Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
Pfaffenwaldring 5B
D-70569 Stuttgart

Master thesis

Evaluation of transfer learning methods in Text-to-Speech

Zhenliang Zhou

Studiengang:	M.Sc. INFOTECH
Prüfer*innen:	Prof. Dr. Ngoc Thang Vu
	Dr. Antje Schweitzer
Betreuer:	Ms. Julia Koch
	Prof. Dr. Thang Vu
Beginn der Arbeit:	23.11.2022
Ende der Arbeit:	23.06.2023

Erklärung (Statement of Authorship)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und dabei keine andere als die angegebene Literatur verwendet habe. Alle Zitate und sinngemäßen Entlehnungen sind als solche unter genauer Angabe der Quelle gekennzeichnet. Die eingereichte Arbeit ist weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen. Sie ist weder vollständig noch in Teilen bereits veröffentlicht. Die beigefügte elektronische Version stimmt mit dem Druckexemplar überein.[1]

(Zhenliang Zhou)

[1]Non-binding translation for convenience: This thesis is the result of my own independent work, and any material from work of others which is used either verbatim or indirectly in the text is credited to the author including details about the exact source in the text. This work has not been part of any other previous examination, neither completely nor in parts. It has neither completely nor partially been published before. The submitted electronic version is identical to this print version.

Contents

List of Figures

List of Tables

1 Introduction

For the classic pre-trained model in natural language processing (NLP)[1], a large-scale designed model will be trained for multi-task in the specific domain. Aiming to swiftly obtain a model which could tackle a single special task, transfer learning[2] is one perfect solution. Instead of retraining the entire model(full fine-tuning), some transfer learning methods[2] take the layers from the large-scale pre-trained model and freeze them so as to avoid destroying any of the information they contain during future training rounds. In this paper, we will apply three methods to achieve transfer learning[2] with smaller datasets and fewer steps in text-to-speech research.

As an alternative for transfer learning, the adapter module is proposed to solve Parameter validity issues. Adapter modules[3] yields a compact and extensible model. We add only a few trainable parameters per task, and new tasks can be added without revisiting previous ones. The parameters of the original network remain fixed, yielding a high degree of parameter sharing. Therefore, Adapters apply smaller task-specific modules which are inserted inside the entire model. This method does not require accessing all tasks during training. Normally, as new task flows arrive and are discovered, the adapter layer can match the performance of a fully fine-tuned model, while each sub-task requires a small amount of parameters' variation(on average).

Bitfit[4], a way of sparsity tuning, only requires adjustments to the bias items of the model. It is effective on small and medium-sized datasets and comparable with the other sparse fine-tuning methods on huge data. In addition to its practicability, it can better enable us to understand fine-tuning and learning knowledge in model training rather than learning new domain tasks in new fields. Experiments show that it is very effective to adjust bias parameters only, and even the bias between the encoder and voice decoder can only be adjusted, accounting for only 0.04% of the total parameters. This result makes it practical to deploy fine-tuned tasks in many environments with limited memory and opens the way for trainable models with fixed parameters. It leads to research directions to explore the role of bias in the pre-training network in the fine-tuning process.

In this paper, Diff-pruning is similar to Adapters, but instead of modifying the

structure of the model, Diff-pruning [5]extends the base model through a task-specific trainable diff vector, which helps to finetune 0.1-1% of the pre-training parameters. To learn this vector, we re-parameterize the model parameters for a particular task as $\theta_{task} = \theta_{pretrained} + \delta_{task}$, in which the parameter vector $\theta_{pretrained}$ is fixed, and task-specific diff vector is fine-tuned. The difference vector is reconstructed with a fine-tuned approximation of the L0-norm penalty to encourage sparsity[6].

In the text-to-speech research, we'll use Fastspeech2[7] as our base model. Fastspeech2 model contains phoneme embedding layer[8], encoder layer, variance adapter layer, and mel-spectrum decoder layer. Compared to the Fastspeech model, fastspeech2 has a much more outstanding structure. On the one hand, the Fastspeech2 model is trained with ground-truth speech instead of the outcome from the teacher used in the Fastspeech model. On the other hand, the duration, pitch, and energy information of speech is extracted by the Fastspeech2 model(extracted by duration, pitch, and energy predictor from variance adapter layer), which could help to expand the text sequence to match the speech detail information of mel-spectrum. This structure could solve the one-to-many problem effectively.

To demonstrate the effectiveness of adapter, Bitfit[4], and diff-pruning structure applied in the text-to-speech domain. We intend to pre-train the Fastspeech2 model for multiple speakers with the LibriTTS dataset first. Then we propose to add these new structures to the Fastspeech2 model and retrain the model with LJSpeech[9], a single-speaker English dataset consisting of 13100 short audio clips of a female speaker reading passages from 7 non-fiction books, approximately 24 hours in total. Other than that, We use vctk single speaker dataset to finetune the Fastspeech model with adapter, BitFit, diff pruning, and full finetuning to evaluate the performance of these models in low resources datasets.
We aim to compare the results between the full finetuning Fastspeech2 model and our new model with adapter, Bitfit[4], and diff-pruning and try to find whether the new transfer model will improve the parameters of single-speaker training.

2 Related works

There are different model structures applied in text-to-speech. Wavenet [10], Tacotron [11]series, Fastspeech[12] series, VITS[13], and PromptTTS are very popular models, which can achieve a very high degree of imitation.

WaveNet replaces the traditional approach of using Fourier transform for audio signals, which is the first neural network to achieve text-to-speech tasks. Therefore, the transformation can be backpropagated, and the original audio data can be processed by some techniques, such as dilated convolution, 8-bit quantization, .etc. But people have been investigating ways to combine WaveNet's method with traditional methods, although this method's loss function is calculated by multiple regression instead of classification used by WaveNet[10].

Tacotron[11] is essentially an end-to-end model with an attention mechanism[14], which consists of an Encoder and a Decoder. In the encoder structure, The phoneme is obtained from the raw text as input. Pre-net is composed of a fully connected layer plus dropout to improve the generalization ability and accelerate the convergence of the model. Then, CBHG is Used to extract high-level features from sequences to improve the generalization ability of the model. Also, k 1-D convolutions of different sizes are added inside the encoder, similar to the idea of the n-gram language model, which extracts context information on different lengths and piles the results together after padding. Residual connection: The output of the convolutional layer and the output of the pre-net are added to solve the problem of gradient disappearance that may be caused by the great depth of the network. Highway: The input is activated by ReLu[15] and Sigmoid function respectively, after passing through a fully connected layer of the first layer. The highway layer mitigates the overfitting problem caused by the deep network and reduces the training difficulty of the deep network. Finally, the output is obtained by bidirectional RNN[16]. As for the decoder, Pre-net, Decoder-RNN[16], and post-processing models are implemented. The new structures are proved to be effective in reducing model training time and improving convergence speed.

On the base of Tacotron[11], Tacotron2 makes some structural improvements.

CBHG is replaced by three-layer convolution and bidirectional LSTM[17], which is more concise. Location-sensitive Attention Mechanism: Reducing repetition or forgetting of sub-sequences during decoding. The Decoder generates only one frame at a time, replaces the Linear-frequency scale spectrum with a lower-level mel-frequency spectrum, and adds a post-net to finetuning because this representation of time-domain waveforms is easier for subsequent calculations.

Fastspeech[12] is a branch of the test-to-speech model. In 2019, Zhejiang University and Microsoft Research published a paper together, FastSpeech: Fast, Robust and Controllable Text to Speech. In this paper, Fastspeech is first proposed. The overall framework of Fastspeech is similar to the Transformer[18] Encoder, which can be simply understood as removing the Transformer module[18] of the Decoder to achieve parallel training of the model and speed up inference. Fastspeech is mainly composed of three parts: FFT Block[19], Length Regulator and Duration Predictor.

Compared with the previous generation Fastspeech, the new proposed Fastspeech2 model has several innovations. Instead of the aligned, synthesized spectrum from the Fastspeechlearning teacher model, the new model directly leverages external alignment tools[20] to provide duration information. Besides duration, the fundamental frequency and energy of speech are modeled separately at the same time. These strategies effectively improve the model's ability to extract speech information to fit the audio file closer to the real human voice.

