PR #51447 完整报告

vllm-project/vllm

Bound generation inputs before expensive work

合并时间：2026-08-11 22:03

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51447

执行摘要

- 一句话：请求输入上游加界，封堵 5 类 CPU 放大攻击
- 推荐动作：值得精读：这是一个教科书式的 " 在昂贵工作前绑定输入 " 安全加固案例，对任何暴露公网 JSON API 的服务都有借鉴价值。建议重点看三处设计决策：(1) sampling_params.py 中先去重再 tokenize、边 tokenize 边计数的时序，使上限校验发生在放大点之前；(2) 环境变量配置化（review 推动）让安全默认值与灵活部署并存；(3) DeepSeek 编码器通过参数透传 last_user_idx 把多条消息共享的一次扫描结果复用。若需维护同类服务，本 PR 的 RED-to-GREEN 测试组织方式（单文件合并 + MockTokenizer 计数断言）也值得模仿。

功能与动机

PR body 将问题定位为典型的资源放大（work amplification）：'An authenticated client could consume frontend CPU or memory out of proportion to the request size and slow other users. These paths do not expose data or run code; their impact is reduced service availability.' 并给出量化样例：{"stop":["x", ... 500000 entries ...]} 能让一次普通补全多耗时约 54 秒；50,000 个坏词会在被拒绝前先完成 tokenize；每生成一个 token 都要扫描 50 万条 stop 字符串；DeepSeek 长对话中每条消息都会重新扫描整个历史。对应安全公告 GHSA-4c3x-h2r4-j55f 的 Track B，目标是让请求开销在进入昂贵工作前被限定到常量级。

实现拆解

1. 入口层 stop 列表上限（Python + Rust 双前端）：在 vllm/entrypoints/openai/engine/protocol.py 定义 StopParam 类型别名（str | Annotated[list[str], Field(max_length=envs.VLLM_MAX_STOP_STRINGS)] | None），并依次替换 CompletionRequest、ChatCompletionRequest、BatchChatCompletionRequest（completion / chat_completion 协议文件）与 ResponsesRequest（responses/protocol.py）的 stop 字段。Rust 前端在 rust/src/server/src/routes/openai/utils/types.rs 的 validate_stop 中先检查 stop.len() > max_stop_strings() 再检查空字符串，错误码携带 "at most {n} items" 文案，与 pydantic 错误信息对齐；同时新增 2 个单元测试固定默认 4 条上限。
2. SamplingParams 去重与序列数前置校验：vllm/sampling_params.py 的 __post_init__ 用 list(dict.fromkeys(...)) 对 stop_token_ids 与 bad_words 顺序去重（去重后再进入后续 tokenize 与 stop 匹配）；将 _verify_args 中内联的 n 校验抽取为

_verify_num_sequences(value, parameter_name)，并让 BeamSearchParams 新增 __post_init__ 对 beam_width 走同一校验——保证流式 beam 在 per-choice 状态分配前就拒绝超限值。

3. bad words 工作量绑定：vllm/envs.py 新增 VLLM_MAX_NUM_BAD_WORDS（128）、VLLM_MAX_BAD_WORDS_TOTAL_TOKENS（1024）两个上限；SamplingParams.update_from_tokenizer 在循环内边编码边累计 _bad_words_token_ids，超过上限立即抛 VLLMValidationError，避免先 tokenize 完 5 万个词条再拒绝；GPU worker 的 BadWordsState（vllm/v1/worker/gpu/sample/bad_words.py）改为从 envs 读取上限来分配 bad_word_token_ids 缓冲宽度与 add_request 校验，删除硬编码常量。
4. DeepSeek 编码器单次扫描：vllm/tokenizers/deepseek_v4_encoding.py 与 deepseek_v32_encoding.py 的 render_message 新增可选参数 last_user_idx，encode_messages 在循环外调用一次 find_last_user_index(full_messages) 后逐条传入；render_message 在 last_user_idx is None 时才自行回退扫描，保持单次调用兼容。原先每条消息都反向扫描整个对话，长历史下为 O(N²)。
5. 配置化与合并回归测试：vllm/envs.py 同时新增 VLLM_MAX_STOP_STRINGS（4）；五个分散测试文件合并为 tests/test_request_input_bounds.py（新增 273 行），覆盖 4 类公开请求构造器对 4 条 / 5 条 stop 的接受与拒绝、环境变量覆盖（含子进程验证）、stop_token_ids / bad_words 去重保序、MockTokenizer 计数断言 tokenize 上限（65 词触发 129 次调用后拒绝）、beam_width / n 上限、DeepSeek 单次扫描与渲染保真。

