3. AI Large Model Voice Interaction

3. AI Large Model Voice Interaction

1. Concept Introduction
    - 1.1 What is "AI Large Model Voice Interaction"?
    - 1.2 Implementation Principle Overview
2. Project Architecture
    - 2.1 Key Code Analysis
3. Practical Operation
    - 3.1 Configuring Online LLM
    - 3.2 Starting and Testing the Functionality

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. Project Architecture

2.1 Key Code Analysis

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

1. Subscribing to ASR Results (located in `largemodel/model_service.py`)
The `model_service` node has a subscriber to receive the text recognized by ASR.

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

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

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

Explanation: The node's `__init__` method clearly defines its role: a middleman that listens to ASR results and commands the TTS to speak, with an internal "brain" (`LargeModelInterface`).

2. Processing ASR Text and Calling the LLM (located in `largemodel/model_service.py`)
When the ASR generates new recognition results, `asr_callback` is triggered.

```
# largemodel/model_service.py (Core logic diagram)
    def asr_callback(self, msg):
        user_text = msg.data
        self.get_logger().info(f'从ASR收到: "{user_text}"')

        # Calling the large model interface for consideration
        # llm_platform determines whether to call Ollama or the 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回复: "{response_text}"')
            # Send LLM's reply to TTS
            self.speak(response_text)
```

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

3. Sending the LLM response to the TTS (located in `largemodel/model_service.py`)
The `speak` method is a simple wrapper that publishes the text to the topic listened to by the TTS node.

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

Explanation: This function completes the final step in the data flow, passing the text generated by the "brain" to the "mouth," thus completing the entire closed-loop voice interaction.

3. Practical Operation

3.1 Configuring Online LLM

Raspberry Pi 5 requires entering a Docker container; RDK X5 and Orin controllers do not:

```
./ros2_docker.sh
```

If you need to enter the same Docker container to input other commands later, simply enter `./ros2_docker.sh` again in the host machine's terminal.

1. **Then you need to 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
```

2. **Enter your API Key**:

Find the corresponding section and paste the API Key you just copied. Here's an example configuration using Tongyi Qianwen:

```
# large_model_interface.yaml

## Tongyi Qianwen
qianwen_api_key: "sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" # Paste your key
qianwen_model: "qwen-vl-max-latest" # You can choose the model as needed, such
as qwen-turbo, qwen-plus
```

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

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

4. **Select the online platform to use**:

Modify the `llm_platform` parameter to the name of the platform you want to use

```
# yahboom.yaml

model_service:
ros__parameters:
# ...
llm_platform: 'tongyi' # Optional platform: 'ollama', 'tongyi', 'spark',
'qianfan', 'openrouter'
```

After modifying the configuration file, you need to recompile and source it in the workspace:

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

3.2 Starting and Testing the Functionality

1. **After entering the Docker container, 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, Xiaoya.”
- **Dialogue:** After the speaker responds, you can speak your question.
- **Observe the 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, invokes the LLM, and prints the LLM's response.

- **Listen to the response:** You should hear the response from the speaker shortly.