# 1. Online speech-to-text (ASR)



# 1. Introduction

## 1.1 What is "ASR"?

ASR (Automatic Speech Recognition) is a technology that converts human speech signals into text. It is widely used in intelligent assistants, voice command control, telephone customer service automation, and real-time subtitle generation. The goal of ASR is to enable machines to "understand" human speech and convert it into a form that computers can process and understand.

## 1.2 Implementation Principles

The implementation of an ASR system relies primarily on the following key technical components:

### 1. Acoustic Model

- The acoustic model is responsible for converting the input audio signal into phonemes or subword units. This typically involves a feature extraction step, such as using Mel-Frequency Cepstral Coefficients (MFCCs) or filter banks to represent the audio signal.
- These features are then fed into a deep neural network (DNN), convolutional neural network (CNN), recurrent neural network (RNN), or the more advanced Transformer architecture for training to learn how to map audio features to corresponding phonemes or text.

### 2. Language Model

- A language model predicts the most likely next word given the previous context, thereby improving recognition accuracy. It is trained on large amounts of text data to understand which word sequences are most likely to occur.
- Common language models include n-gram models, RNN-based language models (LMs), and the more recently popular Transformer-based language models.

### 3. Pronunciation Dictionary

- A pronunciation dictionary provides a mapping between words and their corresponding pronunciations. This is crucial for connecting the acoustic model and language model, as it allows the system to understand and match the sounds it hears based on known pronunciation rules.

### 4. Decoder

- The decoder's task is to find the most likely word sequence as output given the acoustic model, language model, and pronunciation dictionary. This process typically involves complex search algorithms, such as the Viterbi algorithm or graph-based search methods, to find the optimal path.

### 5. End-to-End ASR

- With the development of deep learning, end-to-end ASR systems have emerged. These systems attempt to learn text output directly from raw audio signals, without the need for explicit acoustic models, separate pronunciation lexicons, and language models. These systems are often based on sequence-to-sequence (Seq2Seq) frameworks, using, for example, attention mechanisms or the Transformer architecture, significantly simplifying the complexity of traditional ASR systems.

In general, modern ASR systems achieve efficient and accurate speech-to-text conversion by combining the aforementioned components and leveraging large-scale datasets and powerful computing resources for training. With technological advances, the performance of ASR systems continues to improve, and their application scenarios are becoming increasingly broad.

# 2. Project Architecture

## Key Code

### 1. Speech Processing and Recognition Core (`largemodel/largemodel/asr.py`)

```
# From largemodel/largemodel/asr.py
def kws_handler(self)->None:
    if self.stop_event.is_set():
        return

    if self.listen_for_speech(self.mic_index):
        asr_text = self.ASR_conversion(self.user_speechdir)  # Perform ASR
conversion
        if asr_text =='error':  # Check if ASR result length is less than 4
characters
            self.get_logger().warn("I still don't understand what you mean.
Please try again")
            playsound(self.audio_dict[self.error_response])  # Error response
        else:
            self.get_logger().info(asr_text)
            self.get_logger().info("okay😜, let me think for a moment...")
            self.asr_pub_result(asr_text)  # Publish ASR result
    else:
        return

def ASR_conversion(self, input_file:str)->str:
```

```
    if self.use_oline_asr:
        result=self.modelinterface.oline_asr(input_file)
        if result[0] == 'ok' and len(result[1]) > 4:
            return result[1]
        else:
            self.get_logger().error(f'ASR Error:{result[1]}')  # ASR error.
            return 'error'
    else:
        result=self.modelinterface.SenseVoiceSmall_ASR(input_file)
        if result[0] == 'ok' and len(result[1]) > 4:
            return result[1]
        else:
            self.get_logger().error(f'ASR Error:{result[1]}')  # ASR error.
            return 'error'
```