PromptTTS: Text descriptions as hints to guide the generation of text or images (e.g., GPT-3 [21]or DALLE-2) have recently attracted much attention. In addition to text and image generation, the authors explore the possibility of using text descriptions to guide speech synthesis in this paper. Therefore, they developed a text-to-speech (TTS) system (called PromptTTS) that takes prompts with style and content descriptions as input to synthesize the corresponding speech. Specifically, PromptTTS consists of a style encoder and a content encoder to extract the corresponding representations from the prompts and a speech decoder to synthesize speech based on the extracted style and content representations. Compared to previous controllable TTS works, which required the user to have acoustic knowledge to understand style factors such as rhythm and tone.

3 Fastspeech2 and transfer learning methods

Normally, Text-to-speech is considered a one-to-many mapping problem in NLP[1]. There will be several speech information variations, including pitch, duration, energy, and volume. Therefore, the input texts will correspond to multiple possible pronunciation sequences. Sometimes, the text information is not sufficient to support speech generation. In this case, the model could easily meet the overfitting problem during the model training. In the Fastspeech model, three main issues are discussed: 1) the training pipeline is complicated; 2) the duration information cannot be calculated accurately. 3) information loss of mel-spectrogram. Therefore, the Fastspeech2 model is proposed. In this model, the training pipeline is simplified efficiently. And author adds a variance predictor layer to extract speech information. In the following subsection, we will introduce the structure advantage in Fastspeech2 model[7].

3.1 Fastspeech2 model

The Fastspeech2 model architecture[7] is shown in Figure 1a. First, the phoneme embedding layer[8] could transfer words to embedding vectors for further calculation. Positional encoding[22] inserts the positional information into the vectors to represent the logical relationship inside sentences. The encoder converts phoneme embedding[8] sequence into phoneme hidden sequence. Variance adapter could add duration, pitch, and energy variance information. Mel-spectrogram decoder converts the adapted hidden sequence into the hidden mel-spectrogram sequence in parallel. Fastsoeecg2 has six feed-forward Transformer blocks[18] in the encoder and decoder. The feed-forward transformer block[18] contains a stack of self-attention layers[14] and 1D-convolution layers as the Fastspeech model. In addition, Fastspeech2 make some updates to the structure. First, the teacher-student distillation pipeline in Fastspeech is removed, and Fastspeech2 adopts ground-truth mel-spectrograms as training targets. This change effectively avoids mel-spectrogram information loss

and improves the audio quality of training targets. Then, the variance adapter layer contains energy, pitch, and energy predictors. These predictors could extract energy, pitch, and energy information from training audio signals. 1) the duration predictor extracts the phoneme duration information collected by forced alignment[20] as training target duration, which is more accurate than the duration information collected from the attention map of the teacher model in Fastspeech. 2) the pitch and energy predictors can provide more accurate information on variance. These structures can effectively solve the one-to-many mapping problem in the Test-To-Speech problem. And the detail of each structure in Fastspeech2 will be introduced in the following subsections.

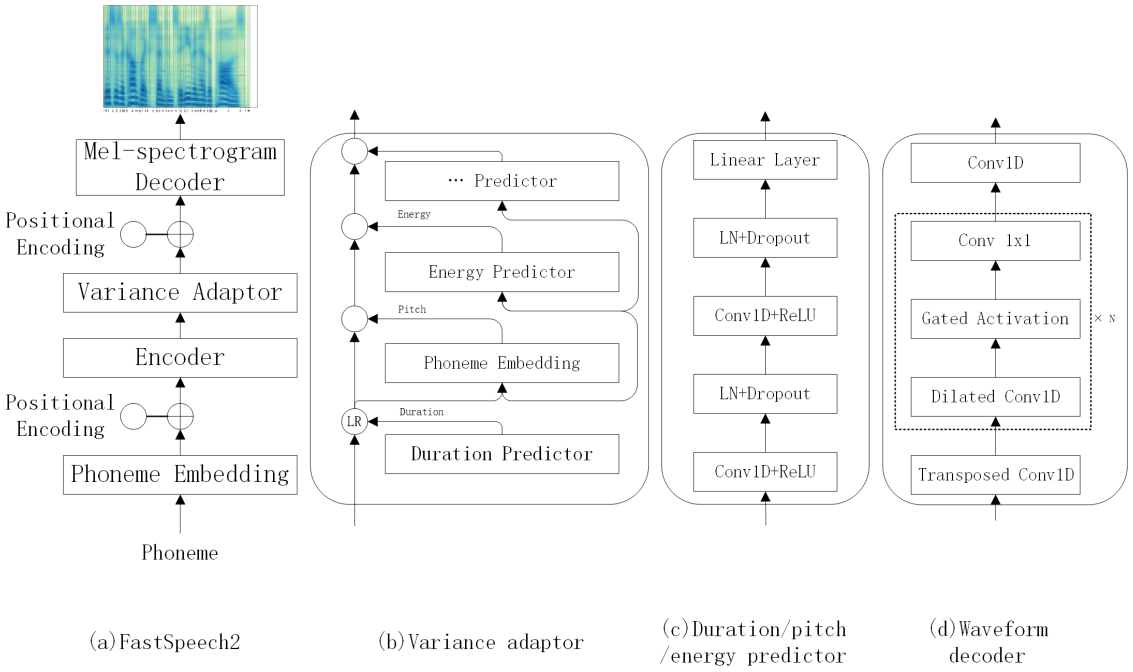

Figure 1. Fastspeech2 model

3.1.1 Phoneme Embedding

When word embedding is applied to speech recognition and speech synthesis tasks, the results are not as good as those in NLP [1]tasks. The main reason is that the semantic and syntactic information extracted by word embedding is difficult to integrate into speech-related tasks directly. Therefore, in text-to-speech, a phoneme-based embedding method is proposed, unlike word embedding. The phonetic characters represent the pronunciation of phoneme sequences to generate phoneme vectors. In phoneme embedding training[8]. The input is a phoneme label, and the output is a corresponding acoustic feature. Word vectors can obtain semantic and syntactic information by training neural network models related to the use of words and phrases in the language. In contrast, the purpose of phoneme embedding[8] is to capture acoustic information(such as speech features) and represent this information as a phoneme vector.
In the phoneme embedding[8] training, the input is phoneme labels, and the output is the corresponding acoustic features. Specifically, the one-hot coding of phoneme labels is embedded into the layer to generate phoneme vectors, which will become the inputs to the bidirectional long and short time memory(BLSTM)[23] cyclic neural network(RNN) regression model[16] to predict acoustic features. The phoneme embedding[8] analysis shows that the phoneme vector has some interesting characteristics. Phonemes with similar acoustic properties are close in cosine distance in the generated phoneme vector space and tend to be similar after k-means clustering. And in the Fastspeech2 model, phoneme labels will be extracted from the text of training samples.

3.1.2 positional Encoding

Positional Encoding[22] is an important technique in the Fastspeech2 model[7] for encoding positional information for each position in the input sequence. In the Fastspeech2 model, there are only attention mechanisms[14] and fully connected neural networks, and no structures that can process sequences like RNNS[16] or CNNS.

Therefore, the role of Positional encoding[22] is to provide the model with positional information relative to each position in the sequence, ensuring that the model considers positional order when processing inputs.

3.1.3 Encoder and Decoder

Fastspeech2[7] uses a new feed-forward transformer[18] network architecture that abandons the traditional encoder-attention-decoder mechanism. Its main modules adopt Transformer's self-attention mechanism[18, 14] and 1D Convolution network. This structure is considered a Feed-Forward Transformer Block(FFT Block)[19]. Multiple FFT blocks are stacked in a feedforward Transformer for Phoneme to mel-spectrum transformation. There are N FFT blocks on the encoder and decoder. In particular, there is a length regulator in the middle, which regulates the length difference between the phoneme sequence and the mel-spectrum sequence.