关键文件：

- vllm/sampling_params.py（模块 采样参数；类别 source；类型 core-logic；符号 _verify_num_sequences, SamplingParams.post_init, SamplingParams.update_from_tokenizer, BeamSearchParams.post_init）：核心采样参数逻辑：顺序去重、_verify_num_sequences 抽取、bad_words 边编码边限流，所有请求必经路径
- rust/src/server/src/routes/openai/utils/types.rs（模块 入口校验；类别 source；类型 entrypoint；符号 max_stop_strings, validate_stop）：Rust 前端 stop 校验入口，新增 max_stop_strings() 与数量上限检查，保持与 Python 端默认值一致
- vllm/envs.py（模块 环境配置；类别 source；类型 core-logic）：新增 3 个 VLLM_MAX_* 环境变量，全部上限配置化的基础
- vllm/tokenizers/deepseek_v4_encoding.py（模块 编码器；类别 source；类型 core-logic；符号 render_message, encode_messages）：DeepSeek V4 编码器 last_user_idx 参数透传，消除 O(N²) 历史扫描
- tests/test_request_input_bounds.py（模块 测试；类别 test；类型 test-coverage；符号 _StopRequest, _completion_request, _chat_request, _batch_chat_request）：合并 5 个分散测试文件而来的 273 行回归测试，参数化覆盖 4 类公开请求的 stop 上限、环境变量覆盖、去重保序、tokenize 次数上限与 DeepSeek 单次扫描
- vllm/entrypoints/openai/engine/protocol.py（模块 协议层；类别 source；类型 dependency-wiring；符号 StopParam）：StopParam 类型别名定义，所有公开协议 stop 字段限制的源头

- vllm/v1/worker/gpu/sample/bad_words.py（模块 采样器；类别 source；类型 dependency-wiring；符号 BadWordsState）：GPU worker bad words 缓冲宽度与 add_request 校验改为读取环境变量，删除硬编码常量
- vllm/tokenizers/deepseek_v32_encoding.py（模块 编码器；类别 source；类型 core-logic；符号 render_message, encode_messages）：DeepSeek V3.2 编码器同款单次扫描优化
- vllm/entrypoints/openai/chat_completion/protocol.py（模块 协议层；类别 source；类型 core-logic）：ChatCompletionRequest 与 BatchChatCompletionRequest 接入 StopParam
- vllm/entrypoints/openai/responses/protocol.py（模块 协议层；类别 source；类型 dependency-wiring）：ResponsesRequest 接入 StopParam
- vllm/entrypoints/openai/completion/protocol.py（模块 协议层；类别 source；类型 core-logic）：CompletionRequest 接入 StopParam

关键符号：_verify_num_sequences, SamplingParams.post_init, SamplingParams.update_from_tokenizer, BeamSearchParams.post_init, max_stop_strings, validate_stop, render_message, encode_messages, BadWordsState.add_request

关键源码片段

vllm/sampling_params.py

核心采样参数逻辑：顺序去重、_verify_num_sequences 抽取、bad_words 边编码边限流，所有请求必经路径

```
def _verify_num_sequences(value: int, parameter_name: str) -> None:
    # 统一校验 n 与 beam_width：在按序列分配任何状态（如流式 beam
    # 的 per-choice 缓冲）之前就拒绝超限值，避免先放大再拒绝。
    if not isinstance(value, int):
        raise VLLMValidationError(
            f"{parameter_name} must be an int, but is of type {type(value)}"
        )
    if value < 1:
        raise VLLMValidationError(f"{parameter_name} must be at least 1, got {value}.")
    max_n = envs.VLLM_MAX_N_SEQUENCES
    if value > max_n:
        raise VLLMValidationError(
            f"{parameter_name} must be at most {max_n}, got {value}. "
            "To increase this limit, set the VLLM_MAX_N_SEQUENCES "
            "environment variable."
        )


def __post_init__(self) -> None:
    ...
    if self.stop_token_ids is None:
```

```
            self.stop_token_ids = []
        else:
            # 顺序去重：dict.fromkeys 保留首次出现顺序。stop 判定最终依赖
            # 集合 _all_stop_token_ids，重复项只会徒增匹配扫描量。
            self.stop_token_ids = list(dict.fromkeys(self.stop_token_ids))

        if self.bad_words is None:
            self.bad_words = []
        else:
            # bad_words 同样先去重再进入 tokenize，避免重复词条放大编码开销。
            self.bad_words = list(dict.fromkeys(self.bad_words))
        ...
        self._verify_args()


    def update_from_tokenizer(self, tokenizer: TokenizerLike) -> None:
        if not self.bad_words:
            return
        self._bad_words_token_ids = []
        max_num_bad_words = envs.VLLM_MAX_NUM_BAD_WORDS
        for bad_word in self.bad_words:
            # 每个词条尝试带空格与不带空格两种前缀编码，tokenize 调用
            # 次数约为词条数的 2 倍；边编码边计数，一旦累计超过上限立即
            # 抛错，而不是先 tokenize 全部再统一校验。
            for add_prefix_space in [False, True]:
                prefix = " " if add_prefix_space else ""
                prompt = prefix + bad_word.lstrip()
                prompt_token_ids = tokenizer.encode(
                    text=prompt, add_special_tokens=False
                )
                # 不加空格已能覆盖，或加空格产生不同首个 token 且长度一致
                # 时才追加为新词条（保留原有歧义消解逻辑）。
                if (not add_prefix_space) or (
                    prompt_token_ids[0] != self._bad_words_token_ids[-1][0]
                    and len(prompt_token_ids) == len(self._bad_words_token_ids[-1])
                ):
                    self._bad_words_token_ids.append(prompt_token_ids)
                    if len(self._bad_words_token_ids) > max_num_bad_words:
                        raise VLLMValidationError(
                            f"Too many bad words after tokenization: "
                            f"{len(self._bad_words_token_ids)}. "
                            f"The max number is {max_num_bad_words}.",
                            parameter="bad_words",
                            value=self.bad_words,
                        )
```

