End-to-end Speech Synthesis for Chinese-English Code-switching Scenario

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Science

in

Computer Sciences

at Massey University, Auckland, New Zealand

Qingci Zhang

2022

Abstract

Text-To-Speech (TTS), namely speech synthesis, allows machines to convert textual information into corresponding audio information by imitating human beings to produce human-like voices. TTS has been widely used in various monolingual speech synthesis tasks such as broadcasting systems and audiobooks. However, it is still a challenge for machines to process multilingual input and output sequences. Challenges may arise from the problem of the lack of code-switching speech data, the mapping problem of mixed languages, and the linguistic complexity of Chinese, such as polyphony and tonal sandhi scenarios in text frontend processing. In this thesis, we propose an end-to-end speech synthesis system based on a traditional monolingual Tacotron model to realize the speech synthesis of Chinese-English code-switching sentences. Firstly, we pre-process the speech data from the perspectives of low-frequency noise removal, frequency smoothness, and volume consistency by using a high-pass filter to smooth the speech frequencies ranging and normalizing the speech volume. Secondly, we apply g2pm and python-pinyin as our G2P tools which are merged into our mixed Chinese-English code-switching fronted processing. We solve the issue of language speaking failure and processing failure of the switched language of the current monolingual-support speech synthesis markup language, which is improved to be able to process mixed Chinese-English code-switching SSML input. We also further extend the rules of polyphone and tone sandhi of the Chinese part in code-switching sentences. Thirdly, we improve the attention mechanism module of the current Tacotron model to avoid the possible posterior collapse issue by transferring all intermediate frames to the next processing to keep the contextual correlation of adjacent frames, instead of only transferring the last frame, which will lose the context information. Fourthly, we accelerate the training process by adding a six-layer unidirectional sequence-to-sequence gated recurrent unit to predict more non-overlapping multi-frame outputs at each decoder step. The result of our test data reaches the highest score of 3.163 PESQ raw MOS and 3.065 MOS-LQO, and the average score of 2.672 PESQ raw MOS, and 2.520 MOS-LQO.

Keywords

TTS; speech synthesis; end-to-end; code-switching; polyphone; tone sandhi; G2P

Acknowledgement

I would like to express my most sincere gratitude to my supervisors Dr Yuanhang Qiu, Dr Zhizhong Ma and Professor Ruili Wang, for their invaluable guidance, encouragement and full patience throughout my research process.

I am also very grateful to Dr Feng Hou, Dr Ming Zong, Zhihan Wang, Satwind Singh, Yuan Gao, Long Shao, and Yujun Ma for their great help in sharing their learning experience.

I would also like to thank my parents and boyfriend. Their support and encouragement helped me a lot in completing my thesis.

Contents

1. Introduction

1.1 Motivation and Goal

The utilization of structured-system language to express thoughts is a distinctive feature of human beings (Dutoit, 2001). Usually, speech is one of the most direct and natural expressed forms of language in human communication (Liu et al., 2022). The exchange of information between people is mainly done through speech. Besides traditional dialogue communication in daily human life, there are a large amount of information data that exists in the form of speech, such as audio recording files, videos, and audio novels.

With the popularization of smart home appliances and smart devices based on the development and application of the Internet of Things (Denes et al.), communication between people and machines also tends to use speech. The demands for speech-based human-computer interaction technology are increasing rapidly such as Apple Siri, Xiao Du of Baidu, Xiao Ai of Xiaomi, Google voice assistant, Microsoft Cortana, and Amazon virtual assistant Alexa. Users can give some commands and inquiries to the machine through speech just like speaking to human beings. Correspondingly, the machine will give natural voice feedback and execute relevant instructions timely.

Speech interaction technology contains three main components including speech-to-text speech recognition, natural language processing, and text-to-speech speech synthesis. As an example of the human-computer speech interaction iteration, the speech information from human beings is converted to text information through speech recognition technology. After that, the text information is further processed and then transferred into acoustic data through speech synthesis technology as the generated acoustic output. Among them, speech synthesis technology plays a significant role in human-computer speech interaction tasks, which is our key research topic.

Speech synthesis, namely Text-To-Speech (TTS), converts text-form information into corresponding audio-form information, which allows machines to imitate human beings to produce human-like voices. According to different application requirements and scenarios, speech

synthesis is used in virtual assistants, audiobooks, the station broadcast of public transport, map navigation systems, and song synthesis tasks etc.

Considering the language of speech processing in TTS tasks, there are more than 7,000 languages in use around the world. In view of the rapid increase and development of globalization progress and cultural trade exchanges across the world, the utilization of multilingual and code-switching scenarios has become a common phenomenon in human life. Code-switching refers to the phenomenon of mixed or switched words or phrases of different kinds of language that occur inter-sentential locations or intra-sentential locations, which widely exists in multilingual communities who can speak multiple languages fluently (K. Li et al., 2019). Inter-sentential code-switching means that different languages switch at the boundary between the two context sentences. Intra-sentential code-switching means that different languages switch at the inner location of a sentence. Different from the unique switching location of inter-sentential code-switching, there are various possibilities of switching locations in intra-sentential code-switching scenarios. Therefore, intra-sentential code-switching tasks contain more unbalanced language distribution than inter-sentential code-switching scenarios (Shan et al., 2019).

As the universal language of the world, English is the most widely used language in 118 countries and regions all over the world, regarded as the dominant internet language, which is often mixed with other languages in various forms(Sreeja & Ilankumaran, 2020). For example, the group of Spanish-speaking people of Hispanic or Latino communities in America are proficient in both Spanish and English. They often switch these two languages, which forms a new Spanish-English code-switching language form called Spanglish (Ardila, 2005). Since English is one of the official languages of India, some Indian residents often mix English together with their local languages such as Hindi leading to a Hindi-English code-switching language called Hinglish (Ganji et al., 2019). In addition, Chinese is the most spoken native language with 1.3 billion native speakers due to the large population of Chinese and the relative consistency of the Chinese common language (Campbell, 2008). People in Chinese communities such as Singapore and Hong Kong who are proficient both in Chinese and English and often switch these two languages into a new mixed Chinese-English code-switching bilingual language while speaking. The Chinese-English code-switching scenario is shown in the following example sentences.

• Inter-sentential Chinese-English code-switching:

“Happy birthday to you! 许个愿然后吹蜡烛吧！”

(English Translation: Happy birthday to you! Make a wish and blow out the candles!)

• Intra-sentential Chinese-English code-switching:

“我刚刚去 Starbucks 买了杯 Vanilla Latte 和两块 Oatmeal Raisin Cookie, 搭配起来还蛮不错的。”

(English translation: I just went to Starbucks and bought a cup of Vanilla Latte and two Oatmeal Raisin Cookies, which went well together) (Zhao et al., 2020)

It is obvious that the inter-sentential code-switching sentence contains the English sentence ‘Happy birthday to you’ and the Chinese sentence ‘许个愿然后吹蜡烛吧’ with a unique switching location between two sentences. Accordingly, it is easy to synthesize the speech of inter-sentential code-switching sentences by regarding the sentences as two independent monolingual sentences separately. The boundary between two different language sentences can be regarded as a signal for the speech synthesis system to switch between two independent monolingual models in a traditional way. Namely, the traditional monolingual English speech synthesis model synthesizes the front sentence, and the traditional monolingual Chinese speech synthesis model synthesizes the behind Chinese sentence. However, this limited method may cause considerable speech inconsistency issues, and may lose the application and adaptability to solve practical code-switching scenarios with a large amount of intra-sentential code-switching sentences.

By contrast, the switching location distribution of the intra-sentential code-switching sentence is unbalanced and unpredictable, which increases the complexity and difficulty of speech synthesis compared with inter-sentential code-switching sentences. For instance, the example intra-sentential code-switching sentence contains the English words and phrases ‘Starbucks’, ‘Vanilla Latte’ and ‘Oatmeal Raisin Cookie’, which are separately inserted in different locations inside a Chinese sentence body. The current traditional monolingual speech synthesis system is inadaptable to this scenario.

However, most of the current speech synthesis systems focus on monolingual speech, which can only process only one language at a time. Such systems are unable to well synthesize speech in mixed code-switching scenarios. The error or inaccurate output of speech synthesis may result in interaction failure by spreading and affecting various related modules in the human-computer interaction system. Moreover, the code-switching scenario is a considerable challenge due to the high cost and difficulty of collecting high-quality code-switching data compared with existing various monolingual TTS works.

Especially, in the Chinese-English code-switching scenario, the linguistic characteristic and complexity of Chinese are considerable issues when dealing with Chinese speech processing. For example, the grapheme-to-phoneme (G2P) processing of the Chinese text frontend is important when facing polyphone and tone sandhi variations of Pinyin. As an abbreviation of Hanyu Pinyin, Pinyin refers to the spelt sounds of Standard Mandarin Chinese, which is an official romanization system written with the Latin alphabet-based initials and finals equipped with different tones. There are 23 pinyin initials and 40 pinyin finals. Pinyin initials include the flat tongue, warped tongue, and zero initials. Pinyin finals include single finals, complex finals, front nasal finals, rear nasal finals, and overall recognition syllables.

Moreover, the tones in Pinyin can be divided into 5 types, including Yinping, Yangping, Shangsheng, Qusheng and Qingsheng, namely four diacritics denoting tones and one without tone (Liu et al., 2021). For example, the corresponding pinyin of ‘我爱打乒乓球’ should be ‘$w\check{o}\ \grave{a}i\ d\check{a}\ p\bar{\imath}$ng $p\bar{a}$ng $qi\acute{u}$’.

However, polyphone and tone sandhi variations of pinyin may happen in some special scenarios. Polyphone refers to one Chinese character that may have different pronunciations of pinyin in different situations. Sometimes, the variation refers to different tones or refers to totally different initial or final syllables. Below are examples of polyphone and tone sandhi variation scenarios of Chinese monolingual sentences:

• Polyphone and tone sandhi of one particular character:

一班一共一百人都还待在一起，你不要不还钱。

yī bān yí gòng yì bǎi rén dōu hái dāi zài yì qǐ nǐ bú yào bù huán qián

• The third tone sandhi:

李老板岂有此理，展览馆只有五种产品。

Bad: *lǐ lǎo bǎn qǐ yǒu cǐ lǐ zhǎn lǎn guǎn zhǐ yǒu wǔ zhǒng chǎn pǐn*

Good: *lǐ láo bǎn qí yǒu cí lǐ zhán lán guǎn zhí yǒu wú zhǒng chán pǐn*

As shown in the example sentence of polyphone and sandhi, the character '还' has different pronunciations *hái* and *huán*. One character may have different tones in different scenarios such as '一' has three different tones *yī*, *yí*, and *yì*, and '不' has two different tones *bú* and *bù*. As for the example sentence of the third tone sandhi namely three sandhi, all independent characters should be the third tone separately like the bad example. However, as shown in the good example,

part of the third tone should be changed into the second tone in the real Chinese speech scenario when combined with other third-tone characters behind them, which leads to tone sandhi.

Obviously, the combinations of Chinese graphemes and phonemes are more complex than those in English, which increases the difficulty of G2P in Chinese-English code-switching TTS tasks compared with those in monolingual English TTS tasks. In the process of G2P, the phoneme sequences of each word are determined which can be combined with other speech synthesis parameters through speech synthesis markup language (SSML) as input to generate the final speech synthesis as TTS output. SSML is an XML-based speech synthesis markup language, which can enrich the content and final changes of the synthesized speech compared with plain text. Currently, only Chinese and English voices support the SSML of Aliyun TTS based on version 1.1 of W3C SSML. However, the current speak root element of SSML cannot be executed when processing speech synthesis of mixed code-switching Chinese-English text, because the mixed Chinese and English text will be classified as monolingual English by the SSML element placed at the front end of the text for language judgment. Therefore, the synthesis processor encounters new character sequences that it cannot process rather than the previously selected or declared language of the text. Accordingly, language speaking failure of the synthesis processor will occur taking the actions such as changing voice, ignoring text, and ignoring language which is called onlangfailure. Namely, as onlangfailure attribute, it contains one value of the desired language speaking failure behaviours. The following Table1 shows the desired language speaking failure behaviours of the synthesis processor namely the values for onlangfailure attribute and the corresponding description.

Table1. Values for onlangfailure attribute of speak element in SSML.

Values for onlangfailure attribute	Description
Changevoice	The synthesis processor will apply ignoretext or ignorelang when there is no voice available to speak the language.
Ignoretext	The synthesis processor will not render the text if the failed language is not supported.

Ignorelang	The synthesis processor will disregard the code-switching newly occurred language without breaking and render the content continuously pretending the contents were always in the same prior monolingual language.
Processorchoice	As the top-level default value for this attribute, the synthesis processor will choose one of the actions from changevoice, ignoretext, and ignorelang.

Therefore, Therefore, our motivation comes from three perspectives including the practical demands of TTS in Chinese-English code-switching scenarios, the challenge of TTS due to the complexity of intra-sentential code-switching sentences and Chinese polyphone and tone sandhi conditions, and the language speaking failure issue when processing speech synthesis of mixed code-switching Chinese-English text. Accordingly, our goal is to implement a Chinese-English code-switching speech synthesis system to achieve good performance in intra-sentential code-switching scenarios and Chinese polyphone and tone sandhi scenarios.

1.2 Contributions

Different from common monolingual speech synthesis, we solve the processing challenge of dealing with mixed Chinese-English code-switching text input. The speak root element of the SSML of Aliyun TTS based on version 1.1 of W3C SSML cannot be executed resulting in language speaking failure when processing speech synthesis of mixed code-switching Chinese-English text, because the mixed Chinese and English text will be classified as monolingual English by the SSML element placed at the frontend of the text for language judgment. We have solved this issue when processing speech synthesis of mixed code-switching Chinese-English text by adding the content of the selected element to the target array, merging the part of Chinese text into the target element tag to avoid the potential intra-sentence pause of the synthesized speech, and processing the Chinese text part separately based on the target element tag.

Polyphone and tone sandhi scenarios of Chinese are also considered in our frontend processing. We applied g2pm[1] and python-pinyin[2] as our G2P tools which are merged into our mixed Chinese-English code-switching fronted processing. We have further extended the rules of tone sandhi of Chinese character '不' and '一', and added some new three sandhi rules.

The attention mechanism module of the current Tacotron model tends to cause the posterior collapse issue by only transferring the last frame to the next processing step without considering the contextual correlation of adjacent frames. We improve the current Tacotron model by transferring all intermediate frames to the next processing step.

Considering we have added the processing intermediate frames, we accelerate the training process by adding a six-layer unidirectional sequence-to-sequence gated recurrent unit to predict more non-overlapping multi-frame outputs at each decoder step at the same time, which is also different from the current Tacotron model. Moreover, we apply Adam with a warm-up learning rate to speed up the loss convergence of the attention mechanism.

[1] https://github.com/kakaobrain/g2pM
[2] https://github.com/mozillazg/python-pinyin

The complexity and variation of Chinese linguistic special cases such as polyphone and tone sandhi in Chinese-English code-switching scenarios present higher challenges compared to the traditional monolingual speech synthesis model. The implementation of our end-to-end TTS system in the Chinese-English code-switching scenario can solve the code-switching issue, especially in intra-sentential conditions, and can achieve good performance in Chinese linguistic polyphone and tone sandhi scenarios.

Our end-to-end Chinese-English code-switching speech synthesis system further explores speech synthesis based on current research. Moreover, it can be applied in map navigation systems containing both Chinese and English location names, international airport broadcasts requiring automatic speech broadcasts, bilingual audiobooks containing Chinese-English code-switching sentences, and automatic reservations for restaurants and hotels. Therefore, our research on speech synthesis of Chinese-English code-switching scenarios is important from the perspectives of the academic sense and application value.

1.3 Organization of the Thesis

The rest of this thesis is organised into five parts:

- Chapter 2 reviews the background and related work of this thesis. After introducing the development history of TTS, the applications of speech synthesis and the G2P process in code-switching scenarios are also discussed.
- Chapter 3 introduces the datasets and acoustic modelling units we have used in our experiment.
- Chapter 4 describes the main method of the end-to-end TTS system after the introduction of related basic principles. We also introduce the method selection of optimization approach for gradient descent to minimize the loss during the training process. The Evaluation method of our experiment is also provided.
- Chapter 5 shows the experimental settings and results. Experimental environment and pre-processing are also introduced. After the detailed experiment setup, we give and analyze the experimental results.
- Chapter 6 summarizes the entire work, provides the general conclusions and presents our future work plans.

