PR #45173 完整报告

vllm-project/vllm

Added real /v1/embeddings support for messages + chat_template_kw

合并时间：2026-06-15 09:08

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45173

执行摘要

- 一句话：为 /v1/embeddings 添加消息形状输入与 chat_template_kwargs 支持
- 推荐动作：建议阅读该 PR 以了解 vLLM Embedding 端点如何扩展以支持聊天模板输入，特别是请求模型的设计模式（model_validator）和 IO Processor 的分支处理。对于需要接入类似嵌入模型的开发者具有参考价值。

功能与动机

支持需要聊天模板渲染和 chat_template_kwargs 的嵌入模型（如指令式嵌入提示）。之前 vLLM 的 /v1/embeddings 端点只接受字符串 /token 输入，消息形状的输入在请求验证阶段被拒绝，且 chat_template_kwargs 无法传递到渲染器。该 PR 填补了这一空白，使 embeddings 请求能够像聊天补全请求一样使用聊天模板。

实现拆解

1. 新增协议模型：在 vllm/entrypoints/pooling/embed/protocol.py 中添加 EmbeddingChatInputRequest、EmbeddingBatchChatRequest、EmbeddingBatchChatInputRequest 以及辅助类型判断函数 _is_chat_message、_is_chat_messages、_is_batched_chat_messages。通过 model_validator 将 input 中的消息映射到 messages 字段，复用已有的聊天模板渲染路径。
2. 拆分基础协议：在 vllm/entrypoints/pooling/base/protocol.py 中将 ChatRequestMixin 拆分为 ChatRequestOptionsMixin（仅包含选项字段）和继承它的 ChatRequestMixin（增加 messages 字段），使得 EmbeddingBatchChatRequest 等无 messages 字段的类也能复用选项。
3. 扩展 IO Processor：在 vllm/entrypoints/pooling/embed/io_processor.py 的 pre_process_online 方法中新增对 EmbeddingChatRequest 及其子类的处理分支，调用新增的 _pre_process_openai_chat_online 方法，该方法通过 _batch_render_openai_chat 调用渲染器生成 EngineInput。
4. 更新类型别名：在 vllm/entrypoints/pooling/typing.py 中将新请求类加入 PoolingChatLikeRequest 和 AnyPoolingRequest 联合类型，确保请求解析能正确路由。
5. 测试覆盖：在 tests/entrypoints/pooling/embed/test_io_processor.py 中新增 TestEmbeddingRequestParsing 和 TestPreProcessOpenAIEmbeddingChatOnline 两个测试类，涵盖消息形状、批量消息、token IDs 等输入的形状解析以及预处理流程。

关键文件：

- vllm/entrypoints/pooling/embed/protocol.py（模块 嵌入协议；类别 source；类型 core-logic；符号 _is_chat_message, _is_chat_messages, _is_batched_chat_messages, EmbeddingBatchChatRequest）：核心变更，新增请求模型和输入类型判断函数
- vllm/entrypoints/pooling/embed/io_processor.py（模块 嵌入处理；类别 source；类型 core-logic；符号 _pre_process_openai_chat_online, _batch_render_openai_chat）：新增 chat 嵌入预处理流程
- tests/entrypoints/pooling/embed/test_io_processor.py（模块 测试；类别 test；类型 test-coverage；符号 TestEmbeddingRequestParsing, test_input_messages_parses_as_chat_request, test_batched_input_messages_parses_as_batch_chat_input_request, test_token_ids_still_parse_as_completion_request）：单元测试覆盖解析和预处理
- vllm/entrypoints/pooling/base/protocol.py（模块 基础协议；类别 source；类型 core-logic；符号 ChatRequestMixin, ChatRequestOptionsMixin）：重构 ChatRequestMixin 为 ChatRequestOptionsMixin 和 ChatRequestMixin，支持无 messages 字段的嵌入请求
- vllm/entrypoints/pooling/typing.py（模块 类型；类别 source；类型 core-logic）：更新类型别名以包含新请求类型

关键符号：_is_chat_message, _is_chat_messages, _is_batched_chat_messages, EmbeddingChatInputRequest.normalize_input_messages, EmbeddingBatchChatInputRequest.normalize_input_messages, EmbedIOProcessor._pre_process_openai_chat_online, EmbedIOProcessor._batch_render_openai_chat