3.1.4 Variance Adapter

The purpose of a variable adapter is to add variable information(for instance, pitch duration and energy, etc.) to a hidden phoneme sequence. This mechanism can provide sufficient information for the prediction of phonological variants, as well as one-to-many problems in TTS. We would like to introduce the variance information as follows:
1) Phoneme duration indicates the duration of pronunciation.
2) Pitch can convey emotional information, which affects speech prosody greatly.
3) Energy directly influences the loudness of the audio and correlates with the audio's amplitude.
Besides, the variance adapter can add more variance information, such as speaking style and emotions. Therefore, this structure provides strong scalability for the Fastspeech2 model. Duration, pitch, and energy structures are shown in Figure 1b. During training, the truth values of duration, pitch, and energy information are

extracted from a recording, which will be implemented into a sequence to predict the target speech. In the meantime, the ground-truth duration, pitch, and energy information will be used to train the duration, pitch, and energy predictors. These trained structures will be used to synthesize target speech. We can see that the duration, pitch, and energy predictors share a similar structure, which is shown in Figure 1c. The structure consists of a 2-layer 1D-convolutional network[24] with ReLU[15] activation function. Behind the output of the activation function, a normalization layer, one dropout layer, and a hidden linear layer are added. We will introduce the details of these three predictors in the following paragraphs.

Duration Predictor: This adapter predictor regards a hidden phoneme sequence as input and predicts the duration of each phoneme. The duration information will indicate how many mel-frames correspond to the hidden phoneme sequence. For predictability, the duration information is converted into logarithmic domain
The duration adapter predictor is optimized with mean square error loss(MSE)[25]. Rather than using a pre-trained auto-regressive TTS model[26] to extract phoneme duration in Fastspeech, Fastspeech2 uses Montreal Forced Alignment(MFA) tool[20] to obtain the phoneme duration information, which can improve alignment accuracy and reduce information difference between model input and output[7].

Pitch Predictor: The previous neural network designed to predict pitch information always predicts pitch contour. But, since the pitch of ground truth has high variations, it is hard to predict the pitch values. And the results are very different from ground truth distribution. Therefore, the continuous wavelet transform(CWT)[27] is used to decompose the continuous pitch series into pitch spectrograms. Then the pitch spectrograms will be designed as training targets for the pitch predictor optimized with MSE loss[25]. And in inference, the pitch predictor will predict the pitch spectrogram. Then the spectrogram will be converted back into pitch contour using inverse continuous wavelet transform(iCWT). After extracting the pitch information, the pitch contour should be set as input in both training and inference. In the

original paper, The authors quantize pitch F_0 of each frame to 256 possible values. Then those will be further converted into pitch embedding vector p. Finally, the corresponding pitch embedding vector is added to the expanded hidden sequence.

Energy Predictor: The energy is calculated by the L2-norm of the amplitude of each short-time Fourier transform(STFT) frame. The same as the pitch predictor, the authors quantize the energy of each frame to 256 possible values. Those values are encoded into energy embedding and then added to the expanded hidden sequence[7].

3.2 Background of Transfer Learning:

Transfer learning[1] mostly uses a pre-train and finetune pattern. The pretraining initializes the model parameter and finetunes on a task-specific objective. Pretraining objectives include autoencoding[28], context prediction, text-to-speech as well as variants of language modeling.
In this thesis, we construct a mathematical expression for transfer learning[2]. We bring up a set with a potentially unknown set of tasks $\tau \in T$ with a training set $D_T = \{x_\tau^{(n)}, y_\tau^{(n)}\}_{n=1}^N$. For finetuning tasks, the final target is to modify model parameters θ_τ and minimize the training loss.

$$\min_{\theta_\tau} \frac{1}{N} \sum_{n=1}^{N} C(f_\tau(x^{(n)}; \theta_\tau), y_\tau^{(n)}) + \lambda R(\theta_\tau) \tag{1}$$

In the function 1, $f_\tau(\cdot; \theta_\tau)$ is the parameterized function with the input x. And $C(\cdot, \cdot)$ is a loss function(eg, cross-entropy or L1-loss).$R(\cdot)$ is an additional regularizer with hyperparameter λ. With this transfer learning method[2], the model can learn independent parameters for each downstream task. However, the dramatic variation of parameters in the pre-trained models makes this approach considerably parameter inefficient. Take an example in NLP[1], the widely-adopted models, such as $BERT_{BASE}$ as well as $BERT_{LARGE}$ contain approximately 110M and 340M parameters, respectively.

As we can imagine, storing a fully finetuned model consumes many resources, making it hard even to store a moderate number of tasks. A classic approach to tackling the parameter-inefficiency is to deploy a single shared model(with a task-specific output layer). This approach is comparable to multiple tasks through joint training. But the single shared model requires the set of tasks T to be known and solved in advance to prevent catastrophic forgetting. Therefore, this method is unsuitable for those unknown tasks when they arrive in the overall task stream.
In the following sections, we intend to introduce four transfer learning methods. And we would implement these methods in Text-to-Speech and make some evaluations to test the methods' performance.

3.3 Adapter structure

3.3.1 Introduction:

In the context of transfer learning[2] in NLP[3], parametric inefficiency occurs when a completely new model needs to be trained for each downstream task, and the number of parameters becomes too heavy to train.
The traditional fine-tuning model copies the weight from the pre-trained neural network. For solving new downstream tasks, the fine-tuning model has to finetune its parameters, which requires a new set of weights while retraining the model. In other words, the parameters are finetuned along with each layer in the model for each task. The advantage of fine-tuning is that parameter efficiency may be higher if the lower layers of the network are shared between tasks.
A recent paper proposes an adapter module in NLP[1] that provides parametric efficiency by adding only a few trainable parameters per task. As new tasks are added, previous tasks do not have to be revisited.
The main idea of this module is to enable the transfer learning of NLP[1] on the new downstream tasks rather than having to train a new model for these tasks.

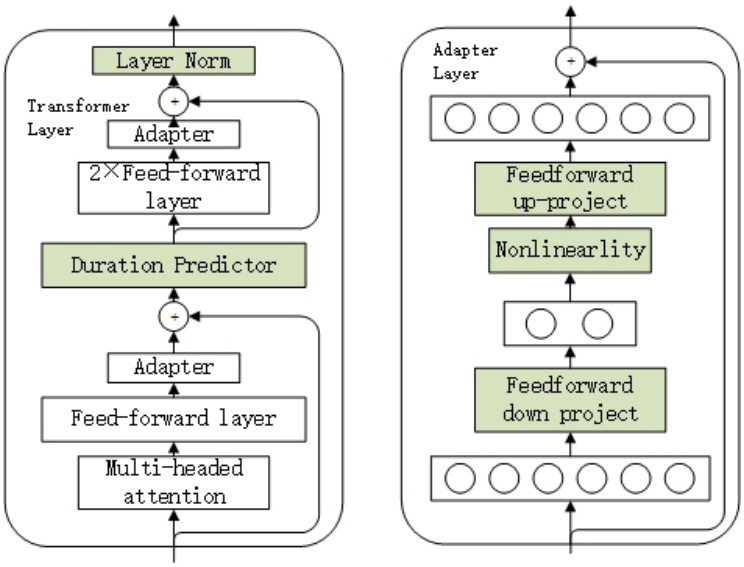

Figure 2. Adapter Model in Transformer

3.3.2 Adapter structure in NLP

The proposed structure of the adapter layer is first applied in Transformer(Figure 2 left). The structure of the adapter layer[3] is shown in Figure 2 (right). Each layer of the Transformer model contains an attention layer[14] and a feed-forward layer. Both layers are followed by a projection that maps the feature size to the size of the layers' input. A skip connection is constructed across the two sub-layers. The output of each sub-layer will be added to the layer normalization. We can see that two serial adapter layers are inserted after each of these sub-layers. And the adapter layer is specially designed and is always applied directly to the output of the sub-layer after the projection to the input but before the skip connection.