2. Background and Related Work

In this chapter, we review the background and related work of speech synthesis. The detail of this chapter is structured as follows. Section 2.1 reviews the development history of TTS. Section 2.2 introduces the application of speech synthesis in code-switching scenarios. Section 2.3 discusses the G2P process in code-switching TTS.

2.1 Development History of Speech Synthesis

Speech synthesis converts text information into corresponding audio information, which allows machines to imitate human beings to produce a human-like voice. There are various application scenarios of speech synthesis, such as virtual assistants, audiobooks, station broadcasts of public transport, map voice navigation, and song synthesis. As the first attempt to electronically synthesize human speech based on acoustic components, Voder was invented in 1938, which provided limited quality of synthesized speech by operating the complex machine equipped with touch-sensitive keys and a pedal (Gold et al., 2011). George Rosen introduced the first articulatory synthesizer invented in 1958 (Kaur & Sharma, 2016). The earliest computer-based sing voice synthesis system was explored at Bell Labs in 1961 with an IBM 704 (Bendel, 2019).

The current mainstream methods of speech synthesis are divided into waveform splicing method, statistical parameters method, hybrid method and end-to-end neural network method. The end-to-end approach includes the acoustic model plus vocoder, and full end-to-end methods. The speech synthesis pipeline includes three main modules: text frontend module, acoustic model and vocoder module, in which the text frontend module converts raw text to characters or phonemes, and the acoustic module converts characters or phonemes into acoustic features such as linear spectrograms, Mel spectrograms, and LPC features, and those acoustic features are further processed by the vocoder to generate waveforms.

As one of the traditional algorithm for speech synthesis, the pitch-synchronous overlap-add (PSOLA) approach for speech synthesis was proposed, which refers to the process of selecting the most suitable speech unit part from a library of existing recorded speech, using the unit adjustment

algorithm to adjust the selected unit appropriately, and finally splicing it into speech (Charpentier & Moulines, 1989). In 1996, Hunt et al. proposed an refined optimized unit selection strategy in a concatenative speech synthesis system with a large number of speech data, which improved the naturalness of synthesized speech (Hunt & Black, 1996). The concatenative speech synthesis approach achieves the naturalness and high quality of the synthesized speech. However, the good performance is significantly limited by the corpus, which leads to a lack of robustness and poor scalability. Moreover, speech components can only be synthesized from the existing corpus without creativity. Due to the development of statistical modelling and machine learning technology in speech processing, researchers have proposed parametric speech synthesis methods such as HMM-based speech parameter generation algorithms for speech synthesis (Tokuda et al., 2000). The core idea of statistical parameter speech synthesis is to use statistical models and the features of the input text to model the acoustic parameters of speech, such as the distribution of the fundamental frequency. Namely, the acoustic model is trained based on machine learning technology with the text features as input and acoustic parameters as output to estimate and predict the parameters distribution of the acoustic features. According to this method, although the quality and flexibility in changing the identity and emotion of speakers as well as speaking styles of synthesized speech have been improved, the naturalness of synthesized speech is still a considerable challenges compared to the emotional expressions, emphasis and prosody of actual human speech (Tokuda et al., 2013).

The traditional speech synthesis technology is highly complex and inefficient, which lead to the development of deep learning-based end-to-end TTS systems (Mu et al., 2021). The earliest work exploring the attention-based end-to-end TTS is proposed in 2016, which requires a pre-trained HMM to help with alignment and a vocoder (Wang et al., 2016). In 2017, an end-to-end model for speech synthesis called character-to-waveform (Char2Wav) was proposed, which can produce audio speech from text directly (Sotelo et al., 2017). Char2Wav consists of two parts, one is the reader with an attention-based encoder-decoder model and the other is the SampleRNN neural vocoder to generate waveforms from the intermediate representation (Mehri et al., 2016). Subsequently, Baidu proposed Deep Voice for speech synthesis, which plays an important role in the groundwork for truly end-to-end TTS (Arık et al., 2017). Deep Voice is mainly composed of five blocks: boundary segmentation, conversion from grapheme to phoneme, phoneme-duration

prediction, prediction of the fundamental frequency, as well as audio synthesis. However, it is difficult to find which of the five modules has an error because each component is independently trained, which may lead to the transmission and accumulation of errors.

In 2017, Google proposed a content-based attention end-to-end speech synthesis model called Tacotron, which can be trained from scratch with random initialization to obtain Mel spectrograms directly from characters (Wang et al., 2017). Tacotron uses Griffin-Lim as vocoder to generate synthesized waveforms from the predicted spectrogram. Before Tacotron, there have also been previous studies on sequence-to-sequence models. However, the attention mechanism of those models still introduces supervised information such as segmentation duration in aligning input and output sequences, and the effect does not exceed traditional methods. The Tacotron model is the first model reported in the paper to achieve a higher naturalness degree of synthesized speech than traditional parametric language synthesis methods, which inspires and influences the following work on sequence-to-sequence acoustic models.

In 2018, on the base of Tacotron, Google proposed Tacotron2 (Shen et al., 2018), which replaces the complex post-processing network CBHG and GRU recurrent layers with LSTM and convolutional layers. It applies a position-sensitive attention mechanism (Raffel et al., 2017) to improve alignment stability and uses WaveNet as vocoder. WaveNet works in speech synthesis as a replacement for vocoder and acoustic model requiring linguistic features of the previous audio sample, which improves the naturalness of synthesized speech (Oord et al., 2016). Compared to WaveNet, the synthesis speed is increased using WaveRNN, which applies a single-layer RNN with a sparse matrix to the reduction of calculation (Kalchbrenner et al., 2018). Furthermore, in 2019, as a variant of WaveRNN combining linear prediction with RNN, LPCNet synthesizes higher quality of speech in a higher efficient way than WaveRNN (Valin & Skoglund, 2019). In 2019, Transformer was used in the field of speech synthesis, which can speed up the training processing equivalent to Tacotron 2 (N. Li et al., 2019). The transformer only depends on the input and output of the attention mechanism without CNN and LSTM (Vaswani et al., 2017).

2.2 Speech Synthesis in Code-switching Scenarios

There are three ways to deal with multilingual text. The first way is to record a multilingual training corpus through one source speaker and train each language separately. When performing speech synthesis, it is necessary to segment text and switch different models, which may lead to unnatural performance. The second way is to mix multilingual corpus for the training of multilingual models, which can avoid switching models as well as the unnatural phenomenon of switching between languages. Both methods have a high requirement on the source multilingual speaker. The third way is the utilization of multi-person monolingual corpus to train multilingual models, which is easier and cheaper than the above two methods.

There are various related studies for multilingual speech synthesis. Zen et al. suggested using a speaker and language separation approach to transfer a voice from one language to others in an HMM-based parametric TTS system (Zen et al., 2012). In 2016, a parametric neural TTS system was suggested that could support multiple languages by utilizing a single input representation and shared parameters, but with separate voices for each language (Li & Zen, 2016). In 2017, a neural TTS system that is bilingual in Chinese and English was proposed to generate bilingual speech with the same voice by the utilization of training on the speech from a bilingual speaker (Ming et al., 2017). In 2019, Zhang et al. propose a framework for cross-language voice cloning and speech synthesis with three parts: inference network, domain adversarial training, and synthesis network. Language embedding and speaker embedding are added in the Tacotron2-based synthesizer for multiple languages and multiple speakers (Zhang et al., 2019). Regarding the limited-data scenario in the field of cross-lingual multi-speaker speech synthesis, language token is embedded with phoneme sequence as the encoder input in Tacotron2 (Cai et al., 2020). In the encoder part of Tacotron2, convolutional layers are used instead of BiLSTM, which leads each language to have its encoder separately (Nekvinda & Dušek, 2020). Moreover, a parameter generator is added to deal with the language IDs, which is embedded with each layer of the encoder through multi-layer FC. With regard to the process method for multilingual input, Zhou et al. use a pre-trained language model to generate cross-lingual words which can encode bilingual words within the same embedding space sharing contextual information. Especially, when synthesising Japanese speech, the phonetic and prosodic information of Japanese is complex. Polyphone disambiguation, segmentation between words, and accent nuclear shift are considerable issues in Japanese speech

synthesis (Kakegawa et al., 2021). In order to enable multilingual input in a unified format, the international phonetic alphabet (IPA) as the unified input set can be converted into a voice feature with many characteristics (Hemati & Borth, 2020). The application of phonetic posteriorgrams (PPGs) can cross the boundaries between speaker and language which leads to improve naturalness and solve the mismatch issues of the phonetic set and the speech prosody (Zhao et al., 2021). Researchers also extend Tacotron2 to realize multilingual multi-speaker speech synthesis by minimizing the mutual information of the speaker embedding and language embedding (Xin et al., 2021). Moreover, the speech chain provides another perspective to combine speech synthesis and speech recognition, leading to performance improvement by learning from each other without parallel code-switching data (Nakayama et al., 2018). Similarly, two-stage training improves controllable speech synthesis by comparing and minimising the differences between the output text of speech recognition and the input text of speech synthesis (Liu et al., 2018).

Furthermore, besides the traditional monodirectional TTS pipeline, the ASR-TTS dual learning speech chain combines both speech recognition and speech synthesis to obtain better performance than the traditional separately TTS training process with limited labelled code-switching data. ASR refers to automatic speech recognition, which aims to convert audio information to corresponding text information. The combination of ASR and TTS as a speech chain is proposed in 2017, which can achieve better performance than separated traditional TTS and ASR models by jointly training both ASR model and TTS model through a loop connection (Tjandra et al., 2017). It is also utilized in speech-to-speech translation tasks for Javanese, Sundanese, Balinese, and Bataks languages (Novitasari et al., 2020). After supervised training of ASR and TTS models in advance with standard Indonesian, the cross-lingual speech chain is applied with unlabelled target ethnic language data. Nakayama et al. (2018) also construct sequence-to-sequence semi-supervised learning speech chain for Japanese-English code-switching ASR and TTS without parallel code-switching data. On the base of the supervised training of ASR and TTS models, unpaired code-switching data is processed through the speech chain in unsupervised learning (Nakayama et al., 2018).

2.3 G2P in Code-switching Speech Synthesis

G2P stands for Grapheme-to-Phoneme, which can be described as the mapping process of converting the written spelling form of a word into its corresponding phonetic pronunciation representations (Yi et al., 2009). This conversion process determines the appropriate phonetic pronunciation of the word on the basis of phonological rules and patterns of the language through G2P converter tools. The application of G2P tools plays an important role in text-to-speech synthesis to help computers to better process the textual and audio data of different languages, thereby improving the accuracy and naturalness of spoken language output.

As for the code-switching speech synthesis, the combinations of Chinese graphemes and phonemes are more complex than those in English, which increases the difficulty of G2P in Chinese-English code-switching TTS tasks compared with those in monolingual English TTS tasks. Considering the pronunciation of polyphonic characters can be determined by the context, Cai et al. (2019) introduces different word-level embedding and sentence-level embedding to achieve Chinese polyphonic disambiguation (Cai et al., 2019). Polyphone disambiguation can be regarded as a classification problem, which needs a classifier to predict the corresponding pronunciation of each character in the given context (Yang et al., 2019). Shan et al. (2016) determine the pronunciation of the Chinese polyphonic word depending on the Part-of-Sentence (POS) of the polyphonic word in the sentence and the POS of the context (Shan et al., 2016). The representation of input is also considerable in the multilingual G2P for different languages. Yu et al. (2020) utilize byte-level input representation in their proposed multilingual G2P model to adapt to grapheme systems of five selected languages. The byte-level input representation performs better than the character-based model because the limitation of byte-level vocabulary cardinality is constrained to no more than 256, which can improve the accuracy of the model by reducing the vocabulary sparsity (Yu et al., 2020).

In summary, this section reviews the background and related work of speech synthesis from the perspectives of the development history of TTS, the applications of TTS in code-switching scenarios, and the G2P gaps in Chinese-English code-switching TTS tasks. Deep learning technology is utilized in the field of speech synthesis instead of the high-complexity and low-efficiency traditional methods based on statistical parameters and waveform splicing. As for the

Chinese-English code-switching speech synthesis, the complex combinations of Chinese graphemes and phonemes, polyphone and tone sandhi scenarios increase the difficulty of G2P compared with those in monolingual English TTS tasks.

3. Dataset and Acoustic Modelling Units

This chapter introduces the dataset used in our experiment and acoustic modelling units in text frontend. Dataset D_{CS}^{P} is a labelled Chinese-English code-switching dataset based on the TAL_CSASR[3] corpus. The TAL_CSASR corpus is collected from teachers who speak bilingually in Chinese-English code-switching language. This corpus consists of 587-hour audio sampled at 16 kHz and 16 bit, and the audio files formatted as wav, transcript label files formatted as txt. In each line of the transcript, there is a unique ID number equipped with a Chinese-English code-switching sentence containing both Chinese characters and English words in capitalized format. Correspondingly, each ID number is equipped with an acoustic audio wave file, which combines the audio sentence together with the transcript label text.

There are mixed bilingual code-switching text sequences which are significantly different from common monolingual acoustic modelling units. The acoustic modelling unit of English includes subwords and characters. The acoustic modelling unit of Chinese includes characters and syllables with tones, which is much more complicated than English. Chinese acoustic modelling units are represented by pinyin.

Yinping, Yangping, Shangsheng, and Qusheng, those four pinyin tones can be represented by four numbers from one to four. Qingsheng refers to syllables without any numbers. The number representation can be customized, for example, the number 5 refers to Qingsheng. Besides the number representation of tones, there are other components that can be customized, such as the space segmentation between initials and finals, the substitution between y and i, w and u, and u and $\ddot{u}$. Besides the basic character-to-pinyin, the text frontend also includes text segmentation, text normalization, word segmentation, prosody, conversion from grapheme to phoneme, polyphone, and tone sandhi. Before input to the acoustic model, the phoneme sequence needs to be further processed. The most important of these modules are the text regularization module and the grapheme-to-phoneme (G2P) module. The following Table 1 shows examples of Chinese text frontend with different modules.

[3] https://ai.100tal.com/dataset

Table1. Polyphone examples of Chinese text frontend.

Text frontend modules	Example
Input text	一共 100 个人都在一起
Text normalization	一共一百个人都在一起
Grapheme-to-Phoneme (G2P)	i2 g ong4 i4 b ai3 g e5 r en2 d ou1 z ai4 i4 q i3
Original form of pinyin tone	*yí gòng yì bǎi gè rén dōu zài yì qǐ*

In the G2P example, polyphone phenomena is illustrated through different pronunciation of the same character '一'. Besides polyphone phenomena, tone sandhi is also a considerable issue. The following Table 2 shows the example of the third tone sandhi.

Table 2. Tone sandhi examples of Chinese text frontend.

Text frontend modules	Example
Input text	李老板岂有此理，展览馆只有 5 种产品
Text normalization	李老板岂有此理，展览馆只有五种产品
Grapheme-to-Phoneme (G2P)	L i3 l ao2 b an3 q i2 iou3 c ii2 l i3 zh an2 l an2 g uan3 zh i2 iou3 u2 zh ong3 ch an2 p in3
Original form of pinyin tone	*lǐ láo bǎn qí yǒu cí lǐ zhán lán guǎn zhí yǒu wú zhǒng chán pǐn*

Although all the characters should be the same third tone independently, part of them should be converted to the second tone before the next one, which leads to tone sandhi.

Both grapheme and phoneme are representations used in TTS. Grapheme refers to the smallest representation unit of written language. Phoneme refers to the smallest representation unit of spoken language. Namely, the representation 'th' of grapheme means the letter combination of 'th' that are used to form word 'they'. The same representation 'th' of phoneme refers to the

pronunciation sound ‘th’ in word "think". Besides grapheme and phoneme, UTF-8 encoded byte is also a format to represent text using numerical codes of the corresponding characters. We also explore the byte-based representation of Chinese tone sandhi word. The following Table 3 shows the example of byte representation and pinyin representation of Chinese tone sandhi words.

