent —
/// from an already-parsed request body, using the shared tokenizer registry.
pub fn request_tokens_for(
    tokenizers: &TokenizerRegistry,
    model_id: &ModelId,
    value: &serde_json::Value,
) -> Option<RequestTokens> {
    // Chat-encoder path: only when the model has a chat encoder and the
    // request has `messages`. The engine-equivalent flag is set so the
    // caller knows these ids can be forwarded as `input_ids`.
    if tokenizers.has_chat_encoder(&model_id.0) {
        if let Some(messages) = value.get("messages").filter(|m| m.is_array()) {
            if let Some(ids) = tokenizers.encode_chat(&model_id.0, messages) {
                return Some(RequestTokens {
                    ids,
                    engine_equivalent: true,
                });
            }
        }
    }
    // Raw-prompt fallback: extract text from `prompt` / `text` field
    // and tokenize it. These ids are NOT engine-equivalent (the engine
    // will still apply its own template).
    let text = extract_prompt_text_from_value(value)?;
    let ids = tokenize_text(tokenizers, model_id, &text)?;
    Some(RequestTokens {
        ids,
        engine_equivalent: false,
    })
}
```

## 评论区精华

- chat_template 阻断：Review 指出若用户指定自定义 chat_template，转发 default 模板的 input_ids 将静默忽略自定义模板，必须阻断。开发者在最后一个 commit 将其加入阻断列表并更新测试，问题已修复。
- 序列化优化建议：建议使用 serde_json::to_value(ids) 代替手动构造 Array，代码更简洁。该建议未在 diff 中明确体现，仍可改进。

## 风险与影响

- tokenization 一致性：转发依赖 router 与 engine 使用相同 chat encoder，已有 cache-aware 依赖此假设；guard 确保不安全时不转发，不会导致错误但可能错过优化。
- 安全守卫完整性：需持续维护 input_ids_safe_to_forward 列表，确保覆盖所有影响 prompt 渲染的新字段。

- 性能收益明确：实测数据显示长上下文延迟显著降低，短请求增加一次 JSON parse 但影响微小。
- 可运维性增强：新增指标帮助监控 offload 健康，WARN 日志在失败时可见。

## 关联脉络

该 PR 延续了 router 可观测性方向上的改进（如近期 PR#28717 对齐 TTFT 桶），并与其他 router 优化（如 #28717）协同。此外，本 PR 为后续更激进的 offload（如跳过模板渲染）奠定了基础。与引擎侧的 runner 重构 PR 无直接关联。