

Offline Text-to-Speech (TTS)

Offline Text-to-Speech (TTS)

1. Concept Introduction
 - 1.1 What is "TTS"?
 - 1.2 Overview 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 Calling (`largemodel/largemodel/model_service.py`)
 2. TTS backend implementation (`largemodel/utils/large_model_interface.py`)
 - Code Analysis
3. Practical Operations
 - 3.1 Configuring Offline TTS
 3. Start and Test the Functionality
4. Common Problems and Solutions
 - 4.1 Playback Issues
 - Problem 1: The program runs normally without errors, but no sound is heard.

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 /ri:d/). 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 variations 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.
- This audio data can then 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 achieved 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 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.useonline tts:
            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 = "online" if self.useonline tts 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='online', language='zh'):
        if model_type=='online':
            if self.tts_supplier=='baidu':
                self.token=self.fetch_token()

                self.model_type='online'
            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=='online':
            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 `useonline tts` from the ROS parameter server.
- The `system_sound_init` function passes the `'local'` or `'online'` string to the `self.model_client.tts_model_init` method based on the value of `useonline tts`.
- In `large_model_interface.py`, the `tts_model_init` method executes the corresponding initialization logic based on the received string parameter. If the string parameter 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.

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: 'en'                            #Large Model Interface Language
    useolinetts: False                        #Whether to use online speech
    synthesis (True to use online, False to use offline)
    regional_setting : "international"
```

Useolinetts must be 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
```

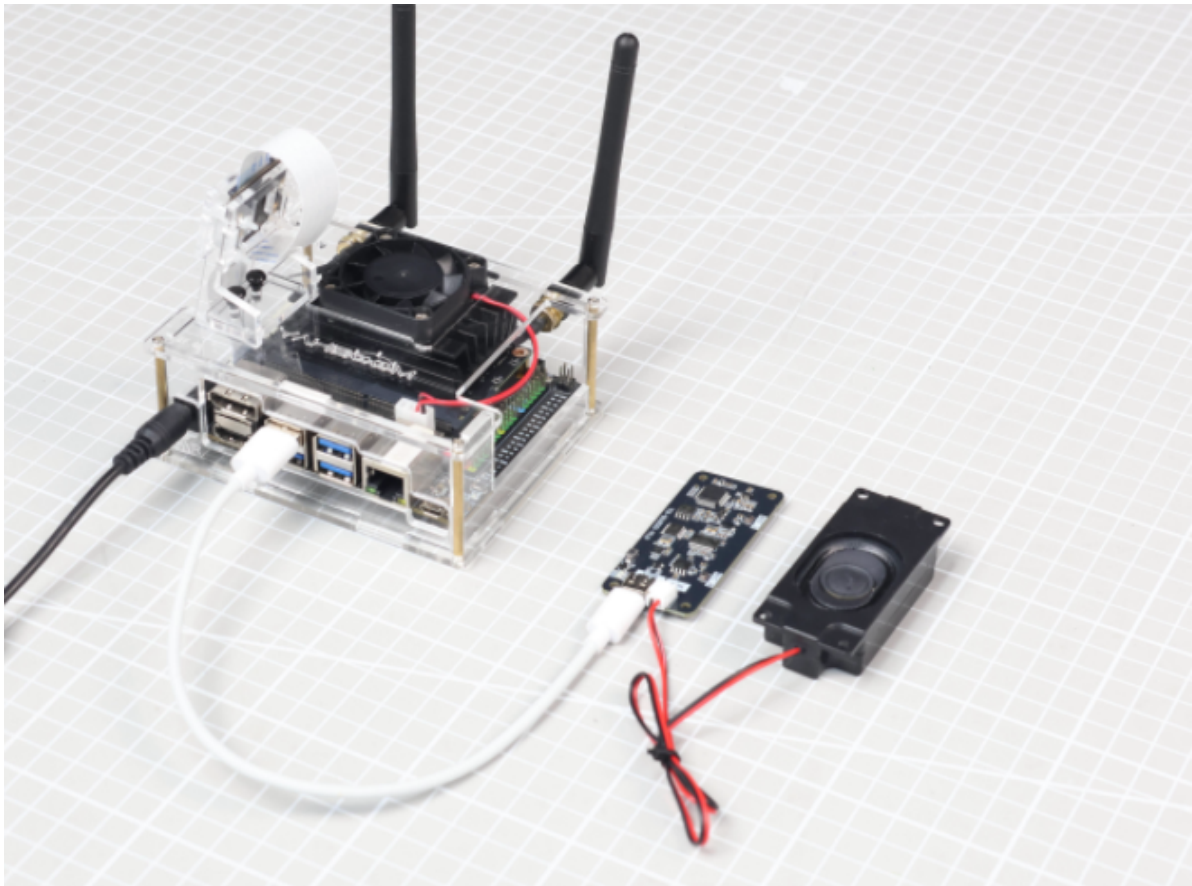
4. Confirm offline model path:

```
# large_model_interface.yaml
## 离线语音合成 (Offline TTS)
# 中文TTS模型 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"
•
# 英文TTS模型 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"
```

3. Start and Test the Functionality

Note: Due to the performance limitations of the Jetson Nano, the experience is limited. We recommend using the online version. After executing the offline text-to-speech command, the Jetson Nano will enter a suspended state while the model is loading. Please wait patiently for at least 2 minutes. If there is no response for more than 5 minutes, the Jetson Nano has crashed. Please power cycle.

Please ensure that the voice module is connected according to the diagram below.



1. Startup Command:

Note: The startup commands for CSI cameras and USB microphone cameras are different. Please run the command corresponding to your camera.

CSI Camera

Start UDP video streaming (host machine)

```
./start_csi.sh
```

Enter the CSI camera docker (host machine)

```
./run_csi_docker.sh
```

Start topic conversion (docker)

```
python3 ~/temp/udp_camera_publisher.py
```

Open another new terminal and check the container id

```
docker ps
```

According to the container ID shown above, change the container ID of the following command to the actual ID displayed, and enter the same docker with multiple terminals

```
docker exec -it container_id /bin/bash
```

Start the TTS node command (docker)

```
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
```

Open another terminal and check the container ID

```
docker ps
```

According to the container ID shown above, multiple terminals enter the same docker

```
docker exec -it container_id /bin/bash
```

Sending text to be synthesized (docker)

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

USB Camera

Enter the USB camera docker (host machine)

```
./run_usb_docker.sh
```

Start the TTS node command (docker)

```
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
```

Open another terminal and check the container ID

```
docker ps
```

According to the container ID shown above, multiple terminals enter the same docker

```
docker exec -it container_id /bin/bash
```

Sending text to be synthesized (docker)

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

2. **TEST:** If everything went well, 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

Problem 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.
2. In Settings, set the Sound Balance to the middle setting.