Table 3. Byte representation and pinyin representation examples for Chinese tone sandhi words.

Chinese tone sandhi word	Byte representation (UTF-8)	Pinyin with tone
展览馆	0xE5 0xB1 0x95 0xE8 0xA7 0x88 0xE9 0xA6 0x86	zhan2 lan2 guan3
展览	0xE5 0xB1 0x95 0xE8 0xA7 0x88	zhan2 lan3

It is clear that the byte representation of ‘zhan2 lan2’ is the same as ‘zhan2 lan3’ ignoring the complex Chinese third-tone sandhi scenarios compared to the pinyin representation.

4. Method

This chapter is divided into three sections to introduce the end-to-end TTS framework for the Chinese-English code-switching corpus. The first section 4.1 gives some basic principles of speech synthesis model from the perspectives of sequence-to-sequence model, end-to-end speech synthesis process, feature extraction, acoustic model and vocoder. After that, section 4.2 introduces the encoder-decoder model, post-processing net module and waveform synthesis module of the Tacotron model. The following section 4.3 introduces the method selection of optimization approach for gradient descent to minimize the loss during the training process.

4.1 Basic Principles of Speech Synthesis Model

This part will introduce some basic principles of speech synthesis model from the perspectives of early sequence-to-sequence model, attention-based sequence-to-sequence model, end-to-end speech synthesis process, feature extraction, acoustic model, and vocoder.

4.1.1 Early Sequence-to-sequence Speech Synthesis Model

Sequence-to-sequence models were first used in handwritten text recognition tasks, and later in sequence modellings such as ASR and machine translation. Then, sequence-to-sequence models were used in speech generation models, including speech synthesis model, voice conversion model and articulation-to-text model. Those models generate the target speech Y with different input data X as shown in Figure 1. For example, the sequence-to-sequence speech synthesis model contains the audio representation domain Y, and text representations domain X. The text sequence x (array of phonemes or graphemes) belongs to the X domain with a defined length S as $x = (x_1, x_2, \dots, x_S)$ & $x \in X$. The audio sequence y (Mel-spectrogram as features) belongs to the Y domain with a defined length T as $y = (y_1, y_2, \dots, y_T)$ & $y \in Y$. As a part of speech generation models, the speech synthesis model deal with the input of text feature $x = (x_1, x_2, \dots, x_S)$ & $x \in X$, and generate the corresponding target speech $y = (y_1, y_2, \dots, y_T)$ & $y \in Y$ as output. The

sequence-to-sequence model attempts to describe the conditional probability of the entire output sequence $P(y_{1:T}|x_{1:S})$ when given the input sequence.

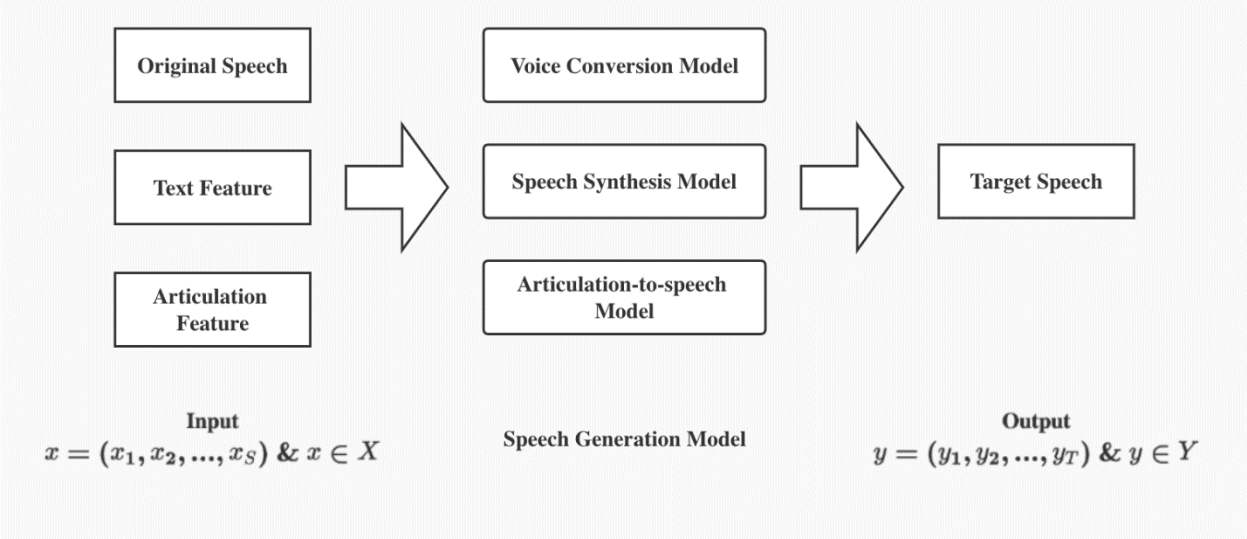

Figure 1. Sequence-to-sequence speech generation tasks.

The sequence-to-sequence model is based on a structure with an encode and a decoder, which aims to describe the correspondent relationship between the input sequence of the encoder and the output sequence of the decoder without any restriction on the length of the input and output sequences. Namely, the input and output can be frame-by-frame aligned feature sequences, such as audio and corresponding synchronized video sequences, or misaligned features between input text and output speech such as the speech synthesis task. There is no detailed definition of the neural network structure of sequence-to-sequence models. According to the different requirements of tasks, some components of the structure can be flexibly constructed with the utilization of neural networks such as RNN as shown in Figure 2. The role of the RNN encoder is to convert an input sequence with variable lengths into an output sequence with fixed and constant lengths in the hidden layer as shown in the following Figure 3.

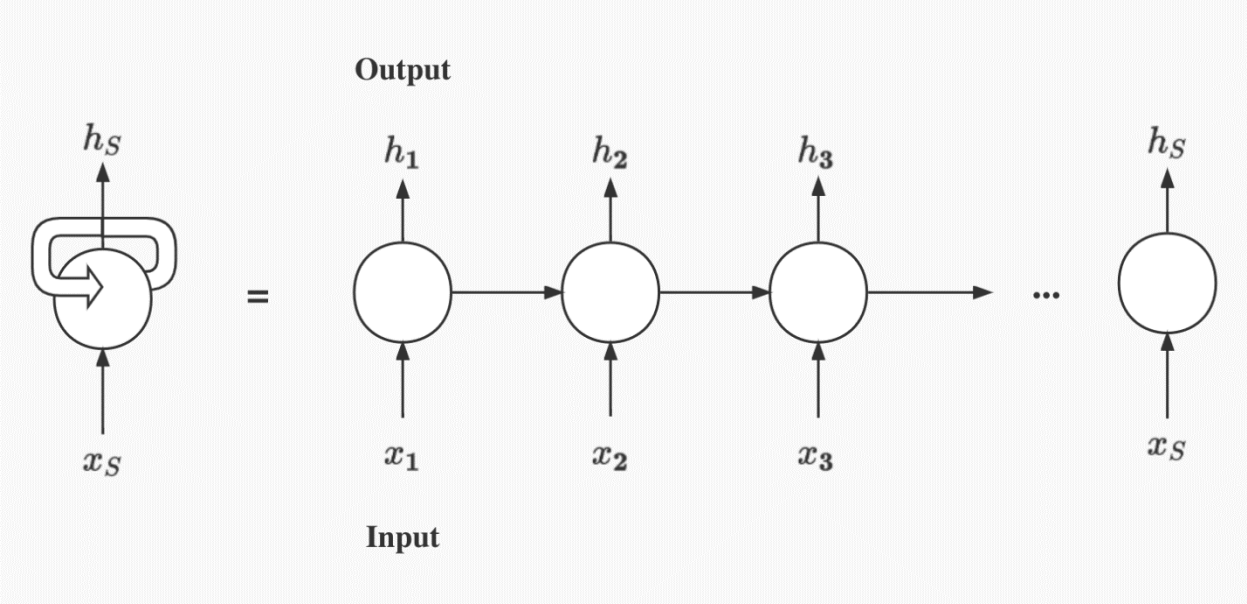

Figure 2. Recurrent neural network.

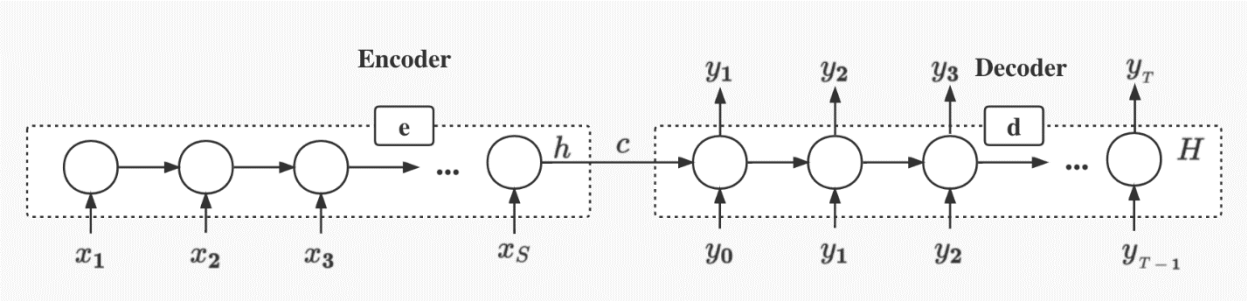

Figure 3. RNN sequence-to-sequence model.

The recurrent neural network sequentially processes the input signal by producing the corresponding output signal at each time step. This structure can encode and process the contextual information of the input sequence signal. The model has a different hidden layer state h at each moment S, which is determined by previous signals. We can use function f in the following equation to describe the transformations made by the recurrent layers of RNN:

$$h_S = f(x_S, h_{S-1})$$

The encoder RNN applies function e to transfer all the hidden states into contextual variables:

$$c = e(h_1, \dots, h_S)$$

The hidden state of the encoder at the last time step is usually used as the initial decoder hidden state, which means that the RNN encoder and decoder should have the same layer number as well as the same unit number of hidden state. The prediction of the decoder output y_T depends on the previous output sequences and the contextual variable c from the encoder output, namely $P(y_T|y_1, y_2, \dots, y_{T-1}, c)$. The decoder utilizes another recurrent neural network with function d to make this transformation in the hidden layers of decoder as shown in the following equation:

$$H_T = d(y_{T-1}, c, H_{T-1})$$

The context variables are connected with the decoder input. On the basis of the hidden status H_T and c, we can get the probability distribution of output y_T through the SoftMax function at the fully connected layer of the last layer of decoder's neural network.

4.1.2 Attention-based Sequence-to-sequence Speech Synthesis Model

The attention mechanism refers to dynamically selected for processing, which is the motivation proposed by In the decoding process, a part of the input sequence is dynamically selected for processing as shown in Figure 4. Early sequence-to-sequence models encoded the entire input sequence into a hidden layer vector, and then used a decoder to decode the output sequence information from this vector. However, much dynamic and contextual information of context is lost in the encoding process. It is difficult to express the multiplication of the entire sequence in a single vector, especially as the length of the input sequence becomes longer. Therefore, from a more efficient point of view, each moment of decoding on the decoder side does not require the entire information of the input sequence, but only needs to pay attention to the smaller-range local information. For example, in speech synthesis, synthesizing the acoustic features of the corresponding phoneme only needs to focus on a small context of the current phoneme in the input sequence.

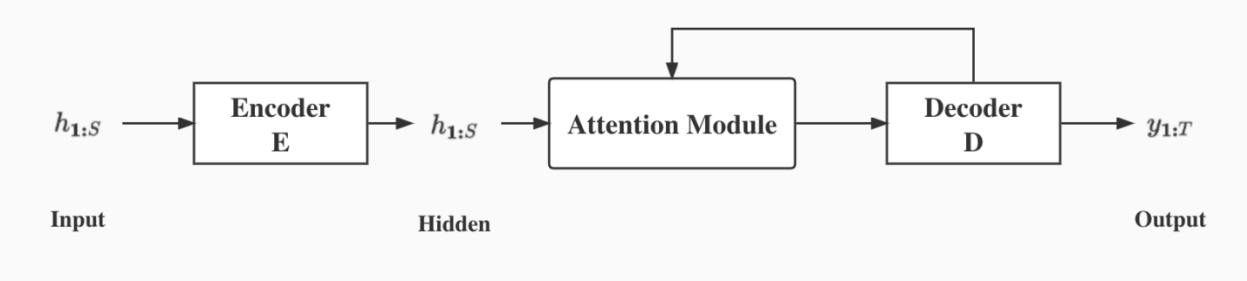

Figure 4. Attention-based sequence-to-sequence model.

4.1.3 End-to-end Speech Synthesis Process

End-to-end speech synthesis technology is a process of mapping text information to speech spectral feature vectors through neural network models, and then restoring them to speech signals through spectral features. The whole process can be summarised as speech feature extraction and deep learning modelling. As for the traditional TTS pipeline, it can be processed through three major modules: text frontend module, acoustic model and vocoder module as shown in Figure 5. The text frontend module can convert original raw text to characters/phonemes. The acoustic module transfers the given characters or phonemes from the previous stage into the corresponding acoustic features, such as linear spectrograms, Mel spectrograms, and LPC features. The vocoder module converts the given acoustic features from the previous stage to target audio waveforms.

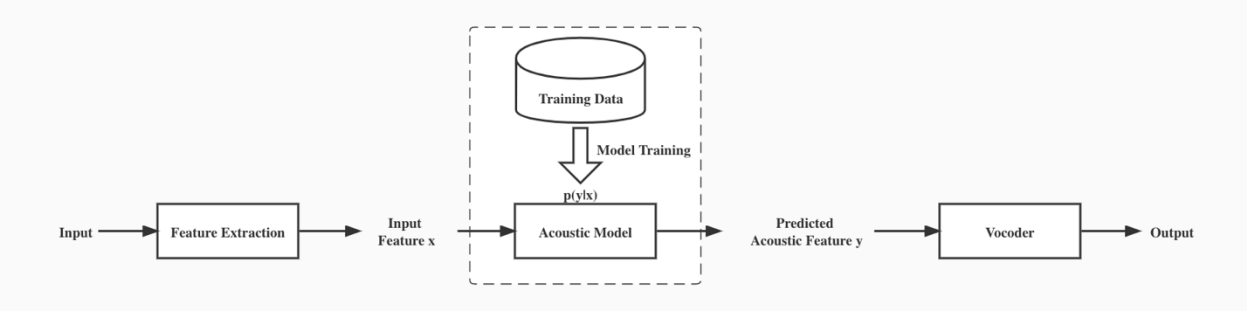

Figure 5. End-to-end speech synthesis process.

4.1.4 Feature Extraction

The feature sequence of the speech can determine the amount of information related to the audio content that the model can receive. Therefore, the selection of speech features is very important. Good speech recognition features not only need to reflect the content of speech but also reflect the

auditory characteristics of the human ear. Because the sound-perception ability of human beings is nonlinear, and as the frequency of the sound increases, the degree of discrimination of higher frequency sounds will continue to decline. For example, the same frequency differs by 400Hz. The average person can easily distinguish the difference between 600 Hz and 1,000 Hz in sound, but it is difficult to distinguish the difference between 10,000 Hz and 10,400 Hz. Therefore, the Mel frequency is proposed to perceive the same value of frequency changes to the same degree as human ears. MFCC and FBank features are often used in speech recognition, and logarithmic transformation is performed on the features.

For a piece of audio, the entire audio is generally divided into frames, each frame contains a certain length of signal data, generally 25ms, the moving distance between frames is called frameshift, generally 10ms, and then for each frame. After windowing the signal data, the discrete Fourier transform (DFT) is performed to obtain the spectrogram. After framing a piece of audio according to the above, we can use Fourier transform to analyze the frequency characteristics of each frame of signal. After splicing the frequency information of each frame, the frequency characteristics of the audio at different times called spectrogram can be obtained.

The extraction process of FBank features includes speech signal sampling, pre-processing, pre-emphasis, framing processing, windowing processing, DFT, and Mel filter bank.