关键源码片段

vllm/entrypoints/pooling/embed/protocol.py

核心变更，新增请求模型和输入类型判断函数

```
# vllm/entrypoints/pooling/embed/protocol.py

# 用于判断单个值是否为聊天消息（dict 且含有 "role" 字段）
def _is_chat_message(value: Any) -> bool:
    return isinstance(value, dict) and isinstance(value.get("role"), str)

# 用于判断是否为聊天消息列表
def _is_chat_messages(value: Any) -> bool:
    return (
        isinstance(value, list)
        and bool(value)
        and all(_is_chat_message(item) for item in value)
    )

# 用于判断是否为批量的聊天消息列表
```

```
def _is_batched_chat_messages(value: Any) -> bool:
    return (
        isinstance(value, list)
        and bool(value)
        and all(_is_chat_messages(item) for item in value)
    )

# 单条对话输入（input 字段为消息列表）的嵌入请求
class EmbeddingChatInputRequest(EmbeddingChatRequest):
    """OpenAI embeddings request with one chat conversation in ``input``."""

    input: list[ChatCompletionMessageParam]

    @model_validator(mode="before")
    @classmethod
    def normalize_input_messages(cls, data):
        # 只处理 dict 且不含 messages 字段但 input 是聊天消息的情况
        if not isinstance(data, dict):
            return data
        if "messages" in data or "input" not in data:
            return data
        input_data = data["input"]
        if not _is_chat_messages(input_data):
            return data
        # 将 input 复制到 messages 字段，使父类 EmbeddingChatRequest 能正常处理
        normalized = dict(data)
        normalized["messages"] = input_data
        return normalized

# 批量对话输入（input 为 batched 消息）的嵌入请求
class EmbeddingBatchChatInputRequest(EmbeddingBatchChatRequest):
    input: list[Annotated[list[ChatCompletionMessageParam], Field(min_length=1)]] = (
        Field(..., min_length=1)
    )

    @model_validator(mode="before")
    @classmethod
    def normalize_input_messages(cls, data):
        if not isinstance(data, dict):
            return data
        if "messages" in data or "input" not in data:
            return data
        input_data = data["input"]
        if not _is_batched_chat_messages(input_data):
            return data
        normalized = dict(data)
        normalized["messages"] = input_data
        return normalized
```

vllm/entrypoints/pooling/embed/io_processor.py

新增 chat 嵌入预处理流程

```
# vllm/entrypoints/pooling/embed/io_processor.py

def _pre_process_openai_chat_online(
    self,
    ctx: PoolingServeContext[
        EmbeddingChatRequest
        | EmbeddingBatchChatRequest
        | EmbeddingChatInputRequest
        | EmbeddingBatchChatInputRequest
    ],
) -> None:
    request = ctx.request
    # 验证聊天模板相关配置
    self._validate_chat_template(
        request_chat_template=request.chat_template,
        chat_template_kwargs=request.chat_template_kwargs,
        trust_request_chat_template=self.trust_request_chat_template,
    )

    # 批量请求的消息已经是 list[list]，单条则包装为 list
    if isinstance(
        request, (EmbeddingBatchChatRequest, EmbeddingBatchChatInputRequest)
    ):
        all_messages = request.messages
    else:
        all_messages = [request.messages]

    # 渲染所有对话并生成 EngineInput
    ctx.engine_inputs = self._batch_render_openai_chat(request, all_messages)


def _batch_render_openai_chat(
    self,
    request: (
        EmbeddingChatRequest
        | EmbeddingBatchChatRequest
        | EmbeddingChatInputRequest
        | EmbeddingBatchChatInputRequest
    ),
    all_messages: Sequence[list[ChatCompletionMessageParam]],
) -> list[EngineInput]:
    renderer = self.renderer
    mm_config = self.model_config.multimodal_config

    tok_params = request.build_tok_params(self.model_config)
    chat_params = request.build_chat_params(
```

```
            self.chat_template,
            self.chat_template_content_format,
        ).with_defaults(
            merge_kwargs(
                None,
                dict(
                    tools=None,
                    tokenize=is_mistral_tokenizer(renderer.tokenizer),
                ),
            ),
            default_media_io_kwargs=(mm_config.media_io_kwargs if mm_config else None),
        )

        # 调用底层 render_chat 进行渲染
        _, engine_inputs = renderer.render_chat(
            all_messages,
            chat_params,
            tok_params,
            prompt_extras={
                k: v
                for k in ("mm_processor_kwargs", "cache_salt")
                if (v := getattr(request, k, None)) is not None
            },
        )
        return engine_inputs
```

