

PR #56338 完整报告

vllm-project/vllm

[Rust Frontend] Forward per-request watermarking controls

合并时间: 2026-09-14 23:17

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56338>

Rust 前端补齐按请求水印控制: PR#56338 分析

执行摘要

PR#56338 让 Rust 前端首次具备与 Python 前端一致的「按请求开关水印」能力：
`/v1/chat/completions`、`/v1/completions`、原始 `/inference/generate` 路由与 gRPC `GenerateRequest` 都能把 `watermarking` 一路透传到 engine-core 的 `MessagePack` 采样参数，四条路径默认 `true`，显式 `false` 即退出引擎已配置的水印。改动是典型的「逐层加字段 + 分层测试」：14 个文件、+133/-1，默认路径与改动前行为等价，回退风险低；主要风险在于默认值语义散落在多处，以及新增 `wire` 字段在版本混跑时的兼容性。维护者 BugenZhao 触发 Codex 评审后直接批准 ("Thanks @NickLucche !")，全程无返工。

功能与动机

- 直接动机来自 PR body: "Rust frontend requests currently drop `watermarking: false`, so callers cannot opt out of the engine's configured watermark as they can through the Python frontend." Rust 前端此前完全不解析该字段，显式关闭会被静默丢弃，造成与 Python 前端的能力不对齐。
- 上游需求来自 RFC Issue #53916 (Native Text Watermarking Support)：该 RFC 把水印定位为负责的 LLM 部署能力，援引 EU AI Act Article 50(2)、California SB 942 的溯源与透明度要求，并强调水印是采样过程的修改、"cannot be left as an out-of-tree functionality"。前端若无法按请求透传开关，运维侧就无法在合规水印与质量评测场景之间切换。
- PR body 同时给出实现约束: "Add an optional gRPC request field that preserves the distinction between omission and explicit `false`. All paths default to `true`, matching Python." 这句话直接决定了 proto 使用 optional 以及各层默认值的写法。

实现拆解

数据流自上而下分四层改造，外加测试与文档配套。主链路为 HTTP types → `vllm_text::SamplingParams` → `lower_sampling_params` → `EngineCoreSamplingParams` (`msgpack`)，gRPC 则走 proto → `to_text_request` 的平行分支。

层	文件	关键符号	变更
HTTP 入口 schema	rust/src/server/src/routes/openai/chat_completions/types.rs、 ../completions/types.rs	ChatCompletionRequest、CompletionRequest	新增 watermarking: bool + #[serde(default = "default_true")] , Default impl 置 true
请求转换	../chat_completions/convert.rs、 ../completions/convert.rs	prepare_chat_request、prepare_completion_request	把 request.watermarking 拷进内部 SamplingParams
内部采样参数	rust/src/text/src/request.rs	SamplingParams	新增 watermarking: bool, Default 为 true; raw /inference/generate 因此自动获得该字段
下沉与协议编码	rust/src/text/src/lower.rs、 rust/src/engine-core-client/src/protocol/sampling.rs	lower_sampling_params、EngineCoreSamplingParams、 default_watermarking	各加一行解构 / 构造 ; 新增 serde 默认值函数, for_test() 与 7 处快照同步
gRPC 三态	rust/proto/inference.proto、 rust/src/server/src/grpc/convert.rs	GenerateRequest、 to_text_request	proto 增加 optional bool watermarking ; 转换层 unwrap_or(true) 收敛三态
测试与文档	routes/tests.rs、inference/generate/convert.rs、 grpc/convert.rs、engine-core-client/src/tests/*、 docs/features/watermarking.md	三个新测试函数	覆盖 HTTP 全链路、raw 路由、protobuf 往返、Python 兼容 fixture, 并补文档

1. HTTP schema 层先落地字段: ChatCompletionRequest 与 CompletionRequest 都新增 watermarking: bool, serde 默认与 Default impl 双写 true, 行内注释统一为 "Whether to apply the engine's configured watermark to this request."。把「省略即开启」编码进反序列化层, 保证老客户端不传该字段时行为不变。