- Speech signal sampling

  Because the voice signal needs to be converted into a digital signal to be processed and reproduced by a computer. According to the Nyquist sampling theorem, when a signal is sampled, it is necessary to ensure that the sampling rate should twice greater than the highest bandwidth of the original analogue signal. In speech processing, we usually use 16k Hz microphone sampling.
- Pre-processing

  Due to the uncertainty of the speech signal environment, it is necessary to pre-process the signal before extracting features in order to extract relatively clean speech signals. For example, commonly used pre-processing methods include VAD and speech enhancement.
- Pre-emphasis

The high-frequency signal in speech signals contains more information. During the transmission process of the signal, some information will be lost when the transmission rate is large. Therefore, it is necessary to compensate for the damaged signal to ensure that a good signal waveform is received. In the audio frequency spectrum, boosting the energy of high frequencies can make this high-frequency information more utilized by the acoustic model. So pre-emphasis first samples the speech signal and then enhances the high-frequency audio signal by using a high-pass filter. Namely, the pre-emphasis technology mainly enhances the high-frequency part of the audio signal at the beginning of the transmission line to reduce the loss of high-frequency information, which can increase the energy of the high-frequency part and compensate for the high-frequency part of the input signal. The following formula represents a common high-pass filter for pre-emphasis.

$$H(z) = 1 - \mu z^{-1}$$

in which, z is the input signal of the frequency domain, μ ranges from 0.9 to 1, usually values as 0.97. Moreover, to avoid the distortion of the speech data after the pre-emphasis processing step, de-emphasis is performed sometimes.

- Framing and Windowing

  In order to facilitate speech analysis, a given speech sample file is divided according to a fixed duration, and each fixed duration sample after division is called a frame. The length of each frame is determined by different task requirements, and the time length is usually from 20ms to 30ms. In addition, in order to decrease the changes and differences between adjacent frames, the artificially designed part of the overlap between adjacent frames is about 1/2 or 1/3 of the frame length. When dividing the frame, it is necessary to artificially add windowing and truncation to the speech sequence. According to the Gibbs phenomenon, if a rectangular window is used to directly truncate the signal during framing, the signal will be distorted in the frequency domain. To reduce distortion, it is necessary to use the window function to increase the continuity of the signal in the time domain. The commonly used window function is the Hamming window, and its definition is shown in the following formula.

$$\omega(n) = 0.5\left[1 - cos\frac{2\pi n}{N-1}\right] = sin^2\frac{\pi n}{N-1}$$

- Discrete Fourier Transform (DFT)

Fourier transform is performed on the above framed and windowed frame signals respectively and squared to obtain a power spectrum. The magnitude spectrum is:

$$S(k) = \sum_{n=1}^{N} s(n) e^{-\frac{j2\prod kn}{N}} \quad 0 \le k \le N-1$$

The power spectrum is:

$$P_i(k) = \frac{1}{N} |S(k)^2|$$

where N is the number of points of Fourier transform, and $s(n)$ is the time domain signal.

- Mel filter bank

  The FFT output represents the energy information in each frequency range. However, the frequency perception of humans is nonlinear and less sensitive to high frequencies. Correspondingly, the Mel scale simulates the sensitivity ranging scale of human beings, which has a high resolution of low frequency and a low resolution of high frequency. The Mel scale and frequency conversion can be represented as the following formula:

$$Mel(f) = 2595 log_{10}(1 + \frac{f}{700})$$

  where f represents the frequency, and Mel(f) represents the MEL scale corresponding to f.

  The Mel filter bank refers to a set of filters covering from 0Hz to the Nyquist frequency distributed with Mel scales.

Similar to the frequency domain, the perception of speech energy from the perspective of the human ear is also not linear. The logarithmic nonlinear relationship can better describe the perception of the human ear. Therefore, taking the logarithm of the output of the Mel filter can obtain the FBank features. Mel Frequency Cepstral Coefficients feature (MFCC) is commonly used in the GMM-HMM speech recognition system. The extraction process of MFCC is similar to that of FBank, the difference is whether discrete cosine transform (DCT) transformation is performed.

In GMMs, each dimension is usually regarded as independent and is modelled using a diagonal covariance matrix. Since the Mel filter banks overlap in the frequency domain during the FBank

calculation in the previous step, the output energies are related to each other. Therefore, DCT is used for decorrelation to better satisfy the independence assumption of GMM. Moreover, since the first few MFCC coefficients represent most speech information, truncating the coefficients of the higher-order DCT components can contribute to a more robust system.

As for the calculation of the Mel frequency, the low-frequency part of the original frequency is sampled more, so as to correspond to more frequencies, and the high-frequency sound is sampled less, so as to correspond to fewer frequencies. The human ear can distinguish the low and high frequencies of the Mel frequency in the same way. Regarding the calculation of Mel FBank, we generally use LogFBank as the identification feature. For example, Figure 6 shows the audio waveform of the Chinese sentence ‘哎上节课的笔记你发给那个老师了吗’ from the TAL_CSASR corpus. Figure 7 illustrates the visualized spectrogram of the extracted spectral features of the sample audio through the Short Time Fourier Transform (STFT). Figure 8 represents the extracted LogFBank of the sample audio. It is clear that the feature dimension of the third figure is smaller than the second figure. The STFT is often used in the field of speech signal processing. In a longer time signal, it can be divided into small segments of the same length, each of which is a stationary signal. Then the Fourier transform is calculated on each small segment.

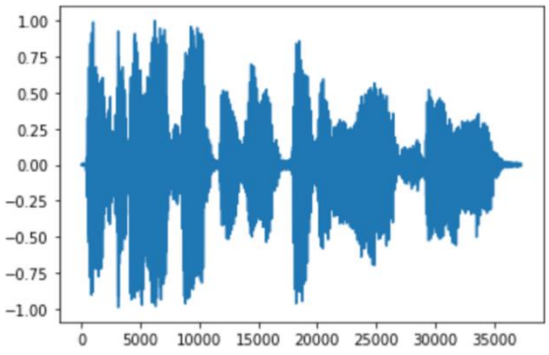

Figure 6. Audio waveform example.

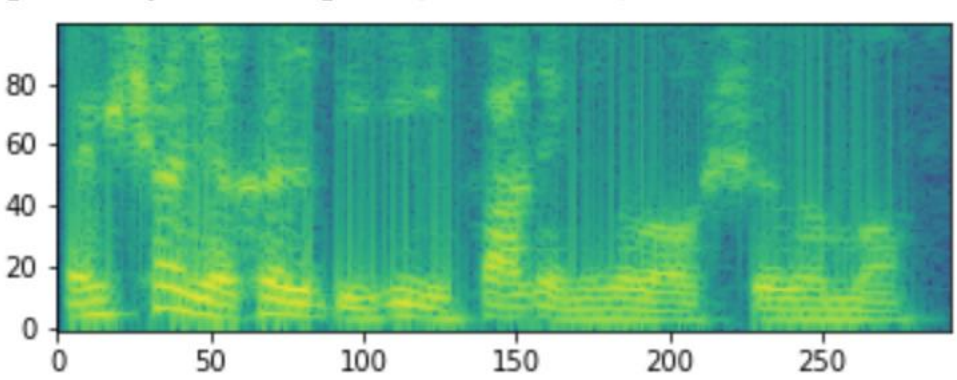

Figure 7. Visualized spectrogram through STFT.

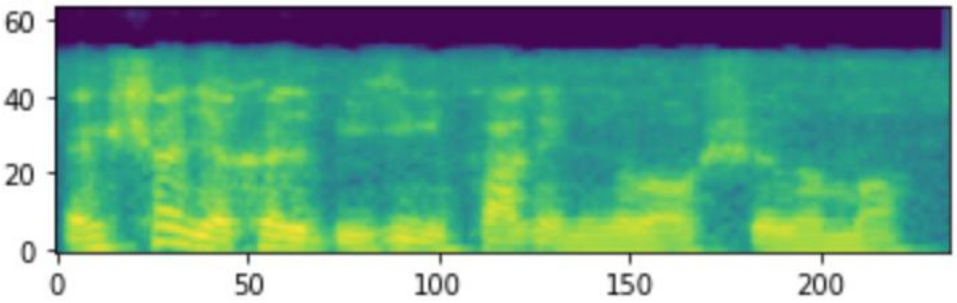

Figure 8. Extracted LogFBank of the sample audio.

4.1.5 Acoustic Model

The acoustic model converts characters or phonemes into acoustic features, such as linear spectrogram, Mel spectrogram, and LPC feature. Acoustic features are in units of frames. Usually, one frame is about 10ms, and one phoneme generally corresponds to about 5~20 frames. The acoustic model aims to solve the mapping problem between sequences with unequal lengths, which is a one-to-many scenario. For example, the mapping issue of unequal length may refer to the different duration of different utterances by the same speaker, the different rates of the same sentence at different times by the same speaker, the different characteristics of different speakers. In the speech recognition framework based on Hidden Markov Model (HMM), each phoneme corresponds to an HMM. In the GMM-HMM framework, the probability is generated by a Gaussian distribution or a mixture of Gaussian distributions. In the DNN-HMM framework, the probability is generated by the output of the neural network. On the basis of Bayes' theorem, the

state sequence of HMM $S = s \in \{1, \dots, J\} | t = \{1, \dots, T\}$ can be introduced into the formula to find the most probable W from all the text sequences that can produce the feature vector X:

$$W = \underset{W}{argmax} \sum_{S} P(X|S,W)P(S|W)P(W) \approx \underset{W}{argmax} \sum_{S} P(X|S)P(S|W)P(W)$$

in which, $P(X|S)$, P(S|W), and $P(W)$ represent the acoustic model, the pronunciation lexicon model and the language model. Based on the first-order Markov assumption and the observation independence assumption in the hidden Markov model, the acoustic model P(X|S) is expressed as the following equation.

$$P(X|S) = \prod_{t=1}^{T} P(x_t|x_1, \dots, x_{t-1}, S) \approx \prod_{t=1}^{T} P(x_t|s_t)$$

In speech recognition based on the GMM-HMM model, the acoustic model $P(X|S)$ is modelled by the GMM-HMM model. The likelihood $P(x|s)$ namely the observed probability of the HMM is modelled by a Gaussian Mixture Model (GMM). In the speech recognition based on the DNN-HMM model, it requires the likelihood $P(x \mid s)$ as the probability, which is modelled by a DNN in order to take advantage of the powerful classification capabilities of the DNN, as well as to take into account the contextual information of neighbouring frames. When DNN models the posterior probability $P(s|x)$ with the audio feature sequence as the input, the posterior probabilities need to be converted to likelihoods as follows.

$$P(x_t|s_t) = P(s_t|x_t)\frac{P(x_t)}{P(s_t)}$$

$P(S_t) = \frac{T_s}{T}$ is the prior probability of each state in the training set, which can be obtained from the alignment of the GMM-HMM model.

The pronunciation dictionary model $P(S|W)$ also adopts the probability chain rule and the first-order Markov assumption, which is decomposed as follows:

$$P(S|W) = \prod_{t=1}^{T} P(s_t|s_1, \dots, s_{t-1}, W) \approx \prod_{t=1}^{T} P(s_t|s_{t-1}, W)$$

where $P(s_t|s_{t-1})$ is modelled by a given HMM state transition probability. The HMM state is modelled in units of phonemes, so a character-to-phoneme mapping can be constructed from a pronunciation dictionary.

4.1.6 Vocoder

The role of the vocoder is to recover the speech waveform on the basis of the given acoustic features. Vocoders can be divided into vocoders based on traditional signal processing methods and data-driven vocoders based on neural network methods. The rule-based traditional vocoder utilizes domain knowledge of signal analysis to decompose speech into source and filter signals, which is used to match acoustic features. The target audio waveform signal can be recovered from the features analyzed by the corresponding vocoder. As for the data-driven neural network vocoder, it can build acoustic models from various acoustic features to speech achieving a higher quality of synthesized speech than traditional methods.

4.2 Tacotron Model

As a content-based attention end-to-end speech synthesis model, Tacotron has an encoder to process the input of raw text, a decoder to generate Mel-spectrogram frame, and a post-processing net to obtain linear-scale spectrogram, and Griffin-Lim as a vocoder to synthesize target waveform. The Tacotron model is introduced from the perspectives of the encoder, decoder, post-processing networks and waveform synthesis.

4.2.1 Encoder of Tacotron

Encoder aims to extract the robust sequential representation of the input text with each character represented as a one-hot vector embedded into a continuous vector (Wang et al., 2017). As shown in Figure 9, the encoder of Tacotron uses characters as input. Accordingly, the encoder of Tacotron can be regarded as the frontend analysis module of traditional speech synthesis, which needs to analyze the pronunciation features of the characters. Therefore, Tacotron's encoder is designed as a structure called CBHG, which refers to a 1 D convolutional filter bank, a highway network and a bidirectional gated recurrent neural network unit as shown in Figure 10.

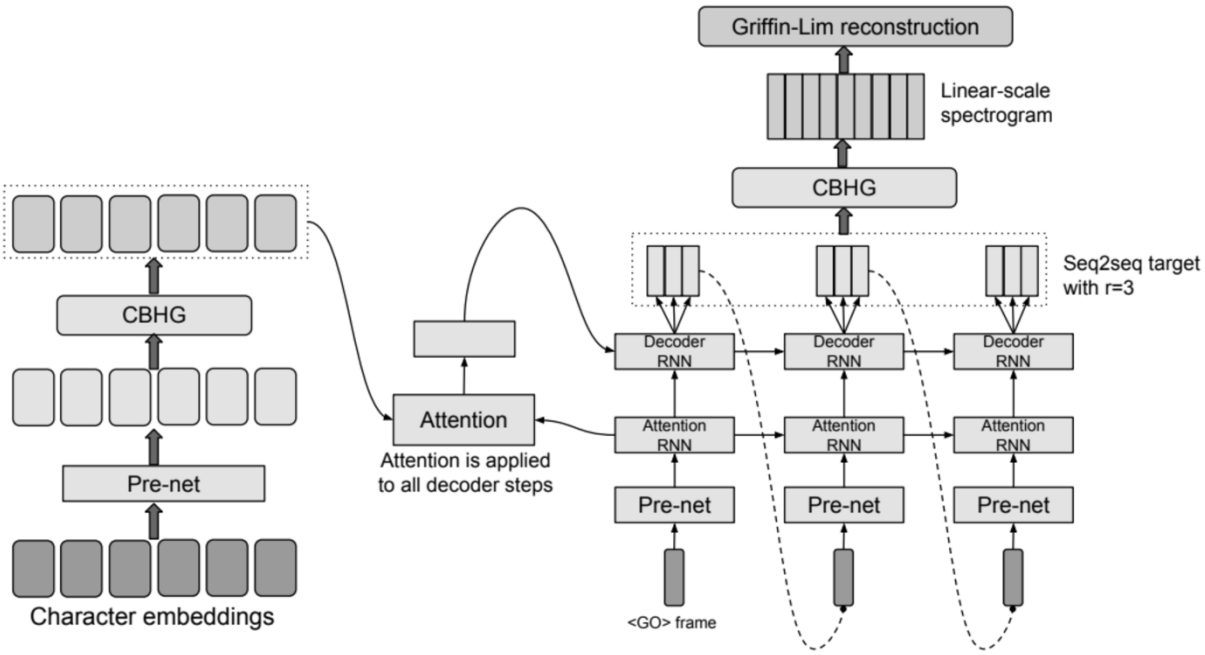

Figure 9. Tacotron model architecture adapted (Wang et al., 2017).

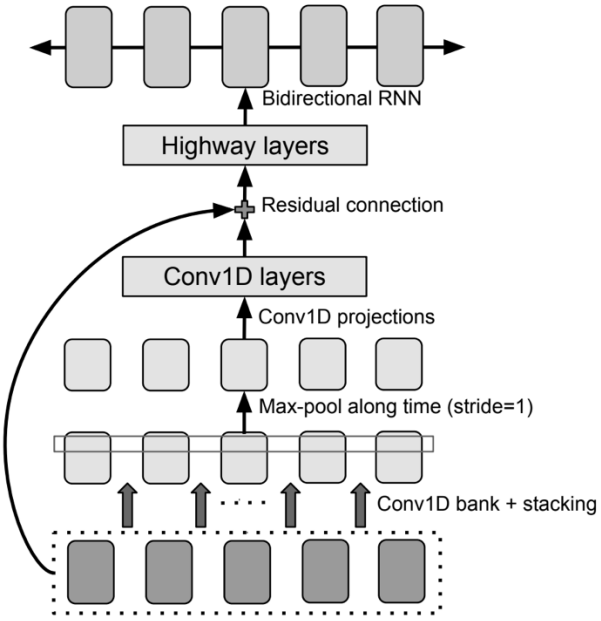

Figure10. CBHG module (Wang et al., 2017).

