2. Online Text-to-Speech (TTS)

1. Concept Introduction

1.1 What is "TTS"?

TTS technology converts written text into human-readable speech output. It enables computers to "read" text aloud and is widely used in a variety of fields, including accessible reading, intelligent assistants, navigation systems, and educational software. Through TTS, users can hear natural, fluent machine-generated human voices, greatly improving the convenience and flexibility of information acquisition.

1.2 Overview of Implementation Principles

The implementation of a TTS system primarily involves 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 capitalization, segmenting words, recognizing special characters such as numbers, and converting them into their corresponding word forms.
- Linguistic analysis is also required, such as determining the pronunciation of each word (this typically requires the use of a pronunciation dictionary), stress placement, intonation patterns, and sentence structure.

2. Language Processing

- This step focuses on correctly pronouncing and adjusting intonation based on context. For example, the word "read" has different pronunciations in different tenses (the past tense/past participle is pronounced as /red/, while other tenses are pronounced as /riːd/). Therefore, a powerful language model is needed to understand these nuances.
- This also involves prosodic modeling, which determines which parts should be emphasized, whether the speaking rate should be fast or slow, and the emotional tone of the entire

sentence.

3. Speech Synthesis

- After the text information processed by the previous two stages is fed into the speech synthesis engine, which is responsible for generating the actual sound waveform.
- Traditional TTS systems use concatenative synthesis, selecting appropriate units from a database of pre-recorded speech segments and concatenating them to form complete sentences. While 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 and Tacotron). These methods can directly predict speech features from text and generate continuous speech signals. Deep learning-based methods, in particular, are better able to capture subtle changes in speech, resulting in more natural and fluent speech.

4. Sound Waveform Generation

- Ultimately, the generated sound waveform undergoes further processing to ensure its quality meets the desired standards, such as adjusting volume and equalizing frequency response.
- Afterward, this audio data can be played back through speakers or other audio playback devices for people to listen to.

With the advancement 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 significant progress in naturalness and emotional expression, making machine-generated speech increasingly similar to human voices.

2. Code Analysis

Key Code

1. TTS Initialization and Invocation (`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) function is invoked by the `LargeModelService` node and implemented by the `model_interface` class. Its design uses parameter configuration to switch between different backend services.

1. **Initialization Process (`model_service.py`):**
    - During `LargeModelService` initialization, the `init_param_config` function reads the Boolean value `useolinetts` from the ROS parameter server.
    - Based on the value of `useolinetts`, the `system_sound_init` function passes either the `'local'` or `'oline'` string to the `self.model_client.tts_model_init` method.
    - In `large_model_interface.py`, the `tts_model_init` method executes the corresponding initialization logic based on the string parameter received. If the value is `'local'`, `piper.PiperVoice.load` is used to load the local model file.
2. **Synthesis and Playback Process (`model_service.py`):**
    - When voice playback is required, the `_safe_play_audio` function is called.
    - This function first calls the `self.model_client.voice_synthesis` method, passing in the text to be converted and the target audio path `self.tts_out_path`.
    - After the `voice_synthesis` method completes and generates an audio file, `_safe_play_audio` calls `self.play_audio_async` to asynchronously play the file.
3. **Backend Implementation Selection (`large_model_interface.py`):**
    - The `voice_synthesis` method is the backend dispatch center for the TTS functionality. Internally, it selects the execution path by checking the `self.model_type` property 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, and sampling rate), and then calls `self.synthesizer.synthesize` to write the synthesized text audio stream directly to the file.
    - If `self.model_type` is `'oline'`, the code branch will be executed for different cloud service providers (such as Baidu and Alibaba Cloud).
    - This structure separates the upper-level node call ("speak this sentence") from the lower-level specific synthesis technology (which library to use, which API to call).

3. Practical Operations

3.1 Configuring Online TTS

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 TTS section (Baidu is used as an example here). Enter your key and optionally specify a preferred voice name.

```
# large_model_interface.yaml

# Baidu Smart Cloud Speech Synthesis Configuration
baidu_API_KEY: 'lQ3ybx9UsPMCvpqZKpgkEa2q'        # Baidu platform API key
baidu_SECRET_KEY: 'KBT3iWvMu1QXUVUL0CeNrKDhc1290Uo9'  # Baidu platform
SECRET key
CUID: 'IN7dAbz1thDJKVhLdypkpB6sDVG86xeG'         # Device identification
PER: 4  # Pronounced by: Basic Sound Library (0=Du Xiaomei, 1=Du Xiaoyu,
3=Du Xiaoyao, 4=Du Yaya)
SPD: 5  # Speech rate (0-15), default 5
PIT: 5  # Pitch (0-15), default 5
VOL: 5  # Volume (0-9), default 5
```

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

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

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

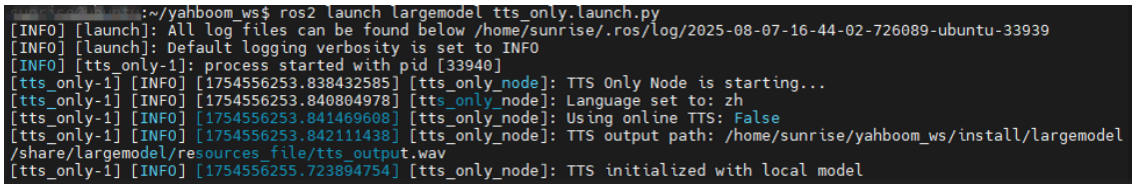
```
# yahboom.yaml

model_service:
  ros__parameters:
    useolinetts: True  # Key: Change from False to True
    regional_setting : "China"
```

3.2 Starting and Testing the Functionality

1. **After entering Docker, start the TTS node**:
   Run the following command:

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



2. **Send the text to be synthesized**:
   Open a new terminal, enter the same Docker container, and run the following command to publish a voice message:

```
ros2 topic pub --once /tts_text_input std_msgs/msg/String '{data: "语音合成测
试成功"}'
```

3. **Test**:
   If everything went well, you should hear the robot say "Speech synthesis test successful" in a synthesized voice through your speakers.