PR #42052 完整报告

vllm-project/vllm

[Frontend] Return rendered prompt text in chat completion response

合并时间：2026-05-11 13:53

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/42052

执行摘要

- 一句话：新增 prompt_text 字段返回聊天模板渲染文本
- 推荐动作：该 PR 设计简洁，值得关注其参数设计思路和直接利用引擎内部数据的做法。推荐在类似场景（如需要暴露内部处理结果）时参考此模式。

功能与动机

关联 Issue #42051 指出，用户在调试时需要查看聊天模板处理后的具体 prompt 文本，而 vLLM 仅能输出 prompt token IDs。SGLang 已支持此功能，因此 vLLM 需添加类似能力，便于开发者准确了解模型输入。

实现拆解

1. 协议定义扩展：在 vllm/entrypoints/openai/chat_completion/protocol.py 中为 ChatCompletionRequest 添加 return_prompt_text 布尔参数，为 ChatCompletionResponse 和 ChatCompletionStreamResponse 添加 prompt_text: str | None 字段。
2. 流式响应生成：在 vllm/entrypoints/openai/chat_completion/serving.py 的 chat_completion_stream_generator 中，首次迭代时从 res.prompt 获取渲染后的 prompt 文本（仅当 request.return_prompt_text 为真），并设置到 ChatCompletionStreamResponse 的 prompt_text 字段。
3. 非流式响应生成：在同一文件的 chat_completion_full_generator 中，从 final_res.prompt 获取文本并设置到 ChatCompletionResponse 的 prompt_text 字段。
4. 利用已有数据：直接使用引擎内部已存储的 prompt（res.prompt），无需通过 tokenizer.decode() 转换，避免阻塞事件循环。

关键文件：

- vllm/entrypoints/openai/chat_completion/protocol.py（模块 协议层；类别 source；类型 core-logic；符号 ChatCompletionResponse, ChatCompletionStreamResponse, ChatCompletionRequest）：定义了 return_prompt_text 请求参数和 prompt_text 响应字段，是 API 规范变更的核心文件。
- vllm/entrypoints/openai/chat_completion/serving.py（模块 服务逻辑；类别 source；类型 core-logic；符号 chat_completion_stream_generator, chat_completion_full_generator）：实现在流式和非流式生成器中从推理结果提取 prompt

文本并设置到响应对象。

关键符号：chat_completion_stream_generator, chat_completion_full_generator

关键源码片段

vllm/entrypoints/openai/chat_completion/protocol.py

定义了return_prompt_text请求参数和prompt_text响应字段，是API规范变更的核心文件。

```
class ChatCompletionResponse(OpenAIBaseModel):
    # ... 已有字段 ...
    # vLLM-specific fields
    prompt_logprobs: list[dict[int, Logprob] | None] | None = None
    prompt_token_ids: list[int] | None = None
    # Rendered prompt text from chat templating (only set when
    # ``return_prompt_text=True`` on the request).
    prompt_text: str | None = None
    kv_transfer_params: dict[str, Any] | None = Field(
        default=None, description="KVTransfer parameters.")

class ChatCompletionStreamResponse(OpenAIBaseModel):
    # ... 已有字段 ...
    system_fingerprint: str | None = None
    prompt_token_ids: list[int] | None = None
    # Rendered prompt text from chat templating (only set when
    # ``return_prompt_text=True`` on the request); only sent on the first chunk.
    prompt_text: str | None = None

# In ChatCompletionRequest:
    return_token_ids: bool | None = Field(
        default=None,
        description="If specified, the result will include token IDs alongside the generated text...")
    return_prompt_text: bool | None = Field(
        default=None,
        description="If true, the response will include ``prompt_text`` containing the prompt
        string produced by chat templating...")
```

vllm/entrypoints/openai/chat_completion/serving.py

实现在流式和非流式生成器中从推理结果提取 prompt 文本并设置到响应对象。

```
# Inside chat_completion_stream_generator, under first_iteration
role = self.get_chat_request_role(request)
# ``res.prompt`` is the rendered chat-templated prompt
prompt_text = res.prompt if request.return_prompt_text else None

for i in range(num_choices):
    choice_data = ChatCompletionResponseStreamChoice(
        index=i,
        delta=DeltaMessage(role=role, content=""),
        logprobs=None,
```

```
            finish_reason=None,
        )
        chunk = ChatCompletionStreamResponse(
            id=request_id,
            object=chunk_object_type,
            created=created_time,
            choices=[choice_data],
            model=model_name,
            prompt_token_ids=(
                res.prompt_token_ids if request.return_token_ids else None
            ),
            prompt_text=prompt_text,
        )
        # ... usage, yield ...

    # Inside chat_completion_full_generator
    prompt_text = final_res.prompt if request.return_prompt_text else None
    response = ChatCompletionResponse(
        id=request_id,
        created=created_time,
        model=model_name,
        choices=choices,
        usage=usage,
        prompt_logprobs=prompt_logprobs,
        prompt_token_ids=(
            final_res.prompt_token_ids if request.return_token_ids else None
        ),
        prompt_text=prompt_text,
        kv_transfer_params=final_res.kv_transfer_params,
        prompt_routed_experts=prompt_routed_experts,
    )
```

评论区精华

1. 参数命名设计：DarkLight1337 建议不要复用 return_token_ids 来控制 prompt_text，而是引入独立参数 return_prompt_text，作者采纳并实现。
2. 数据来源优化：chaunceyjiang 指出 prompt 文本已存储在 engine_inputs 中，无需重新 decode token IDs，作者改为直接使用 res.prompt，减少开销。
3. 事件循环阻塞提示：gemini-code-assist[bot] 提醒同步 decode 可能阻塞，但后续迭代已使用存储文本，该问题自然消除。

- 参数名称设计：return_prompt_text (design): 独立参数 return_prompt_text 被采纳。
- 使用引擎已存储的 prompt 文本 (performance): 改为直接使用 res.prompt 获取渲染后的 prompt 文本，避免额外解码开销。
- 同步解码可能阻塞事件循环 (performance): 已通过使用存储文本自动解决。

风险与影响

- 风险：新增字段和参数属于向后兼容扩展，旧客户端不发送 return_prompt_text 时无影响。响应模型增加可选字段，不会破坏现有解析。流式模式下仅首块包含 prompt_text，需确保客户端正确处理。整体风险低。
- 影响：对用户：提供调试便利性，可查看模型真实输入文本。对系统：几乎无性能影响，因直接使用已存储的 prompt。影响范围限于 chat completion 端点，不涉及其他 API。
- 风险标记：新增字段，流式兼容性

关联脉络

- 暂无明显关联 PR