The convolution filter bank structure uses groups of 1D filters, which extract the contextual information of the input character. After that, the convolutional output is processed through a max-pooling layer to increase the local invariance of the model. A convolutional layer is then used to summarize and further process the information from the filter bank. The output of the convolutional layer is processed through a multi-layer highway network. The final output layer of the encoder uses a bidirectional gated recurrent neural network unit to further encode and process the contextual information of the input sequence. The bidirectional gated recurrent neural network unit as shown in Figure 11 is similar to LSTM, both of which aim to solve issues of long-term memory and gradients in backpropagation. Different from LSTM, gated recurrent unit use only one gate z_t to process forget and select memory.

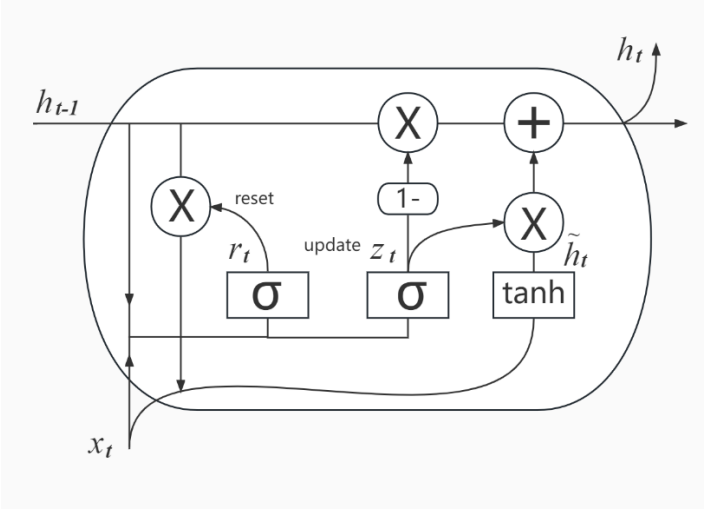

Figure11. Gated recurrent unit.

x_t refers to the input information at the current moment. h_{t-1} refers to the hidden state at the last moment, which acts as the neural network memory containing information about the data that nodes have seen before. h_t refers to the hidden state passed to the next moment. $\tilde{h}_t$ refers to the candidate hidden state. r_t refers to the reset gate. z_t refers to the update gate. σ refers to the sigmoid function, through which the data can be changed to a value in the range of 0-1. tanh refers to tanh function, through which the data can be changed to a value in the range [-1,1].

Moreover, before the input is processed by CBHG, it needs to be pre-processed by pre-net. As a bottleneck layer, pre-net is a module composed of a two-layer fully connected neural network, using ReLU as activation function and number 0.5 as dropout rate to increase random noise.

4.2.2 Decoder of Tacotron

The decoder is responsible for decoding the text embedding into speech frames using the Mel-scale spectrogram as the prediction output. Considering one character commonly corresponds to multiple audio frames, and adjacent audio frames usually also have a strong correlation, Tacotron decoder uses a content-based attention module to align text sequences and speech frames.

However, the autoregressive mechanism on the decoder of the sequence-to-sequence model may cause the posterior collapse issue, which refers to ignore the input of encoder, and only use the previous acoustic features to predict the current acoustic frame. The posterior collapse issue leads to the failure of the loss converge of the attention mechanism. In order to avoid the collapse issue, the pre-net module is utilized to added randomness and noise with the dropout setting, which makes the model concentrate more on the encoder information to predict the output acoustic frame. After the pre-net processing, the content-based attention mechanism selects the input encoder vector and obtains the frame-by-frame result as the input of the decoder. Furthermore, a three-layer unidirectional sequence-to-sequence gated recurrent unit is utilized to predict non-overlapping multi-frame outputs at the same time, which can speed up the loss convergence of the attention mechanism.

4.2.3 Post-processing Net and Waveform Synthesis

A post-processing network such as CBHG aims to convert the Mel-scale spectrogram output by the sequence-to-sequence model into a linear-scale spectrogram. Finally, a vocoder such as the Griffin-Lim algorithm generates the waveform of the target audio saved as a wav format sound file. The conversion step of the linear-scale spectrogram from the Mel-scale spectrogram is not necessary since the Griffin-Lim mechanism needs a linear-scale spectrogram as its input. Namely, the whole CBHG post-processing network and Griffin-Lim can be replaced by other modules to generate waveform directly such as the Wavenet of Tacotron 2.

4.3 Gradient Descent Optimization

This section focuses on the method selection of gradient descent optimization from the perspectives of updating direction based on both movement direction and magnitude of previous gradients, updating pace range based on the magnitude of previous gradients and updating speed based on the learning rate schedule.

Firstly, speech-text paired data (x^P, y^P) is sampled from D^P as $(x^P, y^P) = ([x_1^P, .., x_{S_P}^P], [y_1^P, .., y_{T_P}^P])$. The speech length of the sampled data is S_P, and the text length of the sampled data is T_P. The TTS models are supervised trained with labelled (x^P, y^P) text-speech data pairs. TTS generates the best predicted speech $\hat{x}^P$ from the input y^P using teacher-forcing.

$$\hat{x}_s^P = \underset{z}{argmax} P_{TTS}(z|x_{<s}^P, y^P;\ \theta_{TTS}); \forall s \in [1..S_P]$$

The output $\hat{x}^P$ based on the input y^P is calculated with the ground truth samples x^P to get the loss L_{TTS}^P.

$$L_{TTS}^P = LossTTS(x^P, \hat{x}^P; \theta_{TTS})$$

The TTS parameters θ_{TTS} are self-updated until convergence by gradient descent optimization with the derivative of L to minimize the loss L_{TTS}^P.

$$\theta_{TTS} \leftarrow Optim(\theta_{TTS}, \nabla_{\theta_{TTS}} L)$$

The traditional mechanism of vanilla gradient descent moves to the opposite direction of the derivative result to update the parameters θ^t. It consists of a fixed-step range for updating with the current gradient direction. For example, with the initialized parameter starting at θ^0, the gradient can be computed as g^0. Accordingly, the parameter can be updated by moving to $\theta^1 = \theta^0 - \eta g^0$ through the opposite direction of the derivative result with a minus sign, in which η refers to the learning rate. The Next gradient can be computed as g^1. After that, the parameter can be further updated by moving to $\theta^2 = \theta^1 - \eta g^1$.

$$g^t = \frac{\partial L}{\partial \theta}|_{\theta=\theta^t}$$

$$\theta^{t+1} = \theta^t - \eta g^t$$

On the basis of the traditional gradient descent, momentum can be added to consider the influence of the movement direction of the last step, which is the weighted sum of all the previous gradients $(g^0, g^1, \dots, g^{i-1})$. Therefore, the utilization of momentum enables the movement of gradient descent to escape the critical points with zero gradients in the conditions of saddle points or local minima. Namely, the movement of the next update depends on the movement influence of all the previous gradients. For example, with the initialized parameter starting at θ^0 and the initialized movement $m^0 = 0$, the gradient can be computed as g^0. Accordingly, both m and θ are updated. The weighted sum of all the previous gradient can be updated as $m^1 = \lambda m^0 - \eta g^0 = -\eta g^0$. The parameter is updated moving to $\theta^1 = \theta^0 + m^1$. After that, the Next gradient can be computed as g^1. Similarly, both m^1 and θ^1 are updated as $m^2 = \lambda m^1 - \eta g^1 = -\lambda\eta g^0 - \eta g^1$, and $\theta^2 = \theta^1 + m^2$.

$$m^{t+1} = \lambda m^t - \eta g^t$$

$$\theta^{t+1} = \theta^t + m^{t+1}$$

Besides the previous movement direction of gradient descent g, learning rate η is a considerable parameter during the training process. When the error surface of loss is rugged, the adaptive learning rate is a useful tip for training, which refers to different parameters equipped with different learning rates such as Adagrad. It enables the learning rate to become bigger when the gradient is steep, and become smaller when the gradient is smooth. The learning rate η is divided by the root mean square σ of the previous gradient as $\frac{\eta}{\sigma}$. Accordingly, different parameters θ_i are equipped with different learning rate $\frac{\eta}{\sigma_i}$.

$$\sigma_i^t = \sqrt{\frac{1}{t+1}\sum_{i=0}^{t}(g_i^t)^2}$$

$$\sigma_i^{t+1} = \sigma_i^t - \frac{\eta}{\sigma_i^t} g_i^t$$

However, Adagrad regards each previous gradient as equally important, which may suffer convergence issues when the error surface of loss dramatically changes between steep and smooth even in the same gradient descent direction. Therefore, the utilization of RMSProp can dynamically adjust the weight and importance of different gradient g by adjusting σ_i^t through hyperparameter α. For example, smaller α refers to the recent gradient having a larger influence, and the past gradients having less influence.

$$\sigma_i^t = \sqrt{\alpha(\sigma_i^{t-1})^2 + (1-\alpha)(g_i^t)^2}$$

Therefore, the combination of RMSProp and momentum namely Adam performs efficiently as gradient descent optimization. Adam considers both the movement direction and value of gradient descent based on previous gradients. As time goes by, although the gradient g becomes smaller, the previous σ are accumulated, and the updating pace may become large, which can lead to gradient explosion. Accordingly, learning rate scheduling plays an important role to solve the gradient explosion issue by learning rate decay or warm-up. Because as time progresses, the gradient gets closer and closer to the target convergence, reducing the learning rate η as time-dependent η^t will efficiently slow down the parameter update speed to avoid missing target convergence. Therefore, we apply Adam and learning rate decay as our gradient descent optimization.

$$\sigma_i^{t+1} = \sigma_i^t - \frac{\eta^t}{\sigma_i^t} g_i^t$$

Moreover, we apply warm-up learning rate in our training stage to accelerate the gradient descent process, which will be discussed in detail in the experimental setup section 5.3.

4.4 Evaluation Method

In the field of speech synthesis, we need to evaluate the quality of the synthesized speech. The evaluation method can be divided into the subjective method evaluated by human beings and the objective method evaluated by machines. The subjective evaluation methods include Mean Opinion Score (MOS), Diagnostic Rhyme Test (DRT), Degradation MOS (DMOS) and

Diagnostic Acceptability Measure (DAM). The objective method contains Perceptual Evaluation of Speech Quality (PESQ) and Perceptual Objective Listening Quality Assessment (POLQA).
MOS is the most commonly used subjective evaluation method to evaluate the quality of the synthesized speech, which divides the quality into five level scales ranging from 1 to 5. As shown in Table 4, number 1 refers to bad quality which cannot be heard clearly, while number 5 refers to excellent quality equipped with understandable clear speech.

Table 4. Mean Opinion Score (MOS) classification.

MOS	Quality
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

The MOS is the arithmetic mean of the subjective quality evaluation test results of the human assessors. The assessors give the subjective quality evaluation result based on what they hear. MOS can be calculated as the following equation, in which N refers to the number of human accessors, and R_n refers to the subjective quality evaluation test result of the n^{th} human assessor.

$$MOS = \frac{\sum_{n=1}^{N} R_n}{N}$$

The results of the MOS are very subjective, and the influence of the assessors themselves is large. For the same speech data, if the assessor is changed, the result of MOS may be different. Besides the subjective character of the MOS result, the time-consuming challenge and expensive cost of hiring human assessors are also considerable issues. Accordingly, we apply machine-based objective evaluation method PESQ to evaluate our synthesized speech. PESQ is widely used to assess speech quality in the field of end-to-end speech measurements, narrow-band speech codecs, and mobile networks. PESQ can give predictions of subjective quality evaluation such as MOS in various conditions such as coding distortions, channel errors, background noise, analogous

filtering, and variable delay scenarios. Those effects can be addressed through time alignment, transfer equalisation, and time-averaged distortion algorithms (Gardlo, 2009).

Figure 12 shows the structure of the PESQ algorithm. The PESQ algorithm can predict the test-speech quality that would be assessed by human assessors subjectively in a listening test. The equalized time-aligned original reference speech and test speech are processed through a perceptual model including auditory transforms of calibrated listening level alignment, mapping of time frequency, warping of frequency, and scaling of compressive loudness. These auditory transforms involve the perceptual frequency bark, the loudness sone, as well as the equalization for linear filters and gained variations. Furthermore, the extracted internal representation features of reference speech and test speech are compared and calculated through a cognitive model, in which distortion parameters exist due to the difference between the original reference speech and the test speech. The extracted distortion parameters are aggregated in time and frequency and further mapped to predict the MOS result to evaluate the quality of test speech.

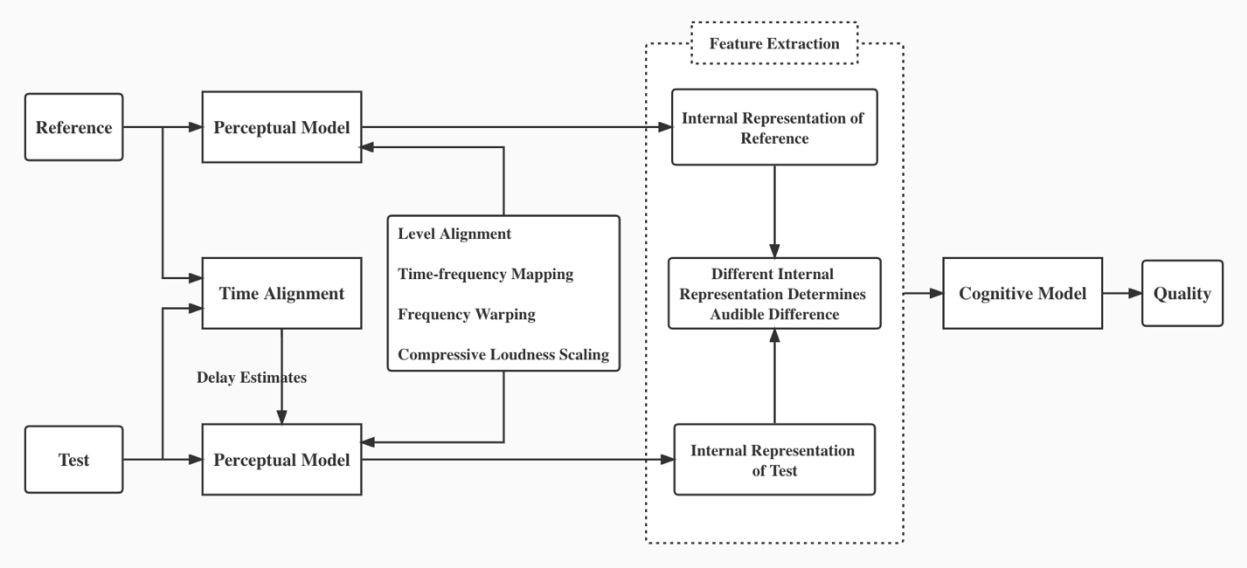

Figure 12. Structure of the PESQ algorithm.

The standard PESQ tool that we used is based on ITU-T Recommendation P.862[4]. The sampling rate of the input speech is 16 kHz as shown in Figure 13.

[4] https://www.itu.int/rec/T-REC-P.862

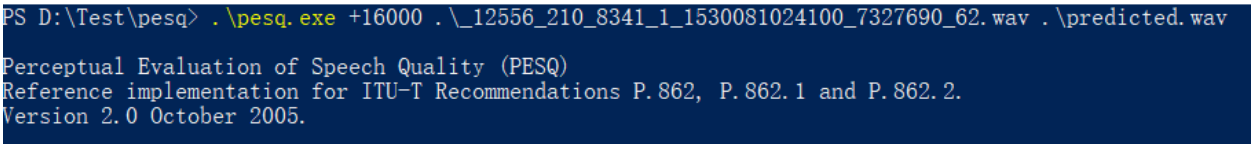

Figure 13. Example of the PESQ tool.

The PESQ tool can calculate the Mean Opinion Score – Listening Quality Objective (MOS-LQO) value of the audio speech by calculating the difference value between the extracted test signal and the original reference signal. The PESQ raw MOS score ranges from 0.5 to 4.5 points. In order to gain a score equivalent to MOS, the original objective PESQ raw MOS score needs to be mapped to the MOS-LQO score through a linear monotonic regression process. This regression not only process preserves the information of the PESQ raw MOS score, but also removes the systematic offset between the objective PESQ score and the subjective MOS score and minimizes the mean square of the residual errors.