A bottle-neck architecture is proposed to limit the number of parameters in the adapter layer. As shown in Figure 2(right), adapters project the original d-dimension features of layer input into a smaller dimension m. And after a non-linearity and feed-forward up-project are applied. The dimension is back to d. Therefore, the total parameter number added per layer, including biases, is $2md + d + m$. Due to the design of the bottle-neck structure, if setting $n \ll d$, we can limit the total param-

eter added per task. In NLP[1] tasks, the author uses around 0.5-8% of the entire original parameters to accomplish new downstream tasks. Because the adapter layer has a skip connection, if the parameters of the projection layers are initialized to zero matrices, the module will be approximately initialized to an identity function. For training the new downstream tasks, the parameters of the pre-trained network will be frozen(meaning they remain fixed), and only a few additional task-specific parameters, as well as parameters in the adapter layer, will be added for each new task. Training the adapter layer will not affect the previous parameters. The innovation of this method is that the new downstream tasks can be solved by tuning the pre-trained model with less trained parameters and less training time, but this adapter model can achieve the same performance as the finetuning method. Therefore, we propose the new Fastspeech2 model with an adapter layer and try to evaluate the performance of this model trained by a small single-speaker dataset in text-to-speech.

3.3.3 Fastspeech2 adapter model

As introduced before, Fastspeech2 as a non-autoregressive TTS model[26] has faster training speed and better speech quality compared with Tacotron[11] and Fastspeech model. And this model can be successfully used to deal with multi-speakers and multi-language speech generation tasks. In this case, Fastspeech2 has great performance trained by a large dataset.

However, with low resources, Fastspeech2 can not perform well as it trained with hours of speech audio. Therefore, finetuning is one solution to this task. Fastspeech2 can be trained with multi-speaker datasets as a pre-trained model. Then, we can build a small dataset with tens or hundreds of audio as a training dataset and re-train the entire Fastspeech2 model. This fine-tuning Fastspeech2 model could combine the prior information of the pre-trained datasets and the extracted tone and intonation from the tuning datasets to imitate the speaker, whose limited audio signals are collected.

The adapter Fastspeech2 model[7] resembles fine-tuning the Fastspeech2 idea. In the structure shown in Figure 3a, an adapter layer is added after the encoder layer

in Fastspeech2. And the adapter output will be used as the input of the variance adapter layer.

The adapter structure we use has the same structure as the adapter in Transformer[18], which we introduced in 3.6.2. As shown in Figure 3e, the input of the adapter layer firstly passes through a linear layer with dimension d, then forwards to a bottleneck layer designed to have dimension$n$, after calculating in non-linearity(ReLU). The bottleneck output restores to d dimension by mapping the linear layer. And the resulting output vector is summed directly with the input of the adapter layer. To reduce adapter layer parameters, we set the adapter's bottleneck dimension much smaller than the input and output dimension $n << d$.

In order to realize speech training with a small dataset, we first choose the multi-speaker dataset LibrTTS[29] for the pre-training model, which contains around 242 hours of utterances from speakers so that the model can learn the intonation, energy, pitch, and duration information of the speech datasets. In the subsequent transfer learning[2] training, we first freeze all the weight matrices and bias vectors of the Fastspeech2, including the embedding layer, variance adapter layer, and mel-spectrogram decoder layer. Then we add the adapter layer inside Fastspeech2 and reload all modules' pre-trained weight matrix data. To evaluate the performance of adapter Fastspeech2 in the speaker adaption task, we build a small dataset from a single speaker dataset for adaption finetuning and generating audio signals imitating the speaker.

3.4 BitFit: Simple Parameter-efficient Fine-tuning

3.4.1 Introduction:

As we introduced before, the Fastspeech2 model is pre-trained on large, multi-speaker datasets, and then finetuned on single-speaker supervised data. The large size of the model makes it expensive to train and deploy. Besides, the extent to which fine-tuning must change the original model led researchers to consider fine-tuning only a small subset of the model parameters. But the entire model can perform well as fine-tuning in the downstream tasks.

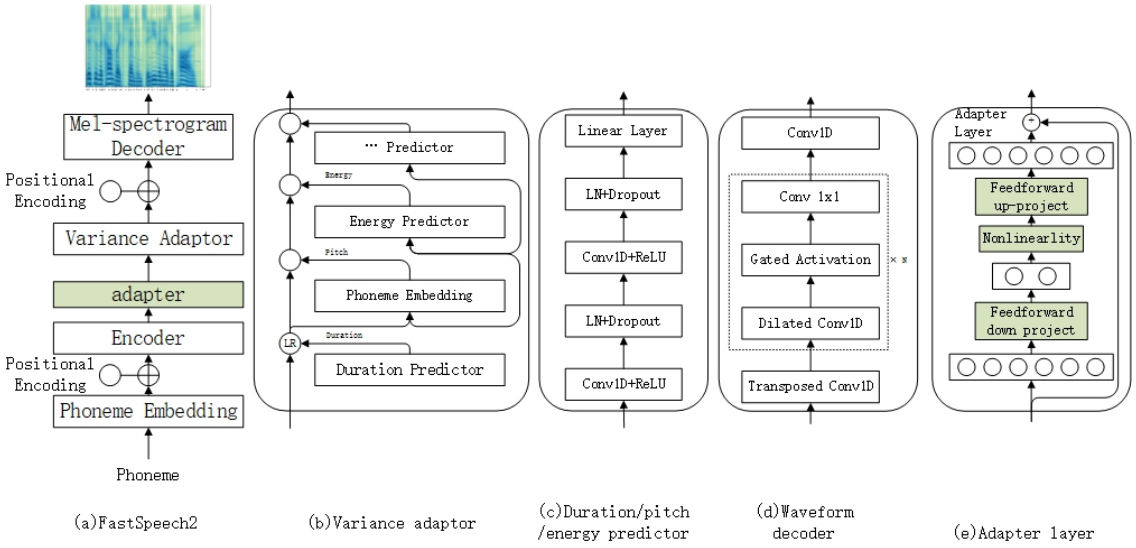

Figure 3. Fastspeech2 adapter model

3.4.2 BitFit algorithm:

The researcher proposed an effective but simple method of fine-tuning, which has several benefits[4]:

1. very few parameters will be changed per downstream task.

2. for every downstream task, the model changes the same subset of parameters

3. The variation of parameters is isolated across the entire parameter space of the model.

4. for small to medium training data, changing only parameters in bias vectors can reach the same or better results as full-finetuning with less training time.

More specifically, when applying the transfer learning[2] in the pre-trained model, the author shows that freezing most of the network and fine-tuning only the bias term is surprisingly effective. Moreover, in the original paper, the author proposes to fine-tune only two bias terms(the 'query' and 'middle-of-MLP'), whose parameters only account for half of the bias parameters in the model. This method is practical in applying fine-tuned models in memory-constrained environments. Since most of the trainable parameters in the model are fixed, it is deployable for some weak com-

putational hardware.
In addition, it provides a research direction regarding the role of the bias term in pre-trained models, and the dynamics of the fine-tuning process.

3.4.3 Bias-term Fine-tuning(BitFit) in Fastspeech2:

As introduced in the last subsection, the author proposed a method called BitFit(Bias-term Fine-tuning)[4]. Specifically, we should freeze most of the parameters in the weight matrix in the pre-trained model and train only the bias terms. In conclusion, BitFit[4] has three key properties: 1). fine-tune only a small portion of the model's parameters. 2). match the results of the full-finetuned model 3). enable tasks training in sub-parameters of the model.
This method is parameter-efficient: each new downstream task requires only bias terms parameters trained. (the proportion of trainable parameters accounts for less than 0.27% of the total parameters).
In specific, Fastspeech2 is composed of Phoneme embedding[8] layer, encoder layer, and mel-spectrogram decoder layer. The encoder and decoder are composed of L layers, where each layer l starts with M self-attention heads[30], in which a self-attention head(m, l) [14]has *key*,*query* and *value* encoders, each calculated with a linear equation:

$$Q^{m,l}(x) = W_q^{m,l}x + b_q^{m,l} \tag{2}$$

$$K^{m,l}(x) = W_k^{m,l}x + b_k^{m,l} \tag{3}$$

$$V^{m,l}(x) = W_v^{m,l}x + b_v^{m,l} \tag{4}$$

In these equations, x is the output of the previous encoder(But for the first encoder layer, s is the output of the phoneme embedding layer[8]). These vectors are combined by an attention mechanism[14], which does not include new parameters:

$$h_1^l = att(Q^{1,l}, K^{1,l}, V^{1,l}, ..., Q^{m,l}, K^{m,l}, V^{m,l}) \tag{5}$$

Besides, after we iterate over all trainable parameters in Fastspeech2, we still find the conv, linear, and normalization layers. All layer L could be expressed as:

$$L_n(x) = W_n x + b_n \tag{6}$$

All the collection of entire matrices W_n and vectors b_n belongs to Fastspeech2's parameters Θ. And the subset of red vectors b_n is the bias term. The subscript n represents the sequence number of layers in Fastspeech2.
The bias terms are additive parameters, corresponding to a very small proportion of the neural network(0.27%). We want to show that by freezing all the parameters W_n and training only the bias term b_n we can also acquire the same and even better audio quality in the Fastspeech2 model trained by small dataset.

