PR #23981 完整报告

sgl-project/sglang

Add ChatCompletionRequest-style support to /v1/tokenize

合并时间：2026-05-07 09:35

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/23981

执行摘要

- 一句话：支持 ChatCompletion 风格的 /v1/tokenize
- 推荐动作：值得精读，尤其是 _tokenize_chat_request 的复用模式和 TokenizeRequest 的模型设计。展示了如何通过组合已有的 chat serving 能力来扩展简单端点，是 API 演进的良好范例。

功能与动机

构建 cache-aware 系统时需获取 chat template 渲染后的实际 token 序列，而 /v1/tokenize 之前只支持原始字符串，无法与 /v1/chat/completions 对齐。此改动使 /v1/tokenize 能接受 messages 格式输入，返回与聊天路径一致的 token IDs。

实现拆解

1. 扩展 TokenizeRequest 数据模型（protocol.py）：添加 messages、tools、tool_choice 等字段，设置 model_config = ConfigDict(extra="allow") 以透传未知字段；添加 validate_tokenize_input 模型验证器强制 prompt 和 messages 二选一；新增 to_chat_completion_request 方法将自身转换为 ChatCompletionRequest，实现字段映射。
2. 改造 OpenAIServingTokenize 类（serving_tokenize.py）：在 __init__ 中条件创建 OpenAIServingChat 实例（依赖 template_manager）；修改 _handle_non_streaming_request 优先判断 request.messages，若存在则调用新增的 _tokenize_chat_request 方法；_tokenize_chat_request 复用 chat serving 的 _validate_request 和 _process_messages 链，获得 prompt_ids 后返回。
3. 注册服务时注入 template_manager（http_server.py）：将 OpenAIServingTokenize 构造参数从单参数改为传入 template_manager，确保 chat template 可用。
4. 添加集成测试（test_srt_endpoint.py）：新增 test_openai_tokenize_chat_messages，验证纯消息、带工具、工具选择为 none 三种场景下的 token 结果，并与 apply_chat_template 直接调用的结果对比。

关键文件：

- python/sglang/srt/entrypoints/openai/serving_tokenize.py（模块 API 入口；类别 source；类型 dependency-wiring；符号 init, _tokenize_chat_request）：核心实现：新增 __init__初始化 chat_serving，修改 _handle_non_streaming_request 支持 messages 输入，新增 _tokenize_chat_request 方法复用 chat 处理逻辑。

- python/sglang/srt/entrypoints/openai/protocol.py（模块 数据模型；类别 source；类型 core-logic；符号 validate_tokenize_input, to_chat_completion_request）：数据模型扩展：TokenizeRequest 新增 messages、tools 等字段，添加 validate_tokenize_input 验证器和 to_chat_completion_request 转换方法。
- test/registered/core/test_srt_endpoint.py（模块 测试；类别 test；类型 test-coverage；符号 test_openai_tokenize_chat_messages）：集成测试：test_openai_tokenize_chat_messages 验证 messages、tools 和 tool_choice 场景的正确性。
- python/sglang/srt/entrypoints/http_server.py（模块 入口路由；类别 source；类型 core-logic）：注册时传递 template_manager，使 OpenAIServingTokenize 能获取 chat template。

关键符号：_tokenize_chat_request, validate_tokenize_input, to_chat_completion_request

关键源码片段

python/sglang/srt/entrypoints/openai/serving_tokenize.py

核心实现：新增 __init__初始化 chat_serving，修改 _handle_non_streaming_request 支持 messages 输入，新增 _tokenize_chat_request 方法复用 chat 处理逻辑。

```
def _tokenize_chat_request(self, request: TokenizeRequest) -> List[int]:
    # 确保 template_manager 已传入，否则无法使用 chat template
    if self.chat_serving is None:
        raise ValueError("Chat template tokenization requires a template manager.")

    # 将 TokenizeRequest 转换为 ChatCompletionRequest 以复用 chat 处理链
    chat_request = request.to_chat_completion_request()
    # 复用 OpenAIServingChat 的请求验证（如 tool_choice 有效性）
    validation_error = self.chat_serving._validate_request(chat_request)
    if validation_error:
        raise ValueError(validation_error)

    # 判断是否多模态，以此控制流程
    is_multimodal = self.tokenizer_manager.model_config.is_multimodal
    # 调用 chat serving 的消息处理，得到 prompt_ids
    processed_messages = self.chat_serving._process_messages(
        chat_request, is_multimodal
    )

    prompt_ids = processed_messages.prompt_ids
    # 如果 prompt_ids 是非空列表，直接返回（已 tokenized）
    if isinstance(prompt_ids, list) and (prompt_ids or not processed_messages.prompt):
        return prompt_ids
    # 如果是字符串，需要再编码一次
    if isinstance(prompt_ids, str):
        return self.tokenizer_manager.tokenizer.encode(
            prompt_ids, add_special_tokens=False
```

```
            )
        # 否则从 prompt 文本编码
        if processed_messages.prompt:
            return self.tokenizer_manager.tokenizer.encode(
                processed_messages.prompt, add_special_tokens=False
            )

        raise ValueError("Failed to render chat messages into token ids.")
```