The obtained MOS-LQO score applies the same score scale as MOS ranging from 1 to 5, in which a low calculated value refers to the bad quality and bad performance equipped with a large discrepancy between the output audio speech and the original reference audio speech, while a high calculated value refers to the good quality and good performance equipped with a small discrepancy between the output audio speech and the original reference audio speech.

5. Experiment

This chapter is divided into four sections to introduce the core information of our experiment. After introducing the experiment environment in the first section 5.1, we introduce the pre-processing of speech and text frontend in section 5.2. Experimental setup is introduced in section 5.3. Results and discussion are given in section 5.4.

5.1 Environment

We train the model under the default environment of Google Colab Pro[5] equipped with Tesla-P100. The CUDA version is 11.2. As for the installation of the dependencies, we install torch with version 0.4.1, librosa with version 0.7.2, numpy with version 1.15.4, numba with version 0.48.0, matplotlib with version 2.2.3, scipy with version 1.2.0, tqdm with version 4.29.0, unidecode, pydub, nnmnkwii, tensorboardX, and pypinyin.

5.2 Pre-processing

This part introduces the pre-processing of speech and text frontend before the training process. We use 28.79 hours of audio data from the TAL_CSASR corpus as our training dataset, which consists of 20,000 labelled speech-text sentence pairs containing 8,292,361 frames. The original audio speech from the TAL_CSASR corpus sampled at 16 kHz and 16 bit. The audio file is formatted as wav, and the transcript label file is formatted as txt. In each line of the transcript label file, there is a unique id number equipped with a Chinese-English code-switching sentence containing both Chinese characters and English words in capitalized format. Correspondingly, each ID number is equipped with an acoustic audio waveform file, which combines the audio sentence together with the transcript label text.

5.2.1 Speech pre-processing

The fundamental speech frequency of males ranges from 85 Hz to 155 Hz, which is lower than female frequency ranging from 165 Hz to 255 Hz (Baken & Orlikoff, 2000). Since the

[5] https://research.google.com

TAL_CSASR corpus contains speech data from male teachers, the boundary between the minimum frequency of male speech and low-frequency noise is considered. Accordingly, to remove the low frequency noise, a high-pass filter is used to cut off the low frequencies under 50 Hz. Moreover, we smooth the speech frequencies ranging from 50 Hz to 200 Hz. To achieve volume consistency, we also normalize the speech volume to 10 dB. Figure 14 shows the processed waveform and the corresponding original waveform selected from the TAL_CSASR corpus. It is obvious that the processed waveform significantly improves the quality of speech data from the perspectives of noise removing, frequency smoothness and volume consistency, which contributes to our training process and the following TTS generation process.

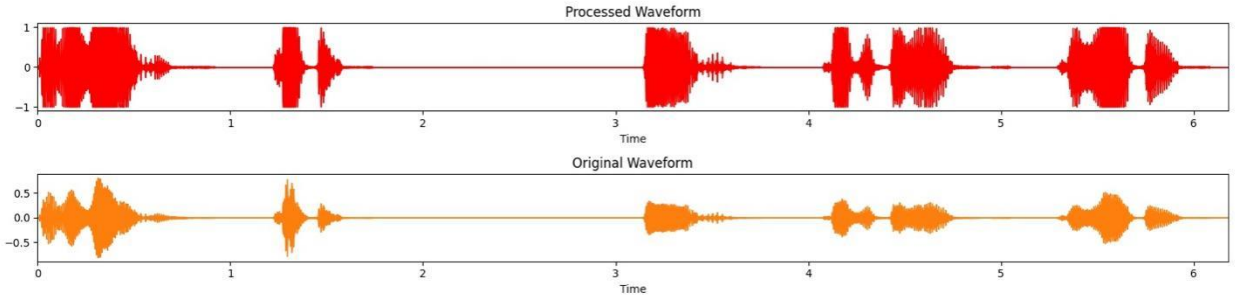

Figure 14. processed waveform and original waveform.

5.2.2 Text frontend pre-processing

The original transcript sentences contain a mix of Chinese characters and English words. We use the Python pypinyin toolkit [6] to convert the Chinese part of the original sentence to Chinese pinyin with tones. The pinyin of a single Chinese character uses data from pinyin-data[7]. The pinyin of the Chinese phrase uses the data from phrase-pinyin-data[8]. We apply TONE3 as the literal style when defining the function of transferring a character to pinyin as a one-to-one mapping in the pre-processing stage. Accordingly, various Chinese characters are transferred to sequences equipped with 26 English letters and 4 numbers referring to five different Chinese tones. Numbers 1, 2, 3, and 4 refer to Yinping, Yangping, Shangsheng, and Qusheng separately. Syllables without numbers refer to Qingsheng without tone. Each English word is separated with space, and all letters

[6] https://pypi.org/project/pypinyin/
[7] https://github.com/mozillazg/pinyin-data
[8] https://github.com/mozillazg/phrase-pinyin-data

of English words are capitalized, which is easy to distinguish English words from Chinese parts in code-switching sentences.

We also provide a process method to solve the challenge of dealing with mixed Chinese-English code-switching SSML input for test demands. The speak root element of the SSML of Aliyun TTS based on version 1.1 of W3C SSML cannot be executed resulting in language speaking failure when processing speech synthesis of mixed code-switching Chinese-English text, because the mixed Chinese and English text will be classified as monolingual English by the SSML element placed at the front end of the text for language judgment. As shown in the following Figure 15, we have solved this issue when processing speech synthesis of mixed code-switching Chinese-English text by adding the content of the selected element ‘say-as’ to the target tmpSegments array, merging the Chinese text part into the target ‘speak’ element tag to avoid the potential intra-sentence pause of the synthesized speech, and processing the Chinese text part separately based on the target ‘speak’ element tag.

```
def get_input_ids(self,
                  sentence: str,
                  merge_sentences: bool=False,
                  get_tone_ids: bool=False,
                  add_sp: bool=True,
                  to_tensor: bool=True) -> Dict[str, List[paddle.Tensor]]:
    ''' 1. add the content of the element say-as to the target tmpSegments array
    '''
    d_inputs = MixTextProcessor.get_dom_split(sentence)
    tmpSegments = []
    for instr in d_inputs:
        '''  say-as only'''
        if instr.lower().startswith("<say-as"):
            tmpSegments.append((instr, "zh"))
        else:
            tmpSegments.extend(self.get_segment(instr))
```

```
    ''' 2. merge Chinese part to avoid the potential intra-sentence pause of the synthesized speech
    '''
    segments = []
    currentSeg = ["", ""]
    for seg in tmpSegments:
        if seg[1] == "en" or seg[1] == "other":
            if currentSeg[0] == '':
                segments.append(seg)
            else:
                currentSeg[0] = "<speak>" + currentSeg[0] + "</speak>"
                segments.append(tuple(currentSeg))
                segments.append(seg)
                currentSeg = ["", ""]
        else:
            if currentSeg[0] == '':
                currentSeg[0] = seg[0]
                currentSeg[1] = seg[1]
            else:
                currentSeg[0] = currentSeg[0] + seg[0]
    if currentSeg[0] != '':
        currentSeg[0] = "<speak>" + currentSeg[0] + "</speak>"
        segments.append(tuple(currentSeg))
```

```
            ''' 3. separately process the Chinese part with <speak></speak>
            '''
            if content.strip() != "" and \
                re.match(r".*?<speak>.*?</speak>.*", content, re.DOTALL):
                input_ids = self.zh_frontend.get_input_ids_ssml(
                    content,
                    merge_sentences=False,
                    get_tone_ids=get_tone_ids,
                    to_tensor=to_tensor)
            else:
                input_ids = self.zh_frontend.get_input_ids(
                    content,
                    merge_sentences=False,
                    get_tone_ids=get_tone_ids,
                    to_tensor=to_tensor)
```

Figure 15. Process mixed Chinese-English code-switching SSML input.

We also import large pinyin from pypinyin_dict.phrase_pinyin_data to optimize the polyphone problem. Besides setting the tone sandhi as true, we extend the rule and list of tone sandhi. The frontend solution is based on g2pm and python-pinyin as our default g2p tools which are merged into our mixed Chinese-English code-switching fronted processing. We have further extended the rules of tone sandhi of Chinese character '不' and '一', and added some new three sandhi rules for different third tone changing scenarios.

For example, as shown in Figure 16, the tone of '不' should be 2 before tone4, such as '不要' with the tone of 'bu2 要 4'. The tone of '一' should be 2 before tone4, such as '一共' with the tone of 'yi2 gong4'.

```
def _bu_sandhi(self, word: str, finals: List[str]) -> List[str]:
    # such as '看不懂'
    if len(word) == 3 and word[1] == "不":
        finals[1] = finals[1][:-1] + "5"
    else:
        for i, char in enumerate(word):
            # the tone of '不' should be 2 before tone4, such as '不要' with the tone of 'bu2 yao4'
            if char == "不" and i + 1 < len(word) and finals[i +
                                                            1][-1] == "4":
                finals[i] = finals[i][:-1] + "2"
    return finals

def _yi_sandhi(self, word: str, finals: List[str]) -> List[str]:
    #  '一' in number sequences, such as '一二三四', '二零一三'
    if word.find("一") != -1 and all(
        [item.isnumeric() for item in word if item != "一"]):
        return finals
    else:
        for i, char in enumerate(word):
            if char == "一" and i + 1 < len(word):
                # '一' before tone4 should be yi2, such as '一共' with the tone of 'yi2 gong4'
                if finals[i + 1][-1] in {'4', '5'}:
                    finals[i] = finals[i][:-1] + "2"
```

Figure 16. Tone sandhi examples of '不' and '一'.

As for the three-sandhi scenario, we apply two processing methods. The first method is segmentation-rule-based method as shown in Figure 17. The different combination order of the disyllabic and monosyllabic may lead to different tones of the first character. For example, although all the characters of '展览馆' and '李老板' should be the third tone separately, the first word is disyllabic plus monosyllabic, which leads to the third tone sandhi of the first character '展'. By contrast, the second word is monosyllabic plus disyllabic, which leads to the third tone sandhi of the second character '老'. The second method is the self-definition-based method as shown in Figure 18, which can extend more possibilities freely.

```
def _three_sandhi(self, word: str, finals: List[str]) -> List[str]:

    if len(word) == 2 and self._all_tone_three(finals):
        finals[0] = finals[0][:-1] + "2"
    elif len(word) == 3:
        word_list = self._split_word(word)
        if self._all_tone_three(finals):
            #  disyllabic + monosyllabic, such as '展览/馆'
            if len(word_list[0]) == 2:
                finals[0] = finals[0][:-1] + "2"
                finals[1] = finals[1][:-1] + "2"
            #  monosyllabic + disyllabic, such as '李/老板'
            elif len(word_list[0]) == 1:
                finals[1] = finals[1][:-1] + "2"
```

Figure 17. Segmentation-rule-based three-sandhi examples.

```
        self.mix_ssml_processor = MixTextProcessor()
        self.tone_modifier = ToneSandhi()
        self.text_normalizer = TextNormalizer()
        self.punc = ": ，；。？！“”‘’':,;.?!"
        self.rhy_phns = ['sp1', 'sp2', 'sp3', 'sp4']
        self.phrases_dict = {
            '李老板': [['li3'], ['lao2'], ['ban3']],
            '岂有此理': [['qi2'], ['you3'],['ci2'], ['li3']],
            '展览馆': [['zhan2'], ['lan2'], ['guan3']],
            '只有': [['zhi2'], ['you3']],
            '五种': [['wu2'], ['zhong3']],
            '产品': [['chan2'], ['pin3']]
        }
```

Figure18. Self-definition-based three-sandhi examples.

Based on the pre-processed speech and text, we can get the meta files mapping the speech wav file and text txt file as one-to-one pairs into meta npy file, which can be used in the following model training process. The metafile contains the information of 8,292,361 frames with a max input length of 738, and a max output length of 1922.

5.3 Experimental Setup

Our end-to-end speech synthesis model for Chinese-English code-switching scenario is based on Tacotron with some modification and improvement. We use the ReLU as our activation function and a dropout rate of 0.5 in the pre-net module to increase random noise, which can improve the generalization ability of the model and speed up the convergence speed. In the decoder part, after the pre-net processing, the content-based attention mechanism selects the input encoder vector and obtains the frame-by-frame result as the input of the decoder. We set the embedding dimension as 256, the Mel-scale spectrogram dimension as 80, and the Linear-scale spectrogram dimension as 1,025. Especially, we set the sequence-to-sequence target with r=6, which means we use a six-layer unidirectional sequence-to-sequence gated recurrent unit to predict non-overlapping multi-frame outputs at each decoder step at the same time, which can speed up the loss convergence of the attention mechanism. Furthermore, all these r frames are transferred to the next step as input rather than only the last frame. By doing this, the contextual correlation of adjacent frames is well considered to avoid the overweight bias of the last frame and possible posterior collapse. We train the model by Adam with an initial learning rate of 0.002. Moreover, we apply the warm-up technology in the learning rate setup. The warm-up learning rate can significantly speed up the loss convergence at the beginning stage of straining to find the critical point. We set the warmup step as 6,000. After the $6{,}000^{th}$ step, the learning rate is time decaying.

5.4 Results and Discussion

As shown in Figure 19, we have obtained the loss training curves and learning rate curves.

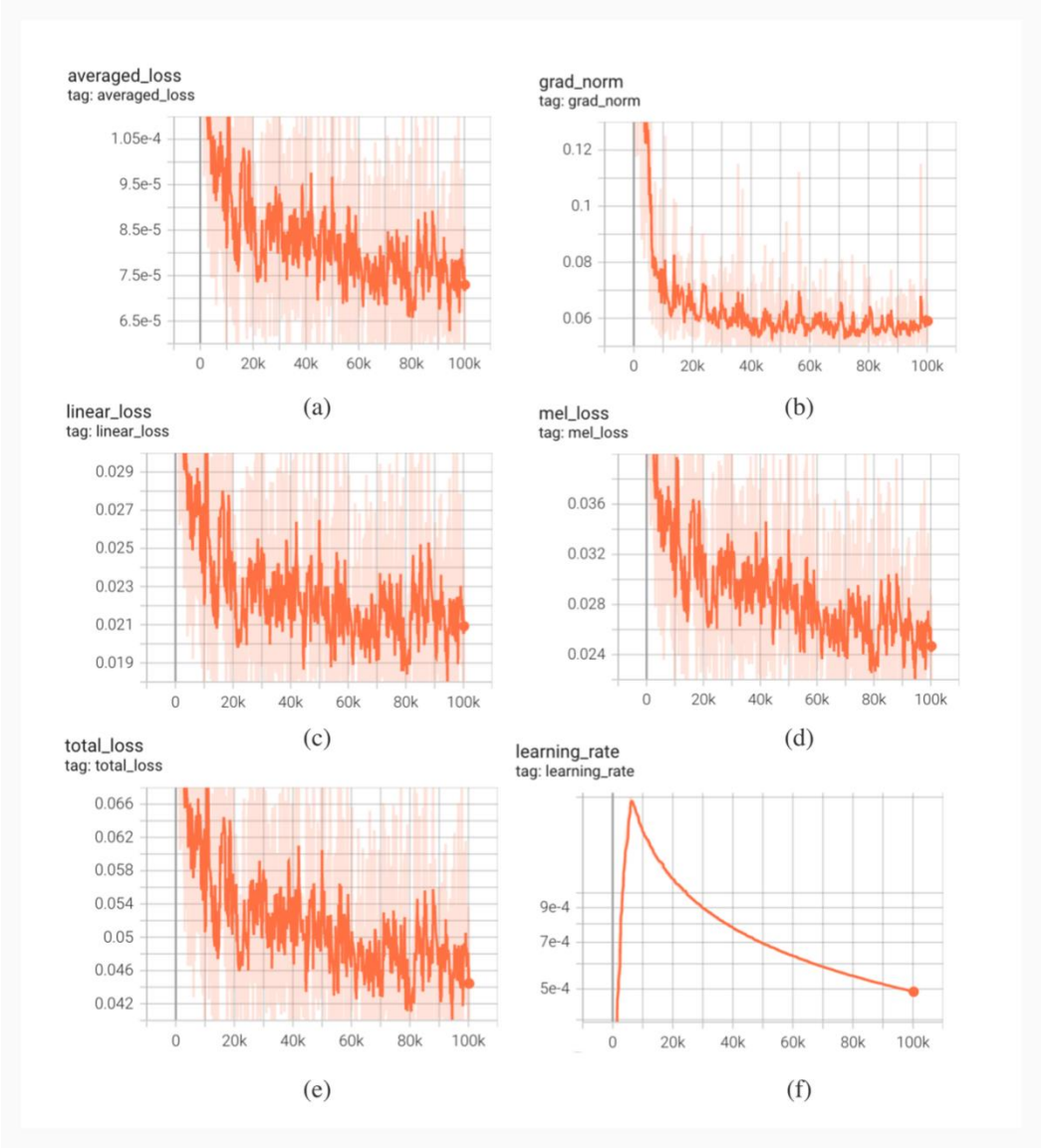

Figure 19. Loss training curves and learning rate curve.