rust/src/server/src/routes/openai/utils/types.rs

Rust 前端 stop 校验入口，新增 max_stop_strings() 与数量上限检查，保持与 Python 端默认值一致

```
// 读取环境变量中的 stop 字符串数量上限，默认 4；parse 失败时回退默认值。
fn max_stop_strings() -> usize {
    std::env::var("VLLM_MAX_STOP_STRINGS")
        .ok()
        .and_then(|value| value.parse().ok())
        .unwrap_or(4)
}

/// Validates stop sequences (at most the configured number of non-empty strings).
pub fn validate_stop(stop: &StringOrArray) -> Result<(), validator::ValidationError> {
    let stop = stop.as_slice();
    let max_stop_strings = max_stop_strings();
    // 先做数量检查再检查空字符串：50 万条 stop 的逐条扫描代价远高于
    // 一次长度比较，且需与 Python 端 Field(max_length=...) 默认值对齐。
    if stop.len() > max_stop_strings {
        let mut error = validator::ValidationError::new("too_many_stop_strings");
        error.code = format!("stop strings must contain at most {max_stop_strings} items").into();
        return Err(error);
    }
    if stop.iter().any(|s| s.is_empty()) {
        return Err(validator::ValidationError::new(
            "stop strings cannot be empty",
        ));
    }
    Ok(())
}
```

vllm/tokenizers/deepseek_v4_encoding.py

DeepSeek V4 编码器 last_user_idx 参数透传，消除 O(N²) 历史扫描

```
def render_message(
    index: int,
    messages: List[Dict[str, Any]],
    thinking_mode: str,
    drop_thinking: bool = True,
    reasoning_effort: Optional[str] = None,
    last_user_idx: Optional[int] = None,
) -> str:
    ...
    msg = messages[index]
    # last_user_idx 由 encode_messages 一次性算好后逐条传入；此前
    # 每条消息都反向扫描整个 messages 找最后用户消息，长对话历史下
    # 是 O(N²) 的重复扫描，攻击者可用大量消息放大编码开销。
    last_user_idx = (
        find_last_user_index(messages) if last_user_idx is None else last_user_idx
    )
```

```
    ...


def encode_messages(
    messages: List[Dict[str, Any]],
    thinking_mode: str,
    context: List[Dict[str, Any]] | None = None,
    drop_thinking: bool = True,
    add_default_bos_token: bool = True,
    reasoning_effort: Optional[str] = None,
) -> str:
    ...
    full_messages = context + messages
    ...
    # 整段对话只需扫描一次最后一个 user/developer 消息的位置。
    last_user_idx = find_last_user_index(full_messages)

    for idx in range(num_to_render):
        prompt += render_message(
            idx + context_len,
            full_messages,
            thinking_mode=thinking_mode,
            drop_thinking=effective_drop_thinking,
            reasoning_effort=reasoning_effort,
            last_user_idx=last_user_idx,
        )

    return prompt
```

