3. AI Large Model Voice Interaction

1. Concept Introduction

1.1 What is "AI Large Model Voice Interaction"?

In the `largemodel` project, **AI Large Model Voice Interaction** connects the previously introduced **offline ASR** and **offline TTS** with the core of the **Large Language Model (LLM)**, forming a complete dialogue system that can hear, speak, and think.

This is no longer an isolated function, but a prototype of a true **voice assistant**. Users can engage in natural language conversations with the robot via voice, and the robot can understand questions, think about answers, and respond with voice. The entire process is completed locally, without a network.

The core of this function is the `model_service` **ROS2 node**. It acts as the brain and neural center, subscribing to the ASR recognition results, calling the LLM for thinking, and then publishing the LLM's text response to the TTS node for speech synthesis.

1.2 Implementation Principle Overview

The implementation principle of this function is a classic data flow pipeline:

1. **Audio -> Text (ASR)**: The `asr` node continuously listens for ambient sound. Once it detects a user uttering a sentence, it converts it into text and publishes it to the `/asr_text` topic.
2. **Text -> Thought -> Text (LLM)**: The `model_service` node subscribes to the `/asr_text` topic. When it receives text from the ASR, it sends it as a prompt to the locally deployed large language model (such as `Qwen` running via `Ollama`). The LLM generates a response text based on the context.
3. **Text -> Audio (TTS)**: After receiving the LLM's response, the `model_service` node publishes it to the `/tts_text` topic.
4. **Audio Playback:** The `tts_only` node subscribes to the `/tts_text` topic. Upon receiving text, it immediately calls the offline TTS model to synthesize it into audio and plays it through the speaker.

This process forms a complete closed loop of **speech input -> text processing -> text output -> speech output**.

---

2. Code analysis

Key code

1. Voice input node (`largemodel/largemodel/asr.py`)

```
# From largemodel/largemodel/asr.py
class ASRNode(Node):
    def __init__(self):
        # ...
        self.asr_pub = self.create_publisher(String, "asr", 5)
        # ...

    def kws_handler(self)->None:
        if self.listen_for_speech(self.mic_index):
            asr_text = self.ASR_conversion(self.user_speechdir)
            if asr_text != 'error':
                self.asr_pub_result(asr_text)

    def asr_pub_result(self,asr_result:str)->None:
        msg=String(data=asr_result)
        self.asr_pub.publish(msg)
```

2. AI service and speech output node (`largemodel/largemodel/model_service.py`)

```
# From largemodel/largemodel/model_service.py
class LargeModelService(Node):
    def __init__(self):
        # ...
        self.asrsub = self.create_subscription(String,'asr',
self.asr_callback,1)
        # ...

    def asr_callback(self,msg):
        """Callback function for handling ASR messages. """
        # ...
        result = self.model_client.infer_with_text(msg.data,
message=messages_to_use)
        self.process_model_result(result)

    def process_model_result(self, result, from_seewhat=False):
        """Process the result returned by the model. """
        # ...
        user_friendly_response = response_json.get("response", "我正在处理...")
        # ...
        self._safe_play_audio(user_friendly_response)
        # ...
        self.execute_tools(tools_list)

    def _safe_play_audio(self, text_to_speak: str):
        """
        Synthesizes and plays all non-empty messages only in non-text chat mode.
        """
        if not self.text_chat_mode and text_to_speak:
```

```
            try:
                self.model_client.voice_synthesis(text_to_speak,
self.tts_out_path)
                self.play_audio_async(self.tts_out_path)
            except Exception as e:
                self.get_logger().error(f"Safe audio playback failed: {e}")
```

Code Analysis

The voice interaction functionality of the large AI model is collaboratively implemented by two independent ROS nodes, `asr.py` and `model_service.py`. They communicate via the ROS topic `/asr`, forming a complete processing loop.

1. **Voice Input and Publication (`asr.py`):**

- The `ASRNode` node handles voice input. The `kws_handler` function performs recording and text conversion.
- After successful conversion, the `asr_pub_result` function is called. This function places the obtained text string into a `std_msgs.msg.String` message and then publishes the message to the `/asr` topic via the `self.asr_pub` publisher.

2. **Subscription and Response (`model_service.py`):**

- During initialization, the `LargeModelService` node subscribes to the `/asr` topic using the `create_subscription` method and specifies `asr_callback` as its callback function.
- When `ASRNode` publishes a message on the `/asr` topic, the ROS system automatically calls the `asr_callback` function of `LargeModelService`, passing the message as the parameter `msg`.

3. **Interaction Loop**:

- After receiving the message, the `asr_callback` function extracts the text data (`msg.data`) and passes it to the large model for inference (`self.model_client.infer_with_text`).
- After inference is complete, the `process_model_result` function is called. This function extracts the text for the reply from the model results.
- Next, the `_safe_play_audio` function is called, which uses TTS (`self.model_client.voice_synthesis`) to synthesize the reply text into audio and play it.
- At this point, a complete interaction loop from "receiving user voice" to "responding to the user with voice" is completed. The two nodes are decoupled using ROS's publish-subscribe pattern; `asr.py` handles the input, and `model_service.py` handles the processing and output.

3. Practical Operation

3.1 Configuring Offline Voice Interaction

To implement a fully offline voice interaction system, you need to ensure that ASR, TTS, and LLM are all configured in offline mode.

1. **Open the main configuration file `yahboom.yaml`:**

```
gedit ~/yahboom_ws/src/largemodel/config/yahboom.yaml
```

2. **Modify/confirm the following key configurations:**

```
asr:
  ros__parameters:
    use_oline_asr: False               # Key: Set to False to enable offline ASR

model_service:
  ros__parameters:
    useolinetts: False                   # Key: Set to False to enable offline
TTS
    llm_platform: 'ollama'              # Key: Set to 'ollama' to enable offline
LLM
```

3. **Open the model interface configuration file `large_model_interface.yaml`:**

```
gedit ~/yahboom_ws/src/largemodel/config/large_model_interface.yaml
```

4. **Confirm all offline model paths are correct**:

```
# large_model_interface.yaml
## Offline large model
ollama_model: "qwen2.5:7b" # Ensure this model has been downloaded via ollama
pull

## Offline speech recognition
local_asr_model: "/path/to/your/SenseVoiceSmall" # Ensure the ASR model path is
correct

## Offline speech synthesis
zh_tts_model: "/path/to/your/zh_CN-huayan-medium.onnx" # Ensure the TTS model
path is correct
# ...
```

Recompile

```
cd ~/yahboom_ws/
colcon build
source install/setup.bash
```

3.2 Startup and testing functionality

1. **Start the `largemodel` main program**:

Run the following command to enable voice interaction:

```
ros2 launch largemodel largemodel_control.launch.py
```

2. **Testing**:

- **Wake-up**: Say into the microphone, "Hello, yahboom."
- **Dialogue**: Speaker responseAfter that, you can speak your question.
- **Observe the log:** In the terminal running the `launch` file, you should see:

1. The ASR node recognizes your question and prints it out.
2. The `model_service` node receives the text, calls the LLM, and prints the LLM's response.

- **Listen to the answer:** Later, you should be able to hear the robot answer your question in synthesized speech through the speaker.