## 2. VAD smart recording (`largemodel/largemodel/asr.py`)

```
# From largemodel/largemodel/asr.py
def listen_for_speech(self,mic_index=0):
    p = pyaudio.PyAudio()   # Create PyAudio instance.
    audio_buffer = []       # Store audio data.
    silence_counter = 0     # Silence counter.
    MAX_SILENCE_FRAMES = 90  # Stop after 900ms of silence (30 frames * 30ms)
    speaking = False  # Flag indicating speech activity.
    frame_counter = 0  # Frame counter.
    stream_kwargs = {
        'format': pyaudio.paInt16,
        'channels': 1,
        'rate': self.sample_rate,
        'input': True,
        'frames_per_buffer': self.frame_bytes,
    }
    if mic_index != 0:
        stream_kwargs['input_device_index'] = mic_index

    # Prompt the user to speak via the buzzer.
    self.pub_beep.publish(UInt16(data = 1))
    time.sleep(0.5)
    self.pub_beep.publish(UInt16(data = 0))

    try:
        # Open audio stream.
        stream = p.open(**stream_kwargs)
        while True:
            if self.stop_event.is_set():
                return False

            frame = stream.read(self.frame_bytes, exception_on_overflow=False)
 # Read audio data.
            is_speech = self.vad.is_speech(frame, self.sample_rate)  # VAD
detection.

            if is_speech:
                # Detected speech activity.
                speaking = True
                audio_buffer.append(frame)
                silence_counter = 0
```

```
            else:
                if speaking:
                    # Detect silence after speech activity.
                    silence_counter += 1
                    audio_buffer.append(frame)  # Continue recording buffer.

                    # End recording when silence duration meets the threshold.
                    if silence_counter >= MAX_SILENCE_FRAMES:
                        break
            frame_counter += 1
            if frame_counter % 2 == 0:
                self.get_logger().info('1' if is_speech else '-')
                # Real-time status display.
    finally:
        stream.stop_stream()
        stream.close()
        p.terminate()

    # Save valid recording (remove trailing silence).
    if speaking and len(audio_buffer) > 0:
        # Trim the last silent part.
        clean_buffer = audio_buffer[:-MAX_SILENCE_FRAMES] if len(audio_buffer) >
MAX_SILENCE_FRAMES else audio_buffer

        with wave.open(self.user_speechdir, 'wb') as wf:
            wf.setnchannels(1)
            wf.setsampwidth(p.get_sample_size(pyaudio.paInt16))
            wf.setframerate(self.sample_rate)
            wf.writeframes(b''.join(clean_buffer))
            return True
```

## Code Analysis

ASR (speech-to-text) functionality is provided by the `ASRNode` node (`asr.py`). This node is responsible for recording, converting, and publishing audio.

1. **Audio Recording (`listen_for_speech`):**
    - This function uses the `pyaudio` library to capture the audio stream from the microphone.
    - It integrates the `webrtcvad` library for voice activity detection (VAD). The function loops through audio frames and uses `vad.is_speech()` to determine whether each frame contains human voice.
    - When speech is detected, the data is written to a buffer. Recording stops when sustained silence (defined by `MAX_SILENCE_FRAMES`) is detected.
    - Finally, the audio data in the buffer is written to a `.wav` file in the `self.user_speechdir` directory.
2. **Backend Selection and Execution (`ASR_conversion`):**
    - The `kws_handler` function calls the `ASR_conversion` function after successful recording.
    - This function determines which backend implementation to call by reading the ROS parameter `use_oline_asr` (a Boolean value).
    - If `false`, the `self.modelinterface.SenseVoiceSmall_ASR` method is called for local recognition.
    - If `true`, the `self.modelinterface.oline_asr` method is called for online recognition.

- This function passes the audio file path as a parameter to the selected method and processes the returned result.

3. **Result Publishing (`asr_pub_result`):**
   - After `ASR_conversion` returns valid text, `kws_handler` calls the `asr_pub_result` function.
   - This function encapsulates a text string in a `std_msgs.msg.String` message and publishes it to the `/asr` topic via the ROS publisher.

# 3. Practice

## 3.1 Configuring Online ASR

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

```
./ros2_docker.sh
```

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

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

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

2. **Enter your API credentials**:
   Find the Online ASR section, enter your key, and select Speech Recognition to Model.

   ```
   # large_model_interface.yaml
   # ----------------------------------------------------------------------------
   ---
   # Large Language Models Settings
   # ----------------------------------------------------------------------------
   ---
   ## Online Large Language Models
   # Tongyi Platform Configuration
   online_asr_api_key: "Your_AccessKey_ID"

   # ----------------------------------------------------------------------------
   ---
   # Speech Recognition Settings
   # ----------------------------------------------------------------------------
   ---
   ## Online ASR
   oline_asr_model: 'paraformer-realtime-8k-v2'  # Optional: 'paraformer-
   realtime-v2', 'paraformer-realtime-v1', etc., using the API of the Tongyi
   Qianwen platform
   oline_asr_sample_rate: 16000      # ASR audio sampling rate
   ```

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

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

4. **Enable online ASR mode**:
   Set the `use_oline_asr` parameter to `True`.

```
# yahboom.yaml

asr:
  ros__parameters:
    use_oline_asr: True  # Key: Change from False to True
    regional_setting : "China"
```

## 3.2 Start and test the functionality

1. **After entering the docker container, start the command**:

```
ros2 launch largemodel asr_only.launch.py
```

2. **Test**:
   Say "Hi,yahboom" into the microphone, and it will respond: "I'm here." Then you can start talking. Finally, the recorded sound will be converted into text and printed on the terminal.