3.5 Diff-pruning: a parameter-efficient transfer learning method

3.5.1 Introduction:

Except for adapter and BitFit[4], Diff pruning[5] also enables parameter-efficient transfer learning[2] that performs well with new downstream tasks. This method has the innovation in creating a task-specific 'diff' vector that extends the training parameters of the pre-trained model. This diff vector can be trained with a differentiable approach optimized by L_0-norm penalty to encourage sparsity[6]. The diff pruning is parameter-efficient, even if the number of downstream tasks increases. Since it only requires saving a small diff vector for each new task. Besides, diff pruning can also achieve the same performance of the full finetuning baseline on the Transformer model[18] in NLP[1], which only modifies 0.5% of the entire model.
To learn the diff vector in this algorithm, we can reparameterize the downstream task parameters as $\theta_{task} = \theta_{pretrained} + \delta_{task}$. In this function, the pre-trained parameter vector $\theta_{pretrained}$ is collected from the original trained model trained by a large dataset and the δ_{task} is the finetuned task-specific diff vector. The diff pruning model also adds the L_0-norm penalty to encourage sparsity[6]. Diff pruning is a parameter-efficient algorithm because it only stores the non-zero masks and weights

of the diff vector for each downstream task. When applied to the GLUE benchmark, diff pruning shows the likely performance compared to full-finetuning.

3.5.2 Diff pruning algorithm:

Diff pruning[5] solves the downstream tasks by training a diff vector δ_τ, Then the diff vector is added to the pre-trained model θ, which remains fixed during training. The main idea of diff pruning is formulated as follows:

$$\theta_\tau = \theta + \delta_\tau \tag{7}$$

In the formula(1), θ_τ represents the task-specific fintuned parameters. To optimize the downstream task, we choose the minimization method empirically.

$$\min_{\theta_\tau} L(D_r, f_r, \theta + \delta_r) + \lambda R(\theta + \delta_r) \tag{8}$$

In this formula, L is defined as training loss during optimizing. We can brevity design $L(D_r, f_\tau, \theta + \delta_r)$ as:

$$L(D_r, f_\tau, \theta + \delta_r) = \frac{1}{N} \sum_{n=1}^{N} C(f_\tau(x^{(n)}; \theta_\tau), y_\tau^{(n)}) \tag{9}$$

This formula(9) explains that the cost of storing the pre-trained parameters θ is separated across tasks. And the marginal cost for new downstream tasks is the diff vector. If we can apply an algorithm, which regularizes δ to be sparse such that $||\delta_\tau||_0 \ll ||\theta||_0$, then the downstream tasks will become more parameter-efficient. Therefore, a l_0-norm penalty is proposed for regularization, which is formulated as follows.

$$R(\theta + \delta_\tau) = ||\delta_\tau||_0 = \sum_{i=1}^{d} 1\{\delta_{\tau,i} \neq 0\}. \tag{10}$$

Differentiable approximation to L_0-norm Since the regularizer is designed as L_0-norm, the penalty term is not differentiable[5]. The optimization for regularizers is very difficult. In order to approximate this L_0-norm, an approach is applied for gradient-based learning with L_0 sparsity using a relaxed mask vector. This approach maps a binary vector into continuous space. And then, we multiply it with a dense weight vector to determine how much of the weight vector in the pre-trained model is applied during downstream training. After training, the mask is made deterministic, and a large proportion of the parameters in the diff vector is zero. To implement this approach, the algorithm decomposes δ_τ into a binary mask vector multiplied by a dense vector:

$$\delta_\tau = z_\tau \odot w_\tau, \quad z_\tau \in \{0,1\}^d, w_\tau \in \mathbb{R}^d. \tag{11}$$

Now, the algorithm wishes to lower bound the true objective and optimize an expectation with respect to z_τ. And α_τ is the trainable and additional parameters which are equivalent to w_τ plus the initial value of α_τ. The distribution of parameters z_τ and α_τ is defined as $p(z_\tau, \alpha_\tau)$, which is initially Bernouli.

$$\underset{\alpha_\tau, w_\tau}{min}\, \mathbb{E}_{z_\tau \sim p(z_\tau;\alpha_\tau)}[L(D_\tau, f_\tau, \theta + \delta_\tau) + \lambda||\delta_\tau||_0]. \tag{12}$$

In this step, this objective is still complicated since the binary mask vector z_τ is still discrete. But the expectation provides guidance for effective relaxation. In this paper (Louizos et al., 2018; Wang et al.,2019b)[31], the author came up with an algorithm. It can achieve a continuous relaxation of discrete random variables. By using this method, the binary mask vector z_τ can be mapped into continuous space $[0,1]^d$ with a stretched hard-concrete distribution, which ensures that the mapped mask vector is differentiable. Specifically, z_τ is now defined to be a deterministic and differentiable formula with a uniform distribution u.:

$$u \sim U(0,1), \tag{13}$$

$$s_\tau = \sigma(log(u) - log(1-u) + \alpha_\tau), \tag{14}$$

$$\overline{s}_\tau = s_\tau \times (r - l) + l, \tag{15}$$

$$z_\tau = min(1, max(0, \overline{s}_\tau)) \tag{16}$$

In formula(14), the σ is the activation function Sigmoid. In formula(15),$l < 0$ and $r > 1$ are two hyperparameters with constant values, which are used to stretch s_τ into the interval$(l, r)^d$ before it mapped to $[0, 1]^d$ by using the minimax method $min(1, max(0, \overline{s}_\tau))$. As a result, the expectation of L_0-norm could be formulated in a differentiable closed-form.

$$\mathbb{E}[||\delta_\tau||_0] = \sum_{i=1}^{d} \sigma\left(\alpha_{\tau,i} - log\frac{-l}{r}\right). \tag{17}$$

Therefore, the final optimization formula is expressed by:

$$\min_{\alpha_\tau, w_\tau} \mathbb{E}_{u\sim U[0,1]}[L(D_\tau, f_\tau, \theta + z_\tau \odot w_\tau) + \lambda \sum_{i=1}^{d} \sigma\left(\alpha_{\tau,i} - log\frac{-l}{r}\right). \tag{18}$$

Now, masks z_τ can be optimized using gradient estimation with respect to α_τ. Finally, we can calculate the diff vector δ_τ by sampling u the uniform distribution once to obtain z_r. The clamping function will set the majority of parameters to zero in the mask vector. But the mask vector can only become likely binary. Then we can obtain the sparse diff vector via $\delta_\tau = z_\tau \odot w_\tau$.[11], $\quad z_\tau \in \{0, 1\}^d, w_\tau \in \mathbb{R}^d$.

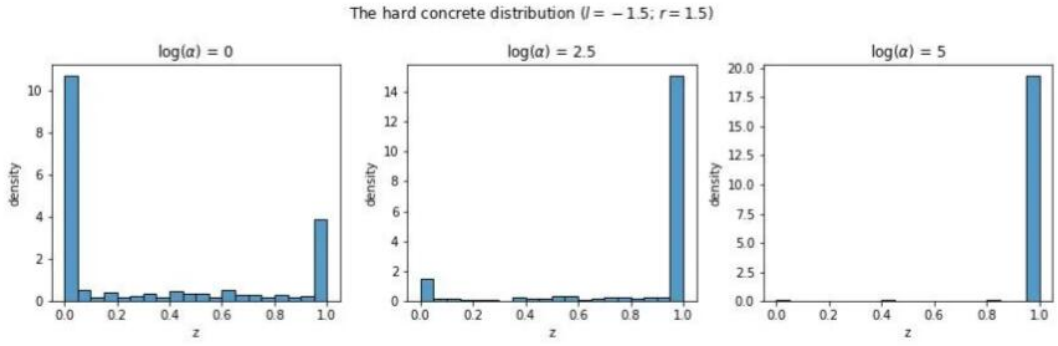

Figure 4. the hard concrete distribution

We found a graphic demonstration from (Lukas Hauzenberger: An practical Introduction to Diff-Pruning for BERT)[32]. The author explains the detail of the hard concrete distribution[31], which reflects the result of the mask vector if we set l and r as -1.5 and 1.5 in the formula(13-16). In conclusion, if the parameter α is smaller and closer to 0, the parameters in mask vector z_τ are more likely to be equal to 0. Therefore, the hyperparameter α typically should be initialized with a high value.

