1. Online Speech to Text (ASR)

1. Concept 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, automated telephone customer service, real-time subtitle generation, and other fields. The goal of ASR is to enable machines to "understand" human language and convert it into a form that computers can process and understand.

1.2 Implementation Principle

The implementation of ASR systems mainly relies on the following key technical components:

1. Acoustic Model

- The acoustic model is responsible for converting input sound signals into phonemes or subword units. This usually involves feature extraction steps, such as using Mel-frequency cepstral coefficients (MFCCs) or filter banks to represent audio signals.
- These features are then fed into deep neural networks (DNNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), or more advanced Transformer architectures for training to learn how to map from audio features to corresponding phonemes or text.

2. Language Model

- The language model is used to predict the most likely next word given the preceding context, thereby helping to improve recognition accuracy. It is trained based on large amounts of text data and understands which word sequences are more likely to appear.
- Common language models include n-gram models, RNN-based LM, and recently popular Transformer-based LM.

3. Pronunciation Dictionary

- The pronunciation dictionary provides the mapping relationship 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 heard sounds 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 usually 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 separation of acoustic models, pronunciation dictionaries, and language models. Such systems are often based on sequence-to-sequence (Seq2Seq) frameworks, such as using attention mechanisms or Transformer architectures, greatly simplifying the complexity of traditional ASR systems.

Overall, modern ASR systems achieve efficient and accurate human speech-to-text capabilities by combining the above components and utilizing large-scale datasets and powerful computing resources for training. As technology advances, the performance of ASR systems continues to improve, and application scenarios become increasingly widespread.

2. Project Architecture

Key Code

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

```python
# 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

The ASR (Speech to Text) functionality is provided by the `ASRNode` node (`asr.py`). This node is responsible for audio recording, conversion, and publishing.

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

    - This function passes the audio file path as a parameter to the selected method and handles 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 the text string in a `std_msgs.msg.String` message and publishes it to the `/asr` topic through a ROS publisher.

3. Practical Operations

3.1 Configure Online ASR

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

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

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

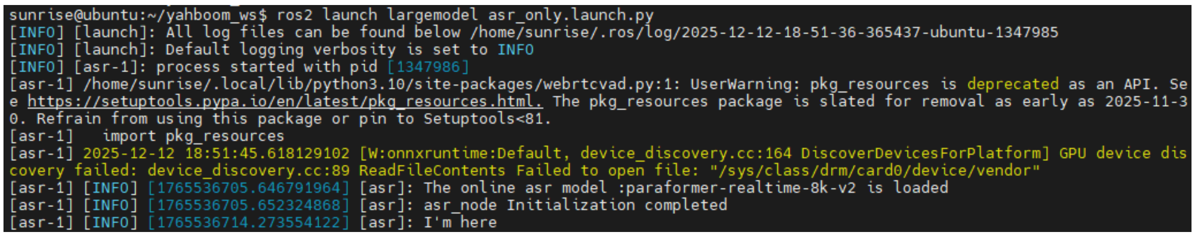
    ```yaml
    # yahboom.yaml

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

3.2 Start and Test Function

1. **Startup command**:

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

2. **Testing**:
    Speak to the microphone: "Hello Yahboom", it will respond: "I'm here", then you can start speaking, and finally it will display that the captured sound has been converted to text and printed in the terminal.



```
sunrise@ubuntu:~/yahboom_ws$ ros2 launch largemodel asr_only.launch.py
[INFO] [launch]: All log files can be found below /home/sunrise/.ros/log/2025-12-12-18-51-36-365437-ubuntu-1347985
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [asr-1]: process started with pid [1347986]
[asr-1] /home/sunrise/.local/lib/python3.10/site-packages/webrtcvad.py:1: UserWarning: pkg_resources is deprecated as an API. Se
e https://setuptools.pypa.io/en/latest/pkg_resources.html. The pkg_resources package is slated for removal as early as 2025-11-3
0. Refrain from using this package or pin to Setuptools<81.
[asr-1]   import pkg_resources
[asr-1] 2025-12-12 18:51:45.618129102 [W:onnxruntime:Default, device_discovery.cc:164 DiscoverDevicesForPlatform] GPU device dis
covery failed: device_discovery.cc:89 ReadFileContents Failed to open file: "/sys/class/drm/card0/device/vendor"
[asr-1] [INFO] [1765536705.646791964] [asr]: The online asr model :paraformer-realtime-8k-v2 is loaded
[asr-1] [INFO] [1765536705.652324868] [asr]: asr_node Initialization completed
[asr-1] [INFO] [1765536714.273554122] [asr]: I'm here
```