The loss training curves include average loss (a), grad norm loss (b), linear loss (c), Mel loss (d), and total loss (e). The learning rate curve (f) clearly illustrates the warm-up setting with a boundary turn around the 6,000th step, which significantly deducts the grad norm loss in the early training

steps. The convergence trend of the Mel loss performs better than linear loss. Considering the post-processing network structure of our model, the Mel-scale spectrogram is converted to a linear-scale spectrogram. The conversion from the Mel-scale spectrogram to the linear-scale spectrogram causes worse convergence loss. After 336,000 steps of training, we obtain the following alignment as shown in Figure 20. The following Figure 21 and Figure 22 show the comparison between the predicted spectrogram speech and the corresponding target original spectrogram sampled from the TAL_CSASR corpus.

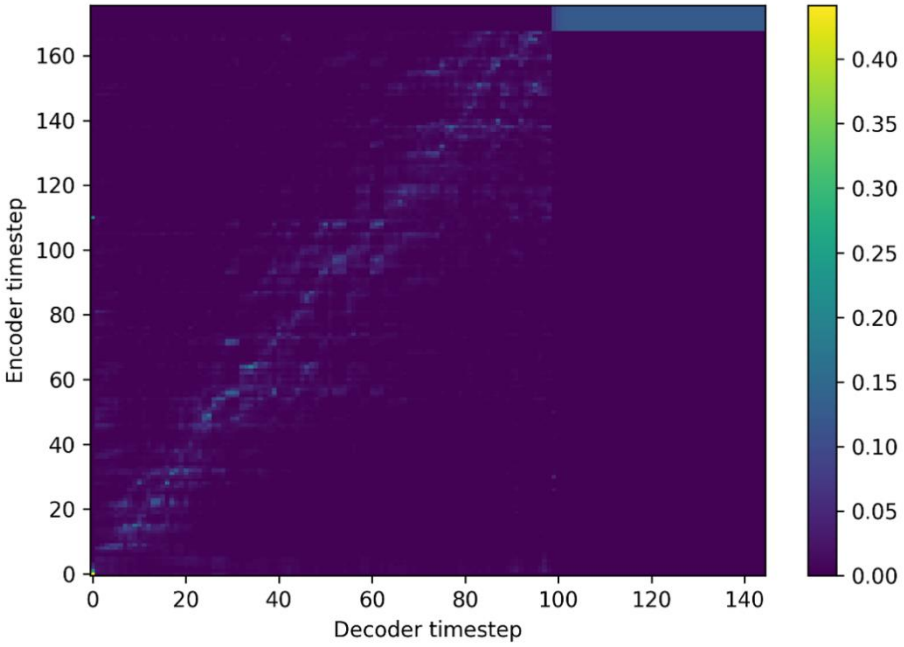

Figure 20. Alignment of training step 336,000.

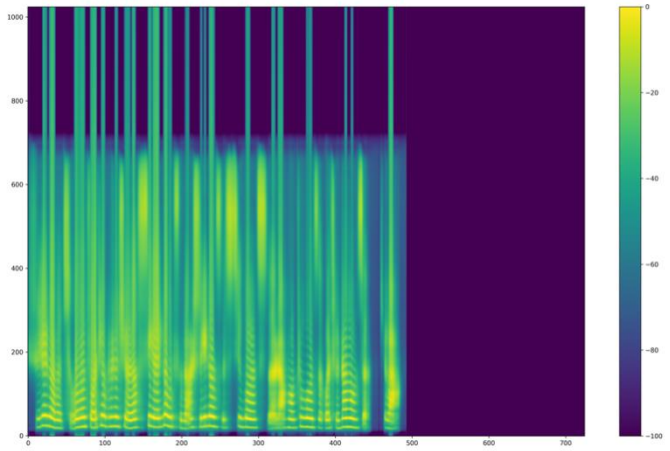

Figure 21. Predicted spectrogram.

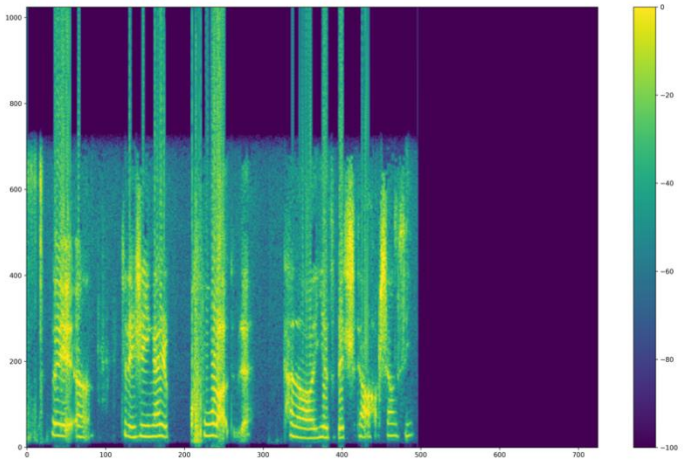

Figure 22. Target spectrogram.

We realised the synthesis of Chinese-English code-switching speech from bilingual code-switching text input. According to the experimental synthesis results, adding a pre-process module to the frontend can extract the text embeddings of different bilingual code-switching inputs and generate the spectrum to synthesize waveform.

However, from the comparison results of Figure 23 and Figure 24, although the harmonic profile of the speech is retained, the model loses some parts of the audio amplitude information, which leads to the gap between the tone of the synthesized audio and the target speech. Considering the training speech data is collected from multiple speakers including males and females, the tone gap may mean that the decoding potentially amortizes the error to minimize the loss. Accordingly, the recovery of the spectral amplitude information tends to average the amplitudes of multiple

speakers. In order to solve this issue, a separate design of the multi-speaker features module should be considered to improve the decoding network and improve the synthesis result.

We synthesized the predicted speech through our model using the text information as input, which is sampled from the TAL_CSASR corpus. The sampling rate of the synthesized speech and the original reference speech is 16 kHz. By the utilization of the PESQ tool, we obtain the PESQ raw MOS score and the MOS-LQO score. The MOS-LQO score is mapped from the PESQ raw MOS score through a linear monotonic regression process to be compared with the subjective MOS score.

As shown in Figure 22, we give an example of PESQ result. We obtained a predicted wave file based on the text information of _12556_210_8341_1_1530081024100_7327690_62.wav sampled from the TAL_CSASR corpus through our end-to-end speech synthesized model. After that we convert our synthesized speech to 16k audio clips. Afterwards, the PESQ processes the original reference speech and the predicted speech and gives PESQ raw MOS score as 3.163 and the MOS-LQO score as 3.065.

```
Reading reference file .\_12556_210_8341_1_1530081024100_7327690_62.wav...done.
Reading degraded file .\predicted.wav...done.
 Level normalization...
 IRS filtering...
 Variable delay compensation...
 Acoustic model processing...

P.862 Prediction (Raw MOS, MOS-LQO):  = 3.163   3.065
```

Figure 23. Example of PESQ result.

We randomly choose 50 pairs of original reference speech and predicted test speech, in which all the original reference speech are sampled from the TAL_CSASR corpus. We write the command code in a new txt file, and then change it to a bat file to run a batch processing as shown in Figure 23.

```
1  pesq +16000 test/_12556_210_8341_1_1530081024100_7327690_62.wav test/predicted_12556_210_8341_1_1530081024100_7327690_62.wav
2  pesq +16000 test/_40380_210_9059_1_1533380594759_4187380_368.wav test/predicted_40380_210_9059_1_1533380594759_4187380_368.wav
3  pesq +16000 test/_13687_210_7497_1_1530705699576_3168359_330.wav test/predicted_13687_210_7497_1_1530705699576_3168359_330.wav
4  pesq +16000 test/_13921_210_9994_1_1530841782272_3503520_81.wav test/predicted_13921_210_9994_1_1530841782272_3503520_81.wav
5  pesq +16000 test/_12328_210_9767_1_1530532499731_4060890_415.wav test/predicted_12328_210_9767_1_1530532499731_4060890_415.wav
6  pesq +16000 test/_43552_210_8913_1_1533726031059_3523429_80.wav test/predicted_43552_210_8913_1_1533726031059_3523429_80.wav
```

Figure24. Example of PESQ batch processing.

We divide 50 pairs of speech data equally into five groups and run batch processing through the PESQ tool. The result of our test single sample audio data reaches the highest score of 3.163 PESQ raw MOS and 3.065 MOS-LQO as shown in Figure 23. As shown in Table 5, we obtain the average score of PESQ raw MOS and MOS-LQO of each group, and the total average scores are also calculated. The result of our test data reaches the highest average score of 2.672 PESQ raw MOS, and 2.520 MOS-LQO.

Table 5. Average Score of PESQ raw MOS and MOS-LQO.

Test Sample Number	PESQ Raw MOS	MOS-LQO
01-10	2.783	2.531
11-20	2.481	2.386
21-30	2.511	2.363
31-40	2.946	2.782
21-50	2.638	2.537
Average Score	2.672	2.520

We record 10 monolingual Chinese speech-text sentences, 10 monolingual English speech-text sentences, and 10 mixed Chinese-English code-switching pairs to test the performance of our model in monolingual and code-switching scenarios. The sampling rate of the speech is 16 kHz. We use the text as input to generate synthesized speech which is evaluated with the original recorded speech by the PESQ tool. The code-switching text contains plain text and SSML format input. As shown in Table6, we obtain the PESQ raw MOS and MOS-LQO of monolingual and code-switching scenarios.

Table6. PESQ raw MOS and MOS-LQO of monolingual and code-switching scenarios.

Language	PESQ Raw MOS	MOS-LQO
Chinese	3.331	3.146
English	1.193	1.021
Code-switching (plain)	2.822	2.613
Code-switching (SSML)	2.822	2.613

The result of monolingual Chinese performs better than code-switching scenario, reaches the score of 3.331 of PESQ Raw MOS, and 3.146 of MOS-LQO. The results of code-switching plain input and SSML input are the same. The result of monolingual English performs worst.

6. Conclusion and Future Work

Speech processing is an interdisciplinary subject, involving a wide range of fields. The modelling and analysis of speech processing can be regarded as the research on the mechanism of human vocalization and hearing. The simulation and modelling of this process can not only help human beings better understand the physiological and neural control process of speech production, but also promote the integration and mutual learning of various interdisciplinary fields.

Chinese-English code-switching scenarios and the complexity and variation of Chinese linguistic special cases such as polyphone and tone sandhi present higher challenges compared to the traditional monolingual speech synthesis model. Our end-to-end Chinese-English code-switching TTS model can solve the code-switching issue, especially in intra-sentential conditions and Chinese linguistic polyphone and tone sandhi scenarios. We also solve the issue of language speaking failure and processing failure of the switched language of the current monolingual-support speech synthesis markup language, which is improved to be able to process mixed Chinese-English code-switching SSML input. We also further extend the rules of polyphone and tone sandhi of the Chinese part in code-switching sentences.

We pre-process the speech data from the perspectives of low-frequency noise removal, frequency smoothness and volume consistency by using a high-pass filter to smooth the speech frequencies ranging and normalizing the speech volume. Secondly, we pre-process the Chinese-English code-switching input text by using the Python pypinyin toolkit to convert the Chinese part of the original bilingual sentence to Chinese Pinyin with tones. Moreover, we use a segmentation rule-based monolingual Chinese G2P module to solve the issues of polyphone and tone sandhi. To improve the attention mechanism module of our model and avoid the possible posterior collapse issue, we transfer all intermediate frames to the next processing to keep the contextual correlation of adjacent frames, instead of only transferring the last frame, which will lose the context information. In addition, we use Adam and warm-up the learning rate at the first 6,000 steps to speed up the loss convergence of the attention mechanism during the model training. We also accelerate the training process by adding a six-layer unidirectional sequence-to-sequence gated recurrent unit to predict more non-overlapping multi-frame outputs at each decoder step. The result of our test data reaches

the highest score of 3.163 PESQ raw MOS and 3.065 MOS-LQO, and the average score of 2.672 PESQ raw MOS, and 2.520 MOS-LQO.

However, there are some limitations of our work. Firstly, the result of monolingual Chinese performs well may because the optimization of Chinese sandhi processing. The result of monolingual English performs worst may because the lack of pure English training data. Secondly, according to the specotrogram comparison, there are gaps between the tone of the synthesized audio and the tone of the target speech. Considering our corpus is collected from multiple speakers, a separate module of the multi-speaker id features needs to be designed to improve the decoding network and improve the synthesis result. Thirdly, the evaluation method PESQ that we used only measures the impacts of speech distortion and noise on speech quality, ignoring the impacts of loudness loss delay, sidetone and echo scenarios. However, the application of synthesized speech is oriented to human beings equipped with more comprehensive considerations of hearing and understanding than machines. Therefore, there is a considerable gap between the machine-based objective PESQ evaluation method and the human-based subjective MOS evaluation method. Only the machine based PESQ score is not enough to well evaluate the speech quality from a comprehensive perspective.

In addition, besides the traditional monodirectional TTS pipeline, the ASR-TTS dual learning speech chain is a good direction for future research. It combines both speech recognition and speech synthesis together to obtain better performance than the traditional separately TTS training process with limited labelled code-switching data. More exploration can be done in the future to combine ASR and TTS in Chinese-English code-switching scenarios to achieve better performance than traditional independent monolingual speech recognition or speech synthesis.

References

Ardila, A. (2005). Spanglish: an anglicized Spanish dialect. *Hispanic Journal of Behavioral Sciences, 27*(1), 60-81.

Arık, S. Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., Li, X., Miller, J., Ng, A., & Raiman, J. (2017). Deep voice: Real-time neural text-to-speech. International Conference on Machine Learning,

Baken, R. J., & Orlikoff, R. F. (2000). *Clinical measurement of speech and voice*. Cengage Learning.

Bendel, O. (2019). The synthetization of human voices. *Ai & Society, 34*(1), 83-89.

Cai, Z., Yang, Y., & Li, M. (2020). Cross-lingual Multispeaker Text-to-Speech under Limited-Data Scenario. *arXiv preprint arXiv:2005.10441*.

Cai, Z., Yang, Y., Zhang, C., Qin, X., & Li, M. (2019). Polyphone disambiguation for mandarin chinese using conditional neural network with multi-level embedding features. *arXiv preprint arXiv:1907.01749*.

[Record #26 is using a reference type undefined in this output style.]

Charpentier, F., & Moulines, E. (1989). Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. Eurospeech,

Denes, P. B., Denes, P., & Pinson, E. (1993). *The speech chain*. Macmillan.

Dutoit, T. (2001). *An introduction to text-to-speech synthesis* (Vol. 3). Springer Science & Business Media.

Ganji, S., Dhawan, K., & Sinha, R. (2019). IITG-HingCoS corpus: A Hinglish code-switching database for automatic speech recognition. *Speech Communication, 110*, 76-89.

Gardlo, B. (2009). *Subjective audiovisual quality in mobile environment* Master's thesis, Vienna University of Technology-Faculty of Electrical …].

Gold, B., Morgan, N., & Ellis, D. (2011). *Speech and audio signal processing: processing and perception of speech and music*. John Wiley & Sons.