L_0-ball projection with magnitude pruning for sparsity control As introduced in the last section, the L_0 regularization of the diff pruning algorithm achieves a high sparsity rate[5]. However, the sparsity rate can not be controlled manually. It would be ideal to define an exact sparsity rate. Since the regularization coefficient λ is a Lagrangian multiplier for the expectation of the diff vector's zero norms $\mathbb{E}[||\delta_\tau||_0]$. The expectation is smaller than η, express as $\mathbb{E}[||\delta_\tau||_0] < \eta$. The sparsity control could be achieved by searching over different values of λ. But the author proposes a more effective method to achieve the sparsity control[6] by projecting onto a L_0-ball after training.
Specifically, magnitude pruning is proposed, which is applied on the diff vector δ_τ. The target vector only keeps the top t% of the parameters in each layer and keeps $t\% \times d$ values in δ_τ. $t\%$ represents the sparsity rate. It should be noted that this magnitude pruning is based on the diff vector rather than the model parameters.
In the diff pruning paper[5], the author has an empirical suggestion that they recommend finetuning further δ_τ with the nonzero masks in magnitude pruning for good performance. Since the magnitude pruning has parameter efficiency through projection onto the L_0-ball. And it can adapt to diff pruning easily.

Diff pruning and Magnitude pruning process As we introduced before, δ_τ is the diff vector during diff pruning fine-tuning, which is defined as:

$$\delta_\tau = z_\tau * (\theta_{full} - \theta_{pre}) \tag{19}$$

Therefore, we can define the output of diff pruning model θ_τ as:

$$\theta_\tau = \theta_{pre} + z_\tau * (\theta_{full} - \theta_{pre}) \tag{20}$$

Here $\theta_{full} = \theta_{pre} + \delta_\tau$ are the full finetuned weights. The gradient can be calculated using θ_{full}. Since the original fully finetuned and pre-trained weights are stored, the diff vector δ_τ could be recovered in each training step.

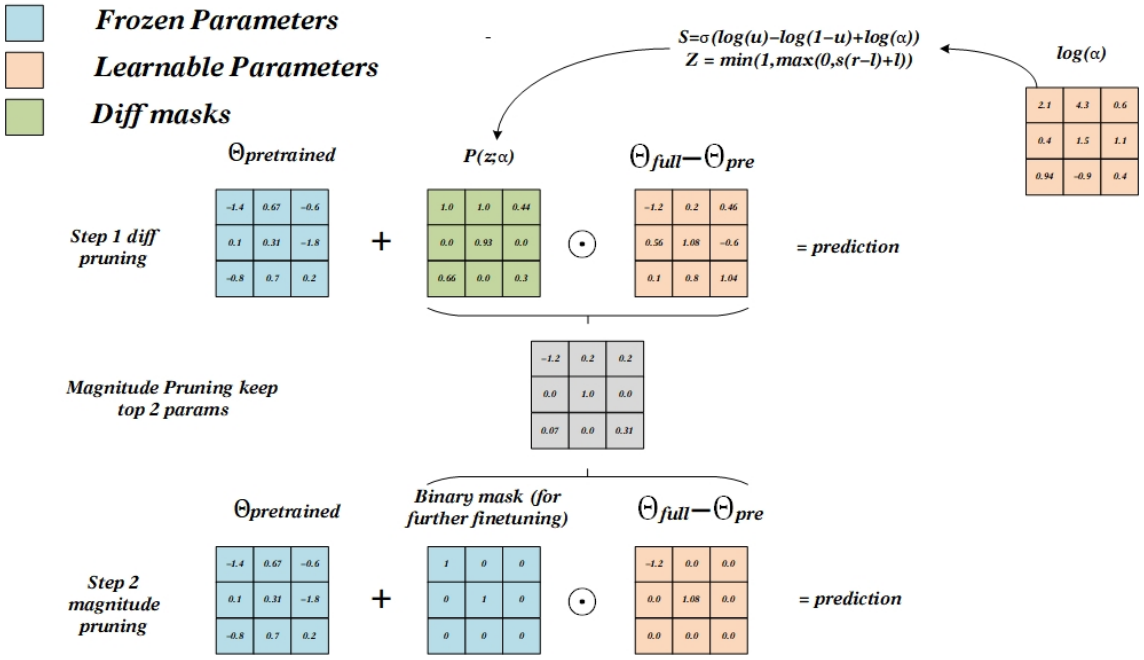

Figure 5. the diff pruning and magnitude pruning process

The diff pruning and magnitude pruning process is shown in Figure 5. Specifically, we create a vector for the trainable parameter α. α is first set as the addition by the pre-trained model θ_{pre} and a initial value α_{init}. :

$$\alpha = \theta_{pre} + \alpha_{init} \tag{21}$$

Based on the conclusion of Figure 4, we can see that the gate of diff mask z_τ will

be open(the value is 1) if we set α as a large value. Therefore, we normally choose 5 as the initial value for α_{init}.

After calculating across logistic domain mapping and the continuous relaxation of the discrete random variables, the diff mask is generated, which is demonstrated by the green mask in Figure 5. During the first training period, we choose the diff pruning algorithm[5] as the finetuning method.

First of all, the model should execute full-finetuning and obtain θ_{full} vector. Then the diff mask δ_τ of the current step will be calculated by the formula(19) graphically shown in Figure 5. Then the output vector of diff pruning algorithm θ_τ is generated with formula(20). And during each step of the first period, we calculated the normal task loss with some loss function (formula (12)). we collect the L_0 penalty terms from each weight of the diff vector by looping over all modules that contain parameters directly. Then the training loss will be backpropagated to α and finetuning weight vectors. As shown in Figure 5, the rose vector will represent the learnable parameters that training loss can influence.

However, the diff pruning can not control the sparsity[6] for finetuning weight vector. Therefore, we will move forward to the second stage (the Magnitude Pruning). In this stage, since the diff mask is determined, we would define an exact sparsity percentage (we set 10% in the experiment), and then the finetuning weight vector will keep only the top 10% values in terms of their magnitude.

We would iterate all the layers in the model and sort the values of parameters in each layer. Then the threshold value is chosen based on the sparsity proportion, shown by the gray mask in Figure 5. And we set the values smaller than the threshold in the diff vector zero. Therefore, this magnitude algorithm can obtain the diff vector with a certain sparsity.

3.6 Full finetuning: a transfer learning method by transfer training the entire model

3.6.1 Introduction:

As a method of transfer learning, full finetuning has been widely used in NLP and text-to-speech. Therefore, in this thesis, we use full finetuning as the evaluation standard for other transfer learning algorithms and compare the performance of these models.

3.6.2 Full finetuning:

We pre-train the Fastspeech2 model using the LibriTTS dataset before using the full finetuning algorithm. Then we can complete the downstream task by using the checkpoint of the pre-trained model and training with the new migration dataset without changing any structure in the original Fastspeech2 model.

4 Experimental setup

4.1 Training datasets

4.1.1 LibriTTS

we would use *all_clean* subsets of LibriTTS[29] for the pre-training model. LibriTTS is a multilingual English language dataset that reads English speech for about 585 hours at a sampling rate of 24kHz, created by Heiga Zen (with assistance from the Google Voice and Google Brain teams). LibriTTS is designed for TTS research. It is derived from the original material of the LibriSpeech corpus[33](MP3 audio files from LibriVox[34] and text files from Project Gutenberg). The difference between LibriTTS[29] and LibriSpeech[33] are shown following:
1. The audio files are sampled by 24kHz.
2. The speech has obvious sentence breaks.
3. The original and normalized texts are collected in LibriTTS.
4. The contextual information can be extracted.
5. The speech audios with significant background noise are deleted.
For finetuning the pre-trained model, we intend to use LJspeech[29] and the single-speaker subsets of vctk.