评论区精华

Review 的核心交锋集中在 " 硬编码常量 vs 环境变量 ": DarkLight1337 在 vllm/entrypoints/openai/engine/protocol.py（MAX_STOP_STRINGS = 4）、vllm/sampling_params.py（MAX_NUM_BAD_WORDS = 128）和 rust 端 types.rs（const MAX_STOP_STRINGS: usize = 4）三处 diff 上连续给出 "Should be an env variable"，最终推动三个 VLLM_MAX_* 环境变量落地；随后又在 vllm/v1/worker/gpu/sample/bad_words.py 追加 "Actually could you also convert this into an env variable along the way?"，把 GPU worker 既有的 MAX_BAD_WORDS_TOTAL_TOKENS 也一并配置化（提交 8d81ecdc）。测试方面 reviewer 要求 "Let's consolidate the test files"，五个分散测试文件被合并为单个 tests/test_request_input_bounds.py。合并后作者主动说明 CI 伞形状态因历史 flaky 任务未绿，DarkLight1337 追问 entrypoints-integration-api-server-openai-part 是否真为 flaky，复跑后通过，最终 APPROVED。

- MAX_STOP_STRINGS 硬编码应改为环境变量 (design): 新增 VLLM_MAX_STOP_STRINGS 环境变量，Python 端 Field(max_length=...) 与 Rust 端 max_stop_strings() 均读取该变量

- MAX_NUM_BAD_WORDS 与 MAX_BAD_WORDS_TOTAL_TOKENS 配置化 (design): 新增 VLLM_MAX_NUM_BAD_WORDS 与 VLLM_MAX_BAD_WORDS_TOTAL_TOKENS，并在 GPU worker 中统一从 envs 读取
- Rust 前端 stop 上限与 Python 保持一致 (design): Rust 端新增 max_stop_strings() 读取同一环境变量，默认 4，并补充接受 / 拒绝两个单元测试
- 测试文件合并 (testing): 五个分散测试文件合并为单个 tests/test_request_input_bounds.py（273 行）
- CI 伞形状态与 flaky 测试确认 (question): 复跑后通过（'Ok it passes now...'），PR 获得 APPROVED

风险与影响

- 风险： 1) 公共 API 行为变更：stop 列表超过 4 个字符串的既有客户端将收到 422 ValidationError（Rust 端同样拒绝），bad_words 超过 128 词条或 1024 token 的请求会被拒。依赖大列表的存量用户必须显式设置 VLLM_MAX_STOP_STRINGS 等环境变量，属于兼容性 breaking change。 2) Python / Rust 双端一致性：Python 端 StopParam 走 pydantic Field(max_length=envs.VLLM_MAX_STOP_STRINGS)，Rust 端 validate_stop 直接读 std::env::var；同一环境变量在两端行为需保持一致（Rust 无 env 缓存即刻生效，Python envs 模块带缓存，测试中可见需要 cache_clear），未来若某端改动默认值会悄悄分叉。 3) 行为语义变化：SamplingParams.__post_init__对 stop_token_ids 与 bad_words 的去重可能改变极少数依赖重复项透传的逻辑（重复项本无语义，风险低）；DeepSeek 渲染改动涉及 prompt 保真，测试 test_encode_messages_preserves_small_chat_prompt 提供了回归保护。 4) 遗漏入口风险：本次仅覆盖 completion / chat / batch chat / responses 四个公开入口的 stop 字段与 SamplingParams 内部路径，其他可能携带同类数组的入口（如 tools、logit_bias）不在本次范围。
  - 影响：影响面覆盖所有公开生成入口（OpenAI completion、chat completion、batch chat、responses API）以及 Rust 高并发前端；单请求最坏前端开销从数十秒量级降至常量级，封堵了未经鉴权客户端拖慢其他用户的 DoS 路径。对内新增 3 个 VLLM_MAX_* 可调环境变量，运维可按部署形态调大或调小；测试从 5 个分散文件收敛为 1 个合并文件（273 行），后续回归维护更集中。DeepSeek 编码器改动同时带来长历史对话编码的真实性能收益（O(N²) → O(N)）。对内部 API（EngineCore、离线接口）无破坏。
  - 风险标记：公共 API 行为变更，Python/Rust 双端一致性，核心路径变更，安全修复，环境变量配置

关联脉络

- PR #51774 [Perf] Avoid repeated multimodal prompt update scans: 同一类 " 避免重复扫描 " 的限流优化模式：多模态 prompt 更新扫描与 DeepSeek 历史扫描都从重复 O(N) 收敛为单次，可互为参照
- PR #51556 [Bugfix][Frontend] Report Cohere stop sequences correctly: 同属 stop 序列处理链路的前端正确性修复，说明 stop 语义是前端持续维护的边界热点

- PR #51654 Fix chat completion 500 on non-object JSON bodies: 同属前端输入校验健壮性强化，与本次在入口处收紧请求边界的思路一致