python/sglang/srt/entrypoints/openai/protocol.py

数据模型扩展：TokenizeRequest 新增 messages、tools 等字段，添加 validate_tokenize_input 验证器和 to_chat_completion_request 转换方法。

```
    # 允许接受额外的字段（如 tools, tool_choice），以便透传给 ChatCompletionRequest
    model_config = ConfigDict(extra="allow")

    model: str = DEFAULT_MODEL_NAME
    # prompt 和 messages 二选一，默认 None
    prompt: Optional[Union[str, List[str]]] = None
    messages: Optional[List[ChatCompletionMessageParam]] = None
    tools: Optional[List[Tool]] = Field(default=None, examples=[None])
    tool_choice: Optional[Union[ToolChoice, Literal["auto", "required", "none"]]] = (
        Field(default=None, examples=["auto"])
    )
    reasoning_effort: Optional[Literal["none", "low", "medium", "high"]] = None
    continue_final_message: bool = False
    chat_template_kwargs: Optional[Dict] = None
    add_special_tokens: bool = Field(
        default=True,
        description="whether to add model-specific special tokens (e.g. BOS/EOS) during encoding.",
    )

    @model_validator(mode="after")
    def validate_tokenize_input(self) -> "TokenizeRequest":
        # 严格确保仅设置 prompt 或 messages 中的一个
        if (self.prompt is None) == (self.messages is None):
            raise ValueError("Exactly one of 'prompt' or 'messages' must be provided.")
        return self

    def to_chat_completion_request(self) -> ChatCompletionRequest:
        # 排除 prompt 和 add_special_tokens，其余字段（包括 messages, tools 等）透传
        data = self.model_dump(
            exclude={"prompt", "add_special_tokens"},
            exclude_none=True,
        )
        # 保留 pydantic extra 字段，允许用户传入未显式声明的参数
        extra = getattr(self, "__pydantic_extra__", None)
        if extra:
            data.update(extra)
```

```
    return ChatCompletionRequest.model_validate(data)
```

评论区精华

- 外部集成：doujiang24 评论称他们正在使用此 API 与 dynamo kvindexer 集成，实现精确的 *cache-aware* 路由。
- 内部协作：stmatengss 回应称 SGLang 的 KV *cache* 事件发射可与该 tokenize API 结合，用于基于前缀匹配的路由。
- 测试建议：ShangmingCai 提出“是否应添加测试”，随后作者在第四个提交中补充了 test_openai_tokenize_chat_messages 单元测试。
  - 集成到 dynamo kvindexer 实现 *cache-aware* 路由 (design): 社区认可此 API 对 *cache-aware* 系统的价值，是正向集成。
  - 是否需要为这个 API 添加测试 (testing): 作者在第四个提交中添加了 test_openai_tokenize_chat_messages 测试。

风险与影响

- 风险：
  - 向后兼容性：TokenizeRequest.prompt 从必填改为可选，但通过 model_validator 确保必须提供 prompt 或 messages 其中之一；仅使用 prompt 的老客户端不受影响。
  - 稳定性：_tokenize_chat_request 依赖 OpenAIServingChat 的内部方法（_validate_request、_process_messages），若这些方法未来重构，需同步更新。
  - 异常处理：新增的 ValueError 捕获使 *chat template* 错误时返回合理的错误响应而非 500。
- 影响：
  - 用户：开发者和下游系统（如 Dynamo）现可通过 /v1/tokenize 获取 *chat template* 渲染后的 token 序列，便于构建 *cache-aware* 路由、日志审计或 token 计数。
  - 系统：无性能影响，token 化仍使用原有 tokenizer；内存占用略有增加（缓存 OpenAIServingChat 实例）。
  - 团队：此 PR 为 *cache-aware* 路由和 KV 事件发射的基础设施铺平了道路，后续工作（如 #23391 SWA HiCache）可在此基础上实现精准前缀匹配。
- 风险标记：向后兼容风险，依赖 *chat serving* 内部方法，数据模型可扩展性风险

关联脉络

- 暂无明显关联 PR