4.1.2 LJspeech

LJspeech[9] is a public domain speech dataset consisting of 13100 short audio clips of a female speaker. The reading passages are collected from seven non-fiction books. The clips range from 1 to 10 seconds, with a total length of about 24 hours.
The texts were published between 1884 and 1964 and are completely published. The audio was recorded from 2016 to 2017 by the LibriVox project[34].

4.1.3 vctk

The vctk corpus includes speech data from 110 English speakers with different accents, such as Irish, English, Indian, and so on. Each speaker read 400 sentences

from a newspaper, a rainbow article, and an inspired paragraph for use in the voice stress archive. All voice data is recorded using the same recording setup: an omni-directional microphone(DPA 4035) and a small diaphragm capacitive microphone with a very wide bandwidth(Sennheiser MKH800), with a sampling frequency of 96kHz, 24 bits, located in a semi-anechoic chamber at the University of Edinburgh. Then all records were converted to 16 bits and downsampled to 48kHz. The corpus was originally used for text-to-speech synthesis systems based on HMM. Especially, speech synthesis is based on speaker adaptive HMM, which uses average speech models of multiple speaker adaptive techniques. The corpus is also suitable for DNN-based multilingual human language synthesis systems and waveform modeling.

4.2 Experiment design

we intend to evaluate the model's performance of Adapter[3], BitFit[4], Diff pruning, and full finetuning in the finetuning tasks, especially finetuned with a small dataset. Therefore we use an all-clean subset of LibriTTS[29] for embedding training and pre-training in Fastspeech2 [7]. LibriTTS contains around 585 hours of utterances from multi-speakers. To evaluate the performance of the mentioned models, vctk[35], another multi-speaker TTS corpus with different acoustic information from LibriTTS, is used to finetune the pre-trained model. In the experiment, two males and one female are randomly selected from vctk[35] to work as the target speaker for the finetuning adaptation. We randomly choose 50, 100, 150, and 200 sentences for each speaker to construct small finetuning datasets. Another six extra sentences from each speaker are randomly selected as a text set. As a result, we construct the finetuning datasets with 50, 100, 150, and 200 training samples for each speaker and six testing samples for each simple subset.

Besides, the subset of vctk [35]only has hundreds of training samples. The small finetuning dataset may not be sufficient to extract the tone, intonation, and accent information. In addition, we conduct an experiment with the LJspeech dataset[9] for finetuning.

The LJspeech dataset[9] includes 13100 short audio clips from a female speaker. Like

the previous experiment, we also use an all-clean subset of LibriTTS[29] for embedding training and pre-training. And the adapter, BitFit[4], diff-pruning, and full finetuning algorithms are implemented to finetune the pre-trained model. LJspeech[9] is used as the downstream training dataset. We randomly selected six sentences as text sets to evaluate the performance of the four algorithms we mentioned before. And We generated the target audio using the adapter, BitFit, diff-pruning, and full finetuning models. Then those output audios will be compared to the human target audios in the text sets.

4.3 Model Configuration

In the previous subsections, we introduced datasets for model training and the experiment design to evaluate the model performance. In the following, we intend to introduce the models' detailed settings.

4.3.1 Fastspeech2 model

Our Fastspeech2 model[7] consists of 6 feed-forward Transformer[18] blocks inside the encoder and the mel-spectrogram decoder. In each feedforward block, the dimension of the phoneme embeddings[8] and self-attention size[14] are set as 256. The number of attention heads is set to 2, and the kernel sizes of the 1D convolution, constructed after the self-attention layer[14], are set to 1. The input/output of the first layer in the attention head is 384/384, and the input/output of the second layer in the attention head is set as 384/384. The size of phoneme vocabulary is 76, including the punctuations. In the variance predictor, the kernel sizes of the 1D convolution are initialized to 5 for the pitch predictor. The kernel sizes are set to 3 for the duration and energy predictor, with input/output sizes of 256/256 for both convolution layers. The dropout rate is set to 0.5 for the variance and energy predictors. The dropout rate is set to 0.2 for the duration predictor. Our waveform decoder consists of a 1-layer transposed 1D-convolution[24], whose filter size is 64

Hyperparameter	Fastspeech2
Phoneme Embedding Dimension	384
Pre-net Layers	/
Pre-net Hidden	/
Encoder Layers	6
Encoder Hidden	384
Encoder Conv1D Kernel	1
Encoder Conv1D Filter Size	1536
Encoder Attention Heads	1
Mel-Spectrogram Decoder Layers	6
Mel-Spectrogram Decoder Hidden	384
Mel-Spectrogram Decoder Conv1D Kernel	1
Mel-Spectrogram Decoder Conv1D Filter Size	1536
Mel-Spectrogram Decoder Attention Headers	1
Encoder/Decoder Dropout	0.2
Duration Predictor Conv1D Kernel	3
Duration Predictor Conv1D Filter Size	256
Duration Predictor Dropout	0.2
Pitch&Energy Predictor Conv1D Kernel	5
Pitch&Energy Predictor Conv1D Filter Size	256
Pitch&Energy Predictor Dropout	0.5
Waveform Decoder Convolution Blocks	42
Waveform Decoder Dilated Conv1D Kernel size	1
Waveform Decoder Transposed Conv1D Filter Size	1536
Batch Size	8
Total Number of Parameters	47M

Table 1. Hyperparameters of Transformer TTS and Fastspeech2

and dilated residual convolution blocks are 30. And the skip channel and kernel sizes of 1D convolution are set to 64 and 3. The hyperparameters and configurations of the Fastspeech2 model is listed in Tabel 1.
We implement the pre-training with the Fastspeech2 model[7] on GeForce GTX TITAN X[36], with a batch size of 8 sentences. And the LibriTTS[29] all clean dataset is selected for pre-training. We set 10 warm-up steps to control the rate of gradient descent. The Adam optimizer is chosen to optimize the model with a learning rate of 0.001. And we set 2 training phases. There are 150000 steps in the first phase for the Fastspeech2 training and 50000 steps in the second phase for the style embedding[37] and Fastspeech2 training. In the inference process, the output mel-spectrograms of Fastspeech2 are transformed to audio samples using the HiFiGAN Generator.[38]

4.3.2 Tansfer Learning with Adapter algorithm

We add an extra adapter structure[3] in the Fastspeech2 model to implement downstream tasks. The adapter layer is added between the encoder and variance adapter layers. There are only two linear layers in the adapter structure, which occupy 0.104% parameters of the Fastspeech2 model. The model configuration of the pre-trained Fastspeech2 model has been introduced in the last section.
In addition, we would like to introduce the experiment settings for adapter finetuning. We also divide the training into two phases. The first phase includes 10000 training steps for the adapter Fastspeech2 training, and the second phase also has 10000 training steps for style embedding[37] and adapter Fastspeech2 training. The batch size for the entire experiment is set as 8. The checkpoint we use in the adapter finetuning is calculated by the average of 3 latest checkpoints in the pre-trained Fastspeech2 model. Besides, we prepare ten warm-up steps to prevent overfitting problems at the beginning of the finetuning. We optimize the model using the Adam optimizer, whose learning rate is increased linearly over the first ten steps and then reaches 0.001. All runs are trained on a GeForce GTX TITAN X[36]. The hyperparameters are collected in Table 2.