tests/entrypoints/pooling/embed/test_io_processor.py

单元测试覆盖解析和预处理

```
# tests/entrypoints/pooling/embed/test_io_processor.py

class TestEmbeddingRequestParsing:
    """Unit tests for OpenAI embedding request parsing."""

    def test_input_messages_parses_as_chat_request(self):
        # input 字段为单条消息列表时，应解析为 EmbeddingChatInputRequest
        request = TypeAdapter(EmbeddingRequest).validate_python(
            {
                "model": "test",
                "input": [{"role": "user", "content": "hello"}],
                "chat_template_kwargs": {"instruction": "Represent the query: "},
            }
        )
        assert isinstance(request, EmbeddingChatInputRequest)
        assert request.input == [{"role": "user", "content": "hello"}]
        assert request.messages == [{"role": "user", "content": "hello"}]
        assert request.chat_template_kwargs == {"instruction": "Represent the query: "}

    def test_batched_input_messages_parses_as_batch_chat_input_request(self):
```

```
    # input 字段为批量消息时，应解析为 EmbeddingBatchChatInputRequest
    request = TypeAdapter(EmbeddingRequest).validate_python(
        {
            "model": "test",
            "input": [
                [{"role": "user", "content": "hello"}],
                [{"role": "user", "content": "goodbye"}],
            ],
            "chat_template_kwargs": {"instruction": "Represent the query: "},
        }
    )
    assert isinstance(request, EmbeddingBatchChatInputRequest)
    assert request.input == [
        [{"role": "user", "content": "hello"}],
        [{"role": "user", "content": "goodbye"}],
    ]
    assert request.messages == [
        [{"role": "user", "content": "hello"}],
        [{"role": "user", "content": "goodbye"}],
    ]
    assert request.chat_template_kwargs == {"instruction": "Represent the query: "}

def test_token_ids_still_parse_as_completion_request(self):
    # token IDs 输入保持不变，应解析为 EmbeddingCompletionRequest
    request = TypeAdapter(EmbeddingRequest).validate_python(
        {"model": "test", "input": [[1, 2, 3], [4, 5]]}
    )
    assert isinstance(request, EmbeddingCompletionRequest)
    assert request.input == [[1, 2, 3], [4, 5]]

def test_messages_still_parses_as_chat_request(self):
    # 使用 messages 字段（原 API）仍解析为 EmbeddingChatRequest
    request = TypeAdapter(EmbeddingRequest).validate_python(
        {
            "model": "test",
            "messages": [{"role": "user", "content": "hello"}],
            "chat_template_kwargs": {"instruction": "Represent the query: "},
        }
    )
    assert isinstance(request, EmbeddingChatRequest)
    assert request.messages == [{"role": "user", "content": "hello"}]
    assert request.chat_template_kwargs == {"instruction": "Represent the query: "}

def test_batched_messages_parses_as_batch_chat_request(self):
    # 批量 messages 字段解析为 EmbeddingBatchChatRequest
    request = TypeAdapter(EmbeddingRequest).validate_python(
        {
            "model": "test",
            "messages": [
```

```
                [{"role": "user", "content": "hello"}],
                [{"role": "user", "content": "goodbye"}],
            ],
            "chat_template_kwargs": {"instruction": "Represent the query: "},
        }
    )
    assert isinstance(request, EmbeddingBatchChatRequest)
    assert request.messages == [
        [{"role": "user", "content": "hello"}],
        [{"role": "user", "content": "goodbye"}],
    ]
    assert request.chat_template_kwargs == {"instruction": "Represent the query: "}
```

评论区精华

Review 中的主要讨论集中在请求类的设计上：

- noooop最初建议将消息形状输入的逻辑放在独立请求类中，而非直接在 EmbeddingChatRequest 中处理。作者采纳并创建了 EmbeddingChatInputRequest。
- noooop还询问新类与 BatchChatCompletionRequest 的关系及是否存在 messages_batch 字段。作者澄清后改为使用与 BatchChatCompletionRequest 一致的消息列表形状，即 messages 字段存储批量对话，避免了引入不规范的 messages_batch 字段。
- 最终 yewentao256在 B300 上验证通过并给予 LGTM，noooop也批准了该 PR。
- 消息形状输入应使用独立请求类 (design): 作者采纳，创建了 EmbeddingChatInputRequest，使用 model_validator 将 input 映射到 messages。
- 新请求类与 BatchChatCompletionRequest 的关系 (design): 作者澄清后移除 messages_batch，改用与 BatchChatCompletionRequest 一致的消息列表形状（messages 字段存储批量对话）。

风险与影响

- 风险：兼容性风险：现有使用字符串或 token 列表输入的请求不受影响，类型分支判断确保了向后兼容。 性能风险：新增渲染路径仅对消息形状的请求触发，与已有的 Cohere 路径类似，额外开销可忽略。 回归风险：新代码与已有预处理路径（Cohere、Completions）通过 isinstance 分支隔离，测试覆盖了多种输入形状，降低了回归概率。 安全风险：chat_template_kwargs 由用户传入，但基础设施已有验证机制（_validate_chat_template），未引入额外风险。
- 影响：用户影响：现在可以向 /v1/embeddings 发送聊天消息格式的输入，并指定 chat_template_kwargs，用于需要指令式提示的嵌入模型（如 Qwen3-Embedding）。 系统影响：EmbedIOProcessor.pre_process_online 方法新增了一个 isinstance 分支，请求处理路径中增加了渲染步骤，但整体流程清晰。 团队影响：需要维护新增的请求模型和相应的测试用例，但设计上复用了已有的 ChatRequestMixin 和渲染器，代码耦合度低。
- 风险标记：请求模型增加，渲染路径新增，基础协议重构

关联脉络

- 暂无明显关联 PR