2. 转换层与内部 SamplingParams 对齐：两个 prepare_* 函数在构造 SamplingParams 时补上 watermarking: request.watermarking; rust/src/text/src/request.rs 新增同名字段并默认 true。因为 raw /inference/generate 路由复用同一个 vllm_text::SamplingParams，它天然获得该字段，测试 prepare_generate_request_preserves_watermarking_defaults_and_opt_out 只需验证 {"sampling_params": {"watermarking": false}} 原样穿过。
3. 下沉到 engine-core 并适配 msgpack 缺键：lower_sampling_params 解构时多绑定一个 watermarking，再原样写入 EngineCoreSamplingParams；协议 DTO 新增 default_watermarking() 与 #[serde(default = "default_watermarking")]，专门应对 Python SamplingParams 的 omit_defaults=True——缺键必须落到 true，否则会被 bool 的默认 false 静默关掉水印。7 处 expect! 快照随之更新，把字段纳入快照比对。
4. gRPC 用 proto3 optional 保留三态：inference.proto 增加 optional bool watermarking，to_text_request 用 req.watermarking.unwrap_or(true) 收敛；测试先做 encode_to_vec() / decode() 往返再断言，覆盖流式与非流式两种分支。这是本 PR 唯一真正的设计取舍：若用普通 proto3 标量，「未设置」与「显式 false」无法区分，opt-out 语义就会丢失。
5. 测试与兼容 fixture 一起补齐：routes/tests.rs 的新测试用三重循环覆盖 2 端点 × 2 种 stream × 3 种取值共 12 个组合；rust/src/engine-core-client/src/tests/client.rs 与 tests/python_compat.py 各加 2 行，把新字段纳入 Python 兼容基准；docs/features/watermarking.md 增加 +6/-1 说明。

rust/src/engine-core-client/src/protocol/sampling.rs

engine-core 协议 DTO 新增 watermarking 字段与 default_watermarking()，是适配 Python omit_defaults=True 缺键语义、避免默认落到 false 的关键改动。

```
// rust/src/engine-core-client/src/protocol/sampling.rs
```

```
// Python 端 SamplingParams 使用 omit_defaults=True，msgpack 会丢弃等于默认值的键；  
// 因此 Rust 侧为 watermarking 单独声明默认值函数，保证「缺键」解码为 true，  
// 否则 bool 的默认值 false 会静默关闭引擎已配置的水印。
```

```
fn default_watermarking() -> bool {  
    true  
}
```

```
#[serde_with::skip_serializing_none]  
#[derive(Debug, Clone, PartialEq, Serialize, Deserialize, DefaultFromSerde)]  
#[serde(default)]  
pub struct EngineCoreSamplingParams {  
    #[serde(default = "default_temperature")]  
    pub temperature: f32,  
    /// Whether to apply the engine's configured watermark to this request.  
    /// 省略该键 -> true，与 Python 前端保持一致；  
    /// 只有显式传 false 才是 opt-out，请求会退出引擎配置的水印。  
    #[serde(default = "default_watermarking")]  
    pub watermarking: bool,  
    #[serde(default = "default_top_p")]  
    pub top_p: f32,
```

```

    // ... 其余采样字段保持不变
}

impl EngineCoreSamplingParams {
    /// 仅供测试使用的默认构造：新增字段必须同步补齐，
    /// 否则 omit-defaults 的 msgpack 回归测试会立刻失败。
    pub fn for_test() -> Self {
        Self {
            temperature: 1.0,
            watermarking: true,
            top_p: 1.0,
            // ... 其余字段
            routed_experts_prompt_start: 0,
        }
    }
}

