2. Online Text to Speech (TTS)

1. Concept Introduction

1.1 What is "TTS"?

TTS technology is a technology that converts written text into audible human speech output. It enables computers to "read aloud" text content and is widely used in accessibility reading, intelligent assistants, navigation systems, educational software, and many other fields. Through TTS, users can hear natural and smooth human voices generated by machines, greatly enhancing the convenience and flexibility of information acquisition.

1.2 Brief Implementation Principle

The implementation of TTS systems mainly includes the following key steps and technologies:

1. Text Analysis

- In this stage, the input text is first preprocessed, including but not limited to removing irrelevant characters, standardizing punctuation and case, word segmentation, and identifying special symbols such as numbers and converting them to corresponding word forms.
- At the same time, linguistic analysis of the text is also required, such as determining the pronunciation of each word (this usually requires using a pronunciation dictionary), stress position, intonation pattern, and sentence structure.

2. Language Processing

- This step mainly focuses on how to correctly pronounce and adjust intonation according to context. For example, the word "read" has different pronunciations in different tenses (past tense/past participle pronounced as /red/, while other cases pronounced as /riːd/). Therefore, a powerful language model is needed to understand these subtle differences.
- This also involves prosody modeling, which determines which parts should be emphasized, whether the speech rate should be fast or slow, and the emotional tone of the entire

sentence.

3. Speech Synthesis

- The text information processed in the previous two stages is sent to the speech synthesis engine, which is responsible for generating actual sound waveforms.
- Traditional TTS systems use concatenative synthesis methods, which select appropriate units from a pre-recorded speech segment database to splice together to form complete sentences. Although this method can produce high-quality speech, it is limited by the samples in the database.
- Modern TTS systems rely more on parametric synthesis or neural network synthesis (such as WaveNet, Tacotron, etc.). These methods can directly predict speech features from text and generate continuous sound signals. Particularly, deep learning-based methods can better capture subtle changes in speech, thereby producing more natural and smooth voices.

4. Sound Waveform Generation

- Finally, the generated sound waveform is further processed to ensure its quality meets expected standards, such as adjusting volume, equalizing frequency response, etc.
- After that, these audio data can be played through speakers or other audio playback devices for people to listen to.

With the development of artificial intelligence and machine learning technologies, especially the application of deep learning, TTS systems have not only significantly improved in accuracy, but also made great progress in naturalness and emotional expression, making machine-generated speech increasingly close to real human voice.

2. Code Analysis

Key Code

1. TTS Initialization and Calling (`largemodel/largemodel/model_service.py`)

```
# From largemodel/largemodel/model_service.py
class LargeModelService(Node):
    def __init__(self):
        # ...
        self.system_sound_init()
        # ...

    def init_param_config(self):
        # ...
        self.declare_parameter('useolinetts', False)
        self.useolinetts =
self.get_parameter('useolinetts').get_parameter_value().bool_value
        if self.useolinetts:
            self.tts_out_path = os.path.join(self.pkg_path, "resources_file",
"tts_output.mp3")
        else:
            self.tts_out_path = os.path.join(self.pkg_path, "resources_file",
"tts_output.wav")

    def system_sound_init(self):
        """Initialize TTS system"""
        model_type = "oline" if self.useolinetts else "local"
```

```
        self.model_client.tts_model_init(model_type, self.language)
        self.get_logger().info(f'TTS initialized with {model_type} model')

    def _safe_play_audio(self, text_to_speak: str):
        """
        Synthesizes and plays all non-empty messages only in non-text chat mode.
        """
        if not self.text_chat_mode and text_to_speak:
            try:
                self.model_client.voice_synthesis(text_to_speak,
self.tts_out_path)
                self.play_audio_async(self.tts_out_path)
            except Exception as e:
                self.get_logger().error(f"Safe audio playback failed: {e}")
```

2. TTS Backend Implementation (`largemodel/utils/large_model_interface.py`)

```
# From largemodel/utils/large_model_interface.py
class model_interface:
    # ...
    def tts_model_init(self,model_type='oline',language='zh'):
        if model_type=='oline':
            if self.tts_supplier=='baidu':
                self.token=self.fetch_token()

            self.model_type='oline'
        elif model_type=='local':
            self.model_type='local'
            if language=='zh':
                tts_model=self.zh_tts_model
                tts_json=self.zh_tts_json
            elif language=='en':
                tts_model=self.en_tts_model
                tts_json=self.en_tts_json
            self.synthesizer = piper.PiperVoice.load(tts_model,
config_path=tts_json, use_cuda=False)

    def voice_synthesis(self,text,path):
        if self.model_type=='oline':
            if self.tts_supplier=='baidu':
                # ... (Baidu TTS implementation)
                pass
            elif self.tts_supplier=='aliyun':
                # ... (Aliyun TTS implementation)
                pass
        elif self.model_type=='local':
            with wave.open(path, 'wb') as wav_file:
                wav_file.setnchannels(1)
                wav_file.setsampwidth(2)
                wav_file.setframerate(self.synthesizer.config.sample_rate)
                self.synthesizer.synthesize(text, wav_file)
```

