# 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** combines the **offline ASR** and **offline TTS** described above with the **large language model (LLM)** core to form a complete conversational system that listens, speaks, and thinks.

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

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

### 1.2 Implementation Principles

This feature is implemented using a classic data flow pipeline:

1. **Audio -> Text (ASR)**: The `asr` node continuously listens to ambient sound. Once it detects a user speaking 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. Upon receiving the text from the ASR, it passes it as a prompt to a locally deployed large language model (such as `Qwen` running through `Ollama`). The LLM generates a text response 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 invokes the offline TTS model to synthesize it into audio and plays it through the speaker.

This process forms a complete closed loop: `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 input and output in series.

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

```
# largemodel/model_service.py (核心逻辑示意)
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 `__init__` method of the node clearly defines its role: a middleman that can both "listen" to ASR results and "command" TTS to speak, and has an internal "brain" (`LargeModelInterface`).

**2. Process ASR text and call LLM (located in `largemodel/model_service.py`)**
When ASR has a new recognition result, `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'Received from ASR: "{user_text}"')

        # Call the large model interface for thinking
        # 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 Reply: "{response_text}"')
            # Send LLM's reply to TTS
            self.speak(response_text)
```

**Explanation**: This is the core logic of the system. Upon receiving the text, the callback function immediately sends it to the LLM via `large_model_interface`. The `call_llm` method internally determines whether to connect to the local Ollama or the 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 for publishing 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 last step of the data flow, passing the text results generated by the "brain" to the "mouth", thus completing the closed loop of the entire voice interaction.

# 3. Practice

## 3.1 Configuring Online LLM

1. **First, obtain an API key from any of the platforms mentioned in the previous tutorial**
2. **Next, you'll 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
   ```

3. **Fill in your API Key**:
   Find the corresponding part and paste the API Key you just copied into it. Here we take Tongyi Qianwen configuration as an example

   ```
   # large_model_interface.yaml

   ## Thousand Questions on Tongyi
   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
   ```

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

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

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

   ```
   # yahboom.yaml

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

## 3.2 Start and test the function

1. **start up**:

**Note: The startup commands for CSi cameras and USB microphone cameras are different. Please run the appropriate command for your camera.**

**CSI Camera**

Start udp video streaming (host machine)

```
./start_csi.sh
```

Enter the CSI camera docker (host machine)

```
./run_csi_docker.sh
```

Start topic conversion (docker)

```
python3 ~/temp/udp_camera_publisher.py
```

View container id

```
docker ps
```

According to the container ID shown above, multiple terminals enter the same docker

```
docker exec -it  container_id  /bin/bash
```

Run the following command to enable voice interaction:

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

**USB Camera**

Enter the USB camera docker (host machine)

```
./run_usb_docker.sh
```

Run the following command to enable voice interaction:

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

2. **test**:
    - **Wake up**: Say "Hi, yahboom" into the microphone.
    - **Talk**: After the speaker responds, you can speak your question.
    - **Watch the log**: In the terminal running the `launch` file, you should see the following:
    1. The ASR node recognizes your question and prints it.

2. The `model_service` node receives the text, calls the LLM, and prints the LLM's response.
- **Listen for the answer**: After a while, you should hear the answer from the speaker.