

2. Offline Text-to-Speech (TTS)

2. Offline Text-to-Speech (TTS)

1. Introduction

1.1 What is "TTS"?

1.2 Brief Description of Implementation Principles

1. Text Analysis

2. Language Processing

3. Speech Synthesis

4. Sound Waveform Generation

2. Code Analysis

Key Code

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

2. TTS backend implementation (`largemodel/utis/large_model_interface.py`)

Code Analysis

3. Practical Operations

3.1 Configuring Offline TTS

3.2 Starting and Testing the Functionality

4. Common Problems and Solutions

4.1 Playback Issues

Issue 1: The program runs normally without errors, but no sound is heard.

1. Introduction

1.1 What is "TTS"?

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

1.2 Brief Description of Implementation Principles

The implementation of a TTS system 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 capitalization, word segmentation, and recognizing special symbols such as numbers and converting them into their corresponding word forms.
- Simultaneously, linguistic analysis of the text is required, such as determining the pronunciation of each word (which usually requires the use of a pronunciation dictionary), stress positions, intonation patterns, and sentence structure.

2. Language Processing

- This step primarily focuses on how to correctly pronounce and adjust intonation based on context. For example, the word "read" has different pronunciations in different tenses (past tense/past participle is pronounced /red/, while in other cases it is pronounced /ri:d/). Therefore, a powerful language model is needed to understand these subtle differences.
- This also involves prosodic modeling, i.e., determining which parts should be emphasized, whether the speech rate should be fast or slow, and the overall emotional tone of the sentence.

3. Speech Synthesis

- The text information processed in the previous two stages is fed into a speech synthesis engine, which is responsible for generating the actual sound waveforms.
- Traditional TTS systems use a concatenation synthesis method, which selects appropriate units from a pre-recorded speech segment database and concatenates them to form a complete sentence. 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, Tacotron, etc.). These methods can directly predict speech features from text and generate continuous sound signals. Especially deep learning-based methods can better capture subtle changes in speech, thus producing more natural and fluent sounds.

4. Sound Waveform Generation

- Finally, the generated sound waveform is further processed to ensure its quality meets expected 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 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 strides in naturalness and emotional expression, making machine-generated speech increasingly closer to real 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/utis/large_model_interface.py`)

```

# From largemodel/utis/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 `useonlinetts` from the ROS parameter server.
- Based on the value of `useonlinetts`, the `system_sound_init` function passes either the `'local'` or `'online'` 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 `'online'`, 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 Offline TTS

To enable offline TTS, you need to correctly configure `yahboom.yaml` and `large_model_interface.yaml` and ensure that the local model is correctly placed.

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

```
./ros2_docker.sh
```

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

1. Open the main configuration file:

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

2. Modify/confirm the following key configuration::

```
model_service:                                #Model server node parameters
ros__parameters:
  language: 'zh'                               #Large Model Interface Language
  useolinetts: False                           #whether to use online speech
  synthesis (True to use online, False to use offline)
  regional_setting : "china"
```

useolinetts Make sure this is set to False to use the local model.

Select "zh" for Chinese and "en" for English.

3. Open the model interface configuration file:

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

4. Confirm the offline model path:

- Raspberry Pi 5

```
# large_model_interface.yaml
## Offline speech synthesis (Offline TTS)
# Chinese TTS model
zh_tts_model: "/root/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-
huayan-medium.onnx"
zh_tts_json: "/root/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-
huayan-medium.onnx.json"

# English TTS model
en_tts_model: "/root/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-
libritts-high.onnx"
en_tts_json: "/root/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-
libritts-high.onnx.json"
```

- RDK X5

```
# large_model_interface.yaml
## Offline speech synthesis (Offline TTS)
# Chinese TTS model
zh_tts_model:
"/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-huayan-medium.onnx"
zh_tts_json:
"/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-huayan-medium.onnx.json"

# English TTS model
en_tts_model:
"/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-libritts-high.onnx"
en_tts_json:
"/home/sunrise/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-libritts-high.onnx.json"
```

- ORIN

```
# large_model_interface.yaml
## Offline speech synthesis (Offline TTS)
# Chinese TTS model
zh_tts_model:
"/home/jetson/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-huayan-medium.onnx"
zh_tts_json:
"/home/jetson/yahboom_ws/src/largemodel/MODELS/tts/zh/zh_CN-huayan-medium.onnx.json"

# English TTS model
en_tts_model:
"/home/jetson/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-libritts-high.onnx"
en_tts_json:
"/home/jetson/yahboom_ws/src/largemodel/MODELS/tts/en/en_US-libritts-high.onnx.json"
```

3.2 Starting and Testing the Functionality

1. Start the TTS node:

Run the following command:

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

```
~/yahboom_ws$ ros2 launch largemodel tts_only.launch.py
[INFO] [launch]: All log files can be found below /home/sunrise/.ros/log/2025-08-07-16-44-02-726089-ubuntu-33939
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [tts_only-1]: process started with pid [33940]
[tts_only-1] [INFO] [1754556253.838432585] [tts_only_node]: TTS Only Node is starting...
[tts_only-1] [INFO] [1754556253.840804978] [tts_only_node]: Language set to: zh
[tts_only-1] [INFO] [1754556253.841469608] [tts_only_node]: Using online TTS: False
[tts_only-1] [INFO] [1754556253.842111438] [tts_only_node]: TTS output path: /home/sunrise/yahboom_ws/install/largemodel/share/largemodel/resources_file/tts_output.wav
[tts_only-1] [INFO] [1754556255.723894754] [tts_only_node]: TTS initialized with local model
```

2. Send the text to be synthesized:

Open a new terminal 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 works correctly, you should hear the robot say "Speech Synthesis Test Successful" in a synthesized voice through your speakers.

4. Common Problems and Solutions

4.1 Playback Issues

Issue 1: The program runs normally without errors, but no sound is heard.

Solution:

1. **Check Audio Output:** Confirm that your system's audio output device is selected correctly and the volume is not muted. Try playing a standard music file to test the hardware.