Code Analysis

The text-to-speech (TTS) functionality is initiated by the `LargeModelService` node and implemented by the `model_interface` class. Its design switches between different backend services through parameter configuration.

1. **Initialization Process (`model_service.py`)**:
    - During `LargeModelService` initialization, the `init_param_config` function reads the boolean value `useolinetts` from the ROS parameter server.
    - The `system_sound_init` function decides whether to pass `'local'` or `'oline'` string to the `self.model_client.tts_model_init` method based on the value of `useolinetts`.
    - In `large_model_interface.py`, the `tts_model_init` method executes corresponding initialization logic based on the received string parameter. If it's `'local'`, it uses `piper.PiperVoice.load` to load local model files.
2. **Synthesis and Playback Process (`model_service.py`)**:
    - When voice playback is needed, the `_safe_play_audio` function is called.
    - This function first calls the `self.model_client.voice_synthesis` method, passing the text to be converted and the target audio path `self.tts_out_path`.
    - After the `voice_synthesis` method completes execution and generates the audio file, `_safe_play_audio` then calls `self.play_audio_async` to play the file asynchronously.
3. **Backend Implementation Selection (`large_model_interface.py`)**:
    - The `voice_synthesis` method is the backend dispatch center for TTS functionality. It internally selects the execution path by checking the `self.model_type` attribute value set during initialization.
    - If `self.model_type` is `'local'`, the code block uses Python's `wave` library to open a WAV file, sets its header parameters (channels, sample bit width, sample rate), and then calls the `self.synthesizer.synthesize` method to directly write the text-synthesized audio stream to that file.
    - If `self.model_type` is `'oline'`, it enters logic branches for different cloud service providers (such as Baidu, Aliyun).
    - This structure separates the upper-level node calls ("say this text") from the lower-level specific synthesis technologies (which library to use, which API to call).

3. Practical Operations

3.1 Configure Online TTS

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

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

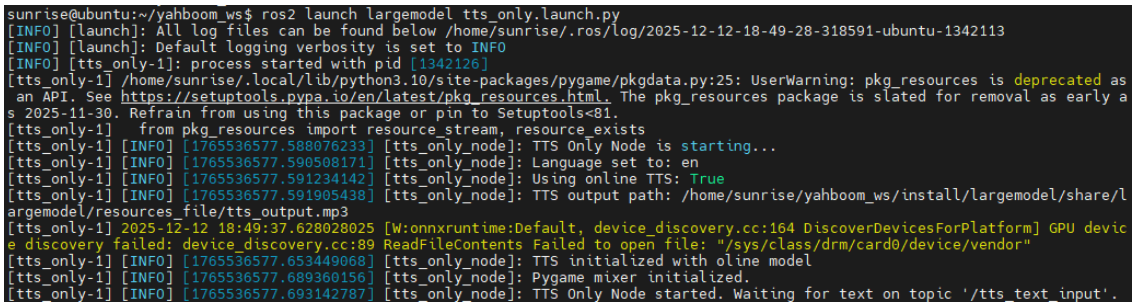
2. **Enable online TTS mode**:
    Set the `useolinetts` parameter to `True`.

```yaml
# yahboom.yaml

model_service:
  ros__parameters:
    useolinetts: True  # Whether to use online TTS (True for online, False
for offline)

tts_only_node:                          # TTS only node parameters
  ros__parameters:
    language: 'en'                      # TTS language
    useolinetts: True                   # Whether to use online TTS
```

3.2 Start and Test Function

1. **Start TTS node**:
   Run the following command:

```
ros2 launch largemodel tts_only.launch.py
```



2. **Send text to be synthesized**:
   Open a new terminal and run the following command to publish a voice broadcast message:

```
ros2 topic pub --once /tts_text_input std_msgs/msg/String '{data: "Speech
synthesis test successful"}'
```

3. **Testing**:
   If everything is working properly, you should hear the robot say "Speech synthesis test successful" in synthesized voice through your speakers.