Hyperparameter	Adapter	BitFit	Diff pruning	full finetuning
Encoder Layers	6	6	6	6
Mel-Spectrogram Decoder Layers	6	6	6	6
Batch Size	8	8	8	8
First Training steps	10000	10000	10000	10000
Second Training steps	10000	10000	10000	10000
optimizer	Adam	Adam	Adam	Adam
training loss	l1 loss+variation predictor loss	l1 loss+variation predictor loss	l1 loss+variation predictor loss+L_0 penalty	l1 loss+variation predictor loss
with Adapter	Yes	/	/	/
parameter variation percentage	0.104%	0.27%	10%	100%
parameter variation in each layer	weight and bias	bias	weight and bias	weight and bias
with L_0-norm penalty	/	/	Yes	/
with additional trainable parameters	Yes, adapter layer	No	Yes, alpha matrix	No
Total Number of Parameters in model	47M+49280	47M	47M	47M
Total trainable Parameters in model	49280	128007	94M	47M

Table 2. Comparison among all four algorithms

4.3.3 Tansfer Learning with BitFit algorithm

In the previous section, we introduced the detail of the BitFit algorithm[4] . The bias terms are additive and correspond to a small subset of the Fastspeech2 model[7], whose bias parameters make up 0.27% of the total number of parameters.
We evaluate the BitFit[4] on Fastspeech2[7], which is consistent with previous work. We also divide the training into two phases and set the same training steps, optimizer, and training loss as the Adapter Fastspeech2. The learning rate and warm-up steps are identical to adapter Fastspeech2. In conclusion, The parameter setting will make the final results comparable to other fine-tuning methods.
The main variation of the BitFit Fastspeech2 model is that we freeze the entire weight matrix of all layers and make the bias vector trainable for linear, self-attention[14], normalization, feed-forward macaron, and convolution layers in Fastspeech2.

4.3.4 Tansfer Learning with Diff pruning algorithm

we intend to compare the diff pruning algorithm [5]against the following baselines: Full fine-tuning, which fully finetunes Fastspeech2, adapter fine-tuning, which trains

tasks-specific bottleneck layer between the encoder and variance adapter layer in adapter Fastspeech2, and BitFit fine-tuning,[4] which freeze all the weight matrix of the Fastspeech2 and make bias vector trainable.

Diff pruning introduces additional hyperparameters l, r(these parameters are used to stretch the Hard-Concrete distribution) and λ(the relaxing factor for weighting the approximate L_0-norm penalty). We set $l = -1.5, r = 1.5, \lambda = 1.25 \times 10^{-10}$ to work well across all tasks. We also initialize the diff vector δ_τ to 0, and α_τ to a positive vector (we use 5) to encourage z_τ to be 1 before training. Besides, we divide the training into two phases. The diff pruning Fastspeech2 model is finetuned during the first phase by training a small dataset. However, in the second phase, the magnitude pruning will replace the diff pruning algorithm to control the sparsity[6] of the diff vector. In the meanwhile, the style embedding is training in the second phase.

We initially train 10000 steps in the first phase for all training tasks for diff pruning. And we set the batch size to 8 and the learning rate of training parameters in Fastspeech2 1×10^{-5} and the learning rate of trainable hyperparameter α_τ to 0.1. Finetuning with the fixed mask after projecting into the L_0-ball with magnitude pruning is training for 10000 steps with a learning rate 1×10^{-5} for all the datasets. And we also set the batch size to 8 in the second phase.

The training of diff pruning is based on the Fastspeech2 model. Therefore the training loss is expressed as:

$$L_{diff_pruning} = L_1 + L_{pitch} + L_{duration} + L_{energy} + \lambda L_{0penalty} \tag{22}$$

where L_1 is calculated by the full finetuning in Fastspeech2[7], and the L_{pitch}, $L_{duration}$, L_{energy} are contributed by variance predictor during training. And $L_{0penalty}$ is the penalty term of the diff vector, which is calculated by formula(17).

In addition, we demonstrate the difference in structure among all three algorithms in Table 2.

4.3.5 Tansfer Learning with full finetuning algorithm

As the evaluation comparison group, full finetuning is a relatively traditional transfer learning algorithm. In this experiment, our experimental parameter settings are the same as those of adapter, BitFit, and diff pruning. We use the pre-training checkpoints and retrain the Fastspeech2 model without any changes. And we set both the first training steps and the second training steps as 10000. Meanwhile, we choose the same optimizer, Adam, as other algorithms. We keep 100% of the training parameters of the Fastspeech2 model as trainable parameters. The detailed experiment setting can be found in Table 2.

5 Results and Evaluation

5.1 Training Results

5.1.1 Embedding training and pre-training

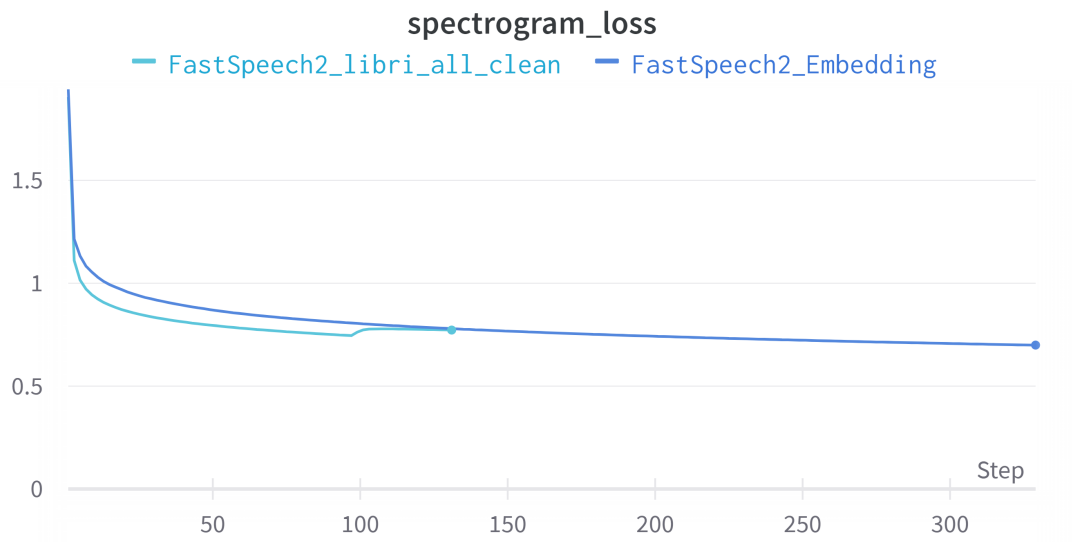

Figure 6. Spectrogtram loss of embedding training and pre-training

In the experiment, we train the embedding as a prior task to extract speaker and style information. Then it is used in a frozen state in Fastspeech2[7]. The embedding is trained with the LibriTTS dataset[29], and the training last 419737 seconds. As we can see in Figure 6, the training loss of embedding finally converges to 0.7.
Then we train the Fastspeech2 model with LibriTTS all clean datasets, a multi-speaker dataset, as the pre-trained model for further finetuning experiments. The training lasts 166012 seconds, and finally, the training spectrogram loss converges to 0.7724. After training, we generate the result example audio. The trained voice sounds are relatively neutral because the data set consists of male and female voices. And then, we intend to enable the pre-trained model to learn a new speaker's pronunciation patterns by finetuning.

5.1.2 Adapter, BitFit, diff pruning, and fully finetuning spectrogram loss comparison

To evaluate the performance of mentioned four finetuning algorithms trained with a small single-speaker dataset, we plan to implement the transfer learning[2] in the pre-trained Fastspeech2 model. We choose speaker p252 and randomly selected 200 training data to construct the finetuning dataset. The spectrogram loss of the entire training process is recorded in Figure 7. The spectrogram losses of the four algorithms start from 2.45-2.48 and gradually decrease as the training progresses. Eventually, the spectrogram losses of adapter, BitFit[4], diff pruning, and fully finetuning converge to 1.929, 2.194, 2.092, and 1.826, respectively. The idea of this experiment comes from [13].

In addition, we implement fine-tuning of the four algorithms trained with the vctk[35] p254 dataset, which has 150, 100, and 50 samples. The spectrogram loss is demonstrated in Figure 8-10. We can see that, with the reduction of the training datasets, the fluctuation in the decline of the spectrogram loss gradually becomes larger.

Besides, we also choose two other speakers, p230 and p254 from vctk[35] to evaluate the model performance. For each speaker, we also randomly select 200, 150, 100, and 50 data samples to construct small datasets to fine-tune the multi-speaker pre-trained model. All the spectrogram loss of the training process is attached in Appendix 1.

Except for using the low-resources datasets(vctk), we also prepared the four algorithms trained with high resources datasets(LJspeech) [9]in Figure 11. LJspeech dataset includes 13,000 samples from a female speaker. For comparison, the model trained with more resources will have a more fluent gradient descent and lower training loss.