Hemati, H., & Borth, D. (2020). Using IPA-Based Tacotron for Data Efficient Cross-Lingual Speaker Adaptation and Pronunciation Enhancement. *arXiv preprint arXiv:2011.06392*.

Hunt, A. J., & Black, A. W. (1996). Unit selection in a concatenative speech synthesis system using a large speech database. 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings,

Kakegawa, N., Hara, S., Abe, M., & Ijima, Y. (2021). Phonetic and Prosodic Information Estimation from Texts for Genuine Japanese End-to-End Text-to-Speech}}. *Proc. Interspeech 2021*, 126-130.

Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., Oord, A., Dieleman, S., & Kavukcuoglu, K. (2018). Efficient neural audio synthesis. International Conference on Machine Learning,

Kaur, R., & Sharma, D. (2016). An Improved System for Converting Text into Speech for Punjabi Language using eSpeak. *International Research Journal of Engineering and Technology (IRJET), 3*(04), 500-504.

Li, B., & Zen, H. (2016). Multi-language multi-speaker acoustic modeling for LSTM-RNN based statistical parametric speech synthesis.

Li, K., Li, J., Ye, G., Zhao, R., & Gong, Y. (2019). Towards code-switching ASR for end-to-end CTC models. ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Li, N., Liu, S., Liu, Y., Zhao, S., & Liu, M. (2019). Neural speech synthesis with transformer network. Proceedings of the AAAI Conference on Artificial Intelligence,

Liu, D.-R., Yang, C.-Y., Wu, S.-L., & Lee, H.-Y. (2018). Improving unsupervised style transfer in end-to-end speech synthesis with end-to-end speech recognition. 2018 IEEE Spoken Language Technology Workshop (SLT),

Liu, J., Xie, Z., Zhang, C., & Shi, G. (2021). A novel method for Mandarin speech synthesis by inserting prosodic structure prediction into Tacotron2. *International Journal of Machine Learning and Cybernetics, 12*(10), 2809-2823.

Liu, S., Yao, Y., Liu, B., Zhu, R., & Ren, J. (2022). Multi-band discriminant speech synthesis analysis based on Natural Language Processing. 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP),

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., & Bengio, Y. (2016). SampleRNN: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*.

Ming, H., Lu, Y., Zhang, Z., & Dong, M. (2017). A light-weight method of building an LSTM-RNN-based bilingual TTS system. 2017 International Conference on Asian Language Processing (IALP),

Mu, Z., Yang, X., & Dong, Y. (2021). Review of end-to-end speech synthesis technology based on deep learning. *arXiv preprint arXiv:2104.09995*.

Nakayama, S., Tjandra, A., Sakti, S., & Nakamura, S. (2018). Speech chain for semi-supervised learning of japanese-english code-switching asr and tts. 2018 IEEE Spoken Language Technology Workshop (SLT),

Nekvinda, T., & Dušek, O. (2020). One model, many languages: Meta-learning for multilingual text-to-speech. *arXiv preprint arXiv:2008.00768*.

Novitasari, S., Tjandra, A., Sakti, S., & Nakamura, S. (2020). Cross-lingual machine speech chain for javanese, sundanese, balinese, and bataks speech recognition and synthesis. *arXiv preprint arXiv:2011.02128*.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Raffel, C., Luong, M.-T., Liu, P. J., Weiss, R. J., & Eck, D. (2017). Online and linear-time attention by enforcing monotonic alignments. International Conference on Machine Learning,

Shan, C., Weng, C., Wang, G., Su, D., Luo, M., Yu, D., & Xie, L. (2019). Investigating end-to-end speech recognition for mandarin-english code-switching. ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Shan, C., Xie, L., & Yao, K. (2016). A bi-directional lstm approach for polyphone disambiguation in mandarin chinese. 2016 10th International Symposium on Chinese Spoken Language Processing (ISCSLP),

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., & Skerrv-Ryan, R. (2018). Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Sotelo, J., Mehri, S., Kumar, K., Santos, J. F., Kastner, K., Courville, A., & Bengio, Y. (2017). Char2wav: End-to-end speech synthesis.

Sreeja, M. R., & Ilankumaran, M. (2020). Remedies over the problem of teaching English as a second language to the rural students. *Journal of Xi'an University of Architecture & Technology, 12*(VIII), 1156-1170.

Tjandra, A., Sakti, S., & Nakamura, S. (2017). Listening while speaking: Speech chain by deep learning. 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU),

Tokuda, K., Nankaku, Y., Toda, T., Zen, H., Yamagishi, J., & Oura, K. (2013). Speech synthesis based on hidden Markov models. *Proceedings of the IEEE, 101*(5), 1234-1252.

Tokuda, K., Yoshimura, T., Masuko, T., Kobayashi, T., & Kitamura, T. (2000). Speech parameter generation algorithms for HMM-based speech synthesis. 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100),

Valin, J.-M., & Skoglund, J. (2019). LPCNet: Improving neural speech synthesis through linear prediction. ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems,

Wang, W., Xu, S., & Xu, B. (2016). First Step Towards End-to-End Parametric TTS Synthesis: Generating Spectral Parameters with Neural Attention. Interspeech,

Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., & Bengio, S. (2017). Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*.

Xin, D., Komatsu, T., Takamichi, S., & Saruwatari, H. (2021). Disentangled Speaker and Language Representations Using Mutual Information Minimization and Domain Adaptation for Cross-Lingual TTS. ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Yang, B., Zhong, J., & Liu, S. (2019). Pre-Trained Text Representations for Improving Front-End Text Processing in Mandarin Text-to-Speech Synthesis. INTERSPEECH,

Yi, L., Li, J., Hao, J., & Xiong, Z. (2009). Improved grapheme-to-phoneme conversion for mandarin tts. *Tsinghua Science & Technology, 14*(5), 606-611.

Yu, M., Nguyen, H. D., Sokolov, A., Lepird, J., Sathyendra, K. M., Choudhary, S., Mouchtaris, A., & Kunzmann, S. (2020). Multilingual grapheme-to-phoneme conversion with byte representation. ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Zen, H., Braunschweiler, N., Buchholz, S., Gales, M. J., Knill, K., Krstulovic, S., & Latorre, J. (2012). Statistical parametric speech synthesis based on speaker and language factorization. *IEEE transactions on audio, speech, and language processing, 20*(6), 1713-1724.

Zhang, Y., Weiss, R. J., Zen, H., Wu, Y., Chen, Z., Skerry-Ryan, R., Jia, Y., Rosenberg, A., & Ramabhadran, B. (2019). Learning to speak fluently in a foreign language: Multilingual speech synthesis and cross-language voice cloning. *arXiv preprint arXiv:1907.04448*.

Zhao, S., Nguyen, T. H., Wang, H., & Ma, B. (2020). Towards natural bilingual and code-switched speech synthesis based on mix of monolingual recordings and cross-lingual voice conversion. *arXiv preprint arXiv:2010.08136*.

Zhao, S., Wang, H., Nguyen, T. H., & Ma, B. (2021). Towards Natural and Controllable Cross-Lingual Voice Conversion Based on Neural TTS Model and Phonetic Posteriorgram. ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Appendix A

Pinyin Initials and Finals

This appendix presents Pinyin initials and finals.

Table 1: Pinyin Initials

General Initials	b, p, m, f, d, t, n, l, g, k, h, j, q, x
Blade-alveolar Initials	z, c, s
Cacuminal Initials	zh, ch, sh, r
Zero Initials	y, w

Table 2: Pinyin Finals

Single Finals	a, o, e, i, u, ü
Complex Finals	ai, ei, ui, ao, ou, iu, ie, üe, er
Front Nasal Finals	an, en, in, un, ün
Rear Nasal Finals	ang, eng, ing, ong
Overall Recognition Syllables	zhi, chi, shi, ri, zi, ci, si, yi, wu, yu, yin, yun, ye, yue, yuan, ying

Appendix B

Source Code Documentation

This appendix shows the source code documentation that we used in our experiment.

1.0 Introduction

The source code of our experiment is mainly on the basis of the open source end-to-end speech synthesis model Tacotron https://github.com/andi611/CS-Tacotron-Pytorch.git, g2pM https://github.com/kakaobrain/g2pM, python-pinyin https://github.com/mozillazg/python-pinyin, and SSML of Aliyun TTS based on version 1.1 of W3C SSML. We have modified them to meet our research goal and settled them together inside CS_TTS.zip.

1.1 File Structure

-**ckpt** folder: checkpoints of saved models.

-**data** folder: Chinese-English code-switching dataset TAL_CSASR downloaded from https://ai.100tal.com/dataset for further model training and testing.

-**image** folder: visualization of the pre-processed speech waveform samples and training alignment curves.

-**result** folder: synthesized audio speech, and the corresponding spectrogram visualization and alignment curves.

-**src** folder: training code and testing code of TTS model.

-**model** folder: attention and tacotron.

-**utils** folder: text-frontend normalization rules and plot setting.

-**config.py**: hyperparameter configuration for TTS model.

-**preprocess.py**: pre-process the speech and text of the original data.

-**train.py**: train TTS model.

-**test.py**: test TTS model.

2.0 Environment setting

Install python3 and the latest version of pytorch.

We use google colab to set our experiment. We can use the following command to check the environment:

```
!nvidia-smi
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   35C    P0    28W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Use the following command to install the requirement:

```
!pip install -r requirements.txt
!pip install pypinyin
!pip install unidecode
!pip install librosa==0.7.2
!pip install numba==0.48.0
!pip install pydub
!pip install nnmnkwii
!pip install tensorboardX
```

3.0 Hyperparameter configuration

This is an example part of the hyperparameter configuration setting. We set the sequence-to-sequence target outputs_per_step with 6, which means we use a six-layer unidirectional sequence-to-sequence gated recurrent unit to predict non-overlapping multi-frame outputs at each decoder step at the same time, which can speed up the loss convergence of the attention mechanism.

We train the model by Adam optimizer with the initial learning rate as 0.002, adam_beta1 parameter as 0.9, dam_beta2 parameter as 0.999.

```
def get_model_config(self):
  self.embedding_dim = 256
  self.outputs_per_step = 6
  self.padding_idx = None
  self.use_memory_mask = False

def get_dataloader_config(self):
  self.pin_memory = True
  self.num_workers = 2

def get_training_config(self):
  self.batch_size = 16
  self.adam_beta1 = 0.9
  self.adam_beta2 = 0.999
  self.initial_learning_rate = 0.002
  self.decay_learning_rate = True
  self.nepochs = 1000
  self.weight_decay = 0.0
  self.clip_thresh = 1.0
  self.checkpoint_interval = 2000
```

Use the following code to realize warm-up learning rate with the warmup_steps as 6,000. After the 6,000th step, the learning rate is time decaying.

```
#---decay learning rate---#
current_lr = _learning_rate_decay(init_lr, global_step)
for param_group in optimizer.param_groups:
  param_group['lr'] = current_lr
```

```
def _learning_rate_decay(init_lr, global_step):
  warmup_steps = 6000.0
  step = global_step + 1.
  lr = init_lr * warmup_steps**0.5 * np.minimum(step * warmup_steps**-1.5, step**-0.5)
  return lr
```

4.0 Pre-process of speech and text

Use the following command to pre-process the speech and get the corresponding visualization file.

```
!python3 preprocess.py --mode audio --audio_input_dir <path of speech
file> --audio_output_dir <path of speech_processed file> --
visualization_dir <path of speech_visualization file>
```

Use the following command to pre-process the text.

```
!python3 preprocess.py --mode text --text_input_raw_path <path of text
file> --text_pinyin_path <path of text_pinyin file>
```

Use the following command to get model-ready meta files that combined the text and speech information.

```
!python3 preprocess.py --mode meta --text_pinyin_path <path of text_pinyin
file>--audio_output_dir <path of speech_processed file>
```

```
########
# MAIN #
########
def main():

  args = get_preprocess_args()

  #---preprocess text---#
  if args.mode == 'all' or args.mode == 'text':
    try:
      mapper = data.get_mapper(os.path.join(args.text_dir, args.mapper_path))
      process_text(mapper, input_path=os.path.join(args.text_dir, args.text_input_train_path), output_path=os.path.join(args.text_dir, args.text_output_train_path))
      process_text(mapper, input_path=os.path.join(args.text_dir, args.text_input_dev_path), output_path=os.path.join(args.text_dir, args.text_output_dev_path))
      process_text(mapper, input_path=os.path.join(args.text_dir, args.text_input_test_path), output_path=os.path.join(args.text_dir, args.text_output_test_path))
    except: pass
    process_pinyin(args.text_pinyin_path,
                   args.text_input_raw_path,
                   args.text_dir,
                   args.all_text_output_path,
                   [args.text_output_train_path,
                   args.text_output_dev_path,
                   args.text_output_test_path],
                   join=args.join)

  #---preprocess audio---#
  elif args.mode == 'all' or args.mode == 'audio':
    process_audio(args.audio_input_dir,
                  args.audio_output_dir,
                  args.visualization_dir,
                  target_dBFS=-10.0,
                  file_suffix='.wav',
                  start_from=0,
                  num_workers=args.num_workers,
                  tqdm=tqdm,
                  vis_process=True,
                  vis_origin=False)
    data.check(args.audio_input_dir, args.audio_output_dir, file_suffix='*.wav')

  #---preprocess text and data to be model ready---#
  elif args.mode == 'all' or args.mode == 'meta':
    make_meta(args.text_pinyin_path, args.audio_output_dir, args.meta_audio_dir, args.meta_text, args.num_workers, config.frame_shift_ms)

  #---dataset analysis---#
  elif args.mode == 'all' or args.mode == 'analysis':
    dataset_analysis(args.audio_input_dir, args.text_dir, [args.text_output_train_path, args.text_output_dev_path, args.text_output_test_path])

  else:
    raise RuntimeError('Invalid mode!')
```

5.0 Train TTS model

Use the following command to train the model:

```
!python3 train.py
```

Use the following code to load saved models with checkpoint to achieve continue train the model:

```
  # Load checkpoint
  if checkpoint_path != None:
    print("Load checkpoint from: {}".format(checkpoint_path))
    checkpoint = torch.load(checkpoint_path, map_location="cuda:0")
    model.load_state_dict(checkpoint["state_dict"])
    optimizer.load_state_dict(checkpoint["optimizer"])
    try:
      global_step = checkpoint["global_step"]
      global_epoch = checkpoint["global_epoch"]
    except:
      print('Warning: global step and global epoch unable to restore!')
      sys.exit(0)

  return model, optimizer, data_loader


########
# MAIN #
########
def main():

  args = get_training_args()

  global DATA_ROOT, META_TEXT
  if args.data_root != None:
    DATA_ROOT = '/content/CS_TTS/data/meta'
  if args.meta_text != None:
    META_TEXT = '/content/CS_TTS/data/text/meta_text.txt'

  checkpoint_dir = '/content/drive/MyDrive/CS/1005'
  checkpoint_path = args.checkpoint_path
  os.makedirs(checkpoint_dir, exist_ok=True)

  model, optimizer, data_loader = initialize_training(checkpoint_path)

  # Train!
  try:
    train(model, optimizer, data_loader, args.summary_comment,
        init_lr=config.initial_learning_rate,
        checkpoint_dir=checkpoint_dir,
        checkpoint_interval=config.checkpoint_interval,
        nepochs=config.nepochs,
        clip_thresh=config.clip_thresh,
        sample_rate=config.sample_rate)
  except KeyboardInterrupt:
    print()
    pass

  print("Finished")
  sys.exit(0)
```

Use the following command to monitor the training curves through TensorboardX:

```
%load_ext tensorboard
%tensorboard --logdir <path of log file>
```

6.0 Test TTS model

Use the following command to test the model:

```
!python3 test.py --plot --model --test_file_path <path of test_text file>
```

7.0 PESQ evaluation

Use the following command to convert the predicted speech to 16k audio clips after install sox tool:

```
!sudo apt install -q -y sox
!mkdir ./wav
!echo "normalize audio clips to sample rate of 16k"
!find ./CS -name "*.wav" -type f -execdir sox --norm=-3 {} -r 16k -c 1
`pwd`/wav/{} \;
!echo "Number of clips" $(ls ./wav/ | wc -l)
```

Use the following command to run the PESQ algorithm:

```
pesq +16000 <path of reference_speech> <path of test_speech>
```