```

rust/src/server/src/grpc/convert.rs

gRPC 入口用 proto3 **optional** 保留三态并以 **unwrap_or(true)** 收敛，同时新增 encode/decode 往返测试，是本 PR 唯一有信息量的设计取舍所在。

```

// rust/src/server/src/grpc/convert.rs —— to_text_request 中的水印透传
let sampling = req.sampling.as_ref();
let decoding = req.decoding.as_ref();
let stopping = req.stopping.as_ref();
let response = req.response.as_ref();
let kv = req.kv.as_ref();

let mut sampling_params =
    build_sampling_params(req.temperature, sampling, decoding, stopping, response)?;
// proto3 的 optional bool watermarking 保留了「字段未设置」与「显式 false」的区别：
// None（未设置）回退到 true，与 Python 前端默认值一致；
// Some(false) 才是真正的 opt-out，会随请求下沉到 engine-core 关闭水印。
sampling_params.watermarking = req.watermarking.unwrap_or(true);

// ... 既有 KVCacheParameters -> SamplingParams 透传逻辑保持不变

```

同一文件新增的测试把「编码 -> 解码 -> 转换」整条链路固定下来：

```

#[test]
fn watermarking_defaults_and_opt_out_survive_protobuf_conversion() {
    for watermarking in [None, Some(true), Some(false)] {
        let request = pb::GenerateRequest { watermarking, ..base_request() };
        let encoded = request.encode_to_vec();
        for stream in [false, true] {
            let decoded = pb::GenerateRequest::decode(encoded.as_slice()).unwrap();
            let text = to_text_request(decoded, stream, &["test-model".to_string()])
                .expect("convert request");
            // 与 HTTP 路径同一断言：省略 -> true，显式 false -> false。
            assert_eq!(text.sampling_params.watermarking, watermarking.unwrap_or(true));
        }
    }
}

```

```

    }
  }
}

```

rust/src/server/src/routes/tests.rs

新增 `openai_watermarking_defaults_and_opt_out_reach_engine_core`, 用 12 个组合锁定两条 OpenAI 路由的默认值与 opt-out 语义。

```

// rust/src/server/src/routes/tests.rs
// 覆盖 2 个端点 × stream 开关 × 3 种取值 (省略 / true / false) ,
// 断言最终送到 mock engine-core 的 sampling_params.watermarking == unwrap_or(true),
// 即「省略 -> 默认开启、显式 false -> opt-out」在 HTTP 全链路上成立。
#[tokio::test(flavor = "multi_thread", worker_threads = 2)]
#[serial]
async fn openai_watermarking_defaults_and_opt_out_reach_engine_core() {
    for endpoint in ["/v1/chat/completions", "/v1/completions"] {
        for stream in [false, true] {
            for watermarking in [None, Some(true), Some(false)] {
                let (mut app, engine_task) = test_app_with_backend_and_engine_request_check(
                    Arc::new(FakeChatBackend::new()),
                    move |request| {
                        let params = request.sampling_params.as_ref().expect("sampling params");
                        assert_eq!(params.watermarking, watermarking.unwrap_or(true));
                    },
                );
                .await;

                let mut body = json!({"stream": stream, "max_tokens": 2});
                if endpoint == "/v1/chat/completions" {
                    body["messages"] = json!([{"role": "user", "content": "hi"}]);
                } else {
                    body["prompt"] = json!("hi");
                }
                if let Some(watermarking) = watermarking {
                    body["watermarking"] = json!(watermarking);
                }
                let response = app
                    .call(
                        Request::builder()
                            .method("POST")
                            .uri(endpoint)
                            .header("content-type", "application/json")
                            .body(Body::from(body.to_string()))
                            .expect("build request"),
                    )
                    .await
                    .expect("call app");
                assert_eq!(response.status(), StatusCode::OK);
                to_bytes(response.into_body(), usize::MAX).await.expect("drain response");
            }
        }
    }
}

```

```
        engine_task.await.expect("mock engine task");
    }
}
}
```

评论区精华

BugenZhao: 先评论@codexreview, 随后 APPROVED 并回复 "Thanks@NickLucche!"。

chatgpt-codex-connector[bot]: "Codex Review: Didn't find any major issues. Swish!"
(Reviewed commit: [0dbb72b7c6](#))

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."

