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 **Large Language Model (LLM)** core to form a complete conversational system that can listen, speak, and think.

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

The core of this functionality is the `model_service` **ROS2 node**. It acts as the brain and central nervous system, subscribing to ASR recognition results, calling LLM for thinking, and then publishing LLM's text replies to the TTS node for voice synthesis.

1.2 Brief Implementation Principle

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

1. **Sound -> Text (ASR)**: The `asr` node continuously monitors environmental sounds. Once it detects that a user has finished speaking a sentence, it converts it to text and publishes it to the `/asr_text` topic.
2. **Text -> Think -> Text (LLM)**: The `model_service` node subscribes to the `/asr_text` topic. When it receives text from ASR, it sends it as a question (Prompt) to the locally deployed large language model (such as `Qwen` running through `Ollama`). The LLM generates a response text based on context.
3. **Text -> Sound (TTS)**: After the `model_service` node gets the LLM's response, it publishes it to the `/tts_text` topic.
4. **Sound 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 speakers.

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

2. Project Architecture

2.1 Key Code Analysis

The core of the entire process lies in how the `model_service` node connects input and output.

1. Subscribe to ASR Results (located in `largemodel/model_service.py`)
The `model_service` node will have a subscriber to receive text recognized by ASR.

```
# largemodel/model_service.py (core logic illustration)
class ModelService(Node):
    def __init__(self):
        super().__init__('model_service')
        # ...
        # Subscribe to ASR text output topic
        self.asr_subscription = self.create_subscription(
            String,
            'asr_text',
            self.asr_callback,
            10)

        # Create TTS text input topic publisher
        self.tts_publisher = self.create_publisher(String, 'tts_text', 10)

        # Initialize large model interface
        self.large_model_interface = LargeModelInterface(self)
```

Explanation: The node's `__init__` method clearly defines its role: an intermediary that can both "listen" to ASR results and "command" TTS to speak, and has a "brain" (`LargeModelInterface`) internally.

2. Process ASR Text and Call LLM (located in `largemodel/model_service.py`)
When ASR has new recognition results, `asr_callback` will be triggered.

```
# largemodel/model_service.py (core logic illustration)
    def asr_callback(self, msg):
        user_text = msg.data
        self.get_logger().info(f'Received from ASR: "{user_text}"')

        # Call large model interface for thinking
        # llm_platform determines whether to call Ollama or online API
        llm_platform = self.get_parameter('llm_platform').value
        response_text = self.large_model_interface.call_llm(user_text,
llm_platform)

        if response_text:
            self.get_logger().info(f'LLM response: "{response_text}"')
            # Send LLM response to TTS
            self.speak(response_text)
```

Explanation: This is the core logic of the system. After the callback function receives text, it immediately sends it to the LLM through `large_model_interface`. The `call_llm` method internally decides whether to connect to local Ollama or online API based on the `llm_platform` configuration.

3. Send LLM Response to TTS (located in `largemodel/model_service.py`)
The `speak` method is a simple wrapper for publishing text to the topic that the TTS node listens to.

```
# largemodel/model_service.py (core logic illustration)
    def speak(self, text):
        msg = String()
        msg.data = text
        self.tts_publisher.publish(msg)
```

Explanation: This function completes the final step of the data flow, passing the text result "thought" by the "brain" to the "mouth", thereby completing the entire closed loop of voice interaction.

3. Practical Operations

3.1 Configure Online LLM

1. **First obtain API Key from OpenRouter platform**
2. **Then update the key in the configuration file, open the model interface configuration file `large_model_interface.yaml`**:

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

3. **Enter your API Key**:
   Find the corresponding section and paste the API Key you just copied. Here we use Tongyi Qianwen configuration as an example

   ```
   # large_model_interface.yaml

   # OpenRouter platform configuration (OpenRouter Platform Configuration)
   openrouter_api_key: "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
   openrouter_model: "nvidia/nemotron-nano-12b-v2-vl:free" # Model to use, e.g., "google/gemini-pro-vision"
   ```

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

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

5. **Select the online platform to use**:
   Modify the `llm_platform` parameter to the platform name you want to use

   ```
   # yahboom.yaml

   model_service:
     ros__parameters:
       # ...
       llm_platform: 'openrouter'              # Currently selected large model platform
       # Available platforms: 'ollama','openrouter'
   ```

Recompile

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

3.2 Start and Test Function

1. **Start the `largemodel` main program**:
   Run the following command to start voice interaction:

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

2. **Testing**:
   - **Wake up**: Speak to the microphone: "Hello, Yahboom."
   - **Dialogue**: After the speaker responds, you can say the question you want to ask.
   - **Observe logs**: 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 out the LLM's response.
   - **Listen to answer**: Shortly after, you should hear the answer from the speakers.