NickLucche (作者): "@BugenZhao lemme know if you have concerns, it's just a FE parity PR"。

解读：这是一次「低争议 + 自动化评审」的合并。没有任何 review comment 线程，说明改动被整体视为机械的字段透传；作者主动声明「纯前端对齐」，维护者用 Codex 做了一遍正确性扫描后批准。需要留意的是，fork PR 无法触发 Claude 自动评审，真正起作用的只有 Codex 与人工审查，而 Codex 的结论并未覆盖「引擎侧是否真的停止加水印」这一层。

风险与影响

1. 默认值语义分散在多处：true 分别硬编码在 ChatCompletionRequest / CompletionRequest 的 serde 默认与 Default impl、vllm_text::SamplingParams::default()、default_watermarking() 以及 to_text_request 的 unwrap_or(true)，共五处以上。一旦 Python 侧默认语义调整，HTTP 与 gRPC 可能分叉。
2. 新增 wire 字段的版本混跑兼容性：EngineCoreSamplingParams 现在总会序列化 watermarking 键，gRPC 侧也新增了 proto 字段。跨版本混部（新 Rust 前端 + ID engine-core）时未知 msgpack 键是否被接受、绑定是否需同步刷新，本 PR 未做版本协商或降级路径；这属于需要在真实部署矩阵中验证的点，无法从本次 diff 确认。
3. 测试为 mock 后端，不覆盖真实水印行为：三处新测试只断言「字段到达 engine-core」，FakeChatBackend + mock engine task 无法验证采样阶段水印是否真的被抑制；若 engine-core 侧字段名或语义有偏差，Rust 侧测试仍会全绿，真实行为验证依赖 Python 侧 e2e（见 #56309）。
4. 回退风险低：改动前 Rust 前端丢弃该字段、由 engine-core 使用默认（等价 true），改动后默认同样为 true，存量调用不受影响；Some(false) 是纯新增能力。
5. proto 变更的工程配套：3 行 proto 变更需要重新生成绑定并由 CI 体现，字段号在后续演进中需保持稳定，review 时不宜忽略。

影响范围上，调用方（HTTP 与 gRPC 用户）首次获得按请求关闭水印的能力，默认行为不变、gRPC 新增字段向后兼容；系统层面是 Rust 前端与 engine-core 采样参数契约的接口扩展，lower_sampling_params 的快照更新将该字段纳入回归保护网；团队层面则沉淀出一套可复用的「五段式透传」模板，后续新采样开关可照此复制。

推荐与关联脉络

是否值得精读：中等偏上。负责 Rust 前端或水印特性的工程师值得通读（约 15 分钟）；纯 Python 侧开发者只需了解「Rust 前端已支持按请求 opt-out」这一结论。

值得关注的设计决策：

1. proto3 optional 换三态——用 optional bool 区分「未设置」与「显式 false」，决定了 `unwrap_or(true)` 的写法，是本次唯一有信息量的取舍。
2. serde 默认值函数与 Python `omit_defaults=True` 的耦合——`default_watermarking()` 存在的原因是 msgpack 会丢默认键，这类「跨语言默认值对齐」在本仓库反复出现（#50894、#56791 都在 `omit_defaults` 相关路径上修过问题）。
3. 默认值当前散落多处，是后续值得收敛为单一常量的清理点。

关联脉络：本 PR 是 RFC Issue #53916 前端对齐工作的收尾环节，与同族的 #56309（e2e 检测 + GSM8K 质量回归）互补，合起来才构成「能力可用 + 行为可验证」的闭环；与 [rust/src/text/src/lower.rs](#)、[EngineCoreSamplingParams](#) 的历次改动共享「Rust 前端与 Python 语义对齐」主题；同一个 [grpc/convert.rs](#) 上的前作 #56406（selected-token logprob 模式保留）也属于「转换层不能丢语义」的同类修复，本 PR 延续了该文件对可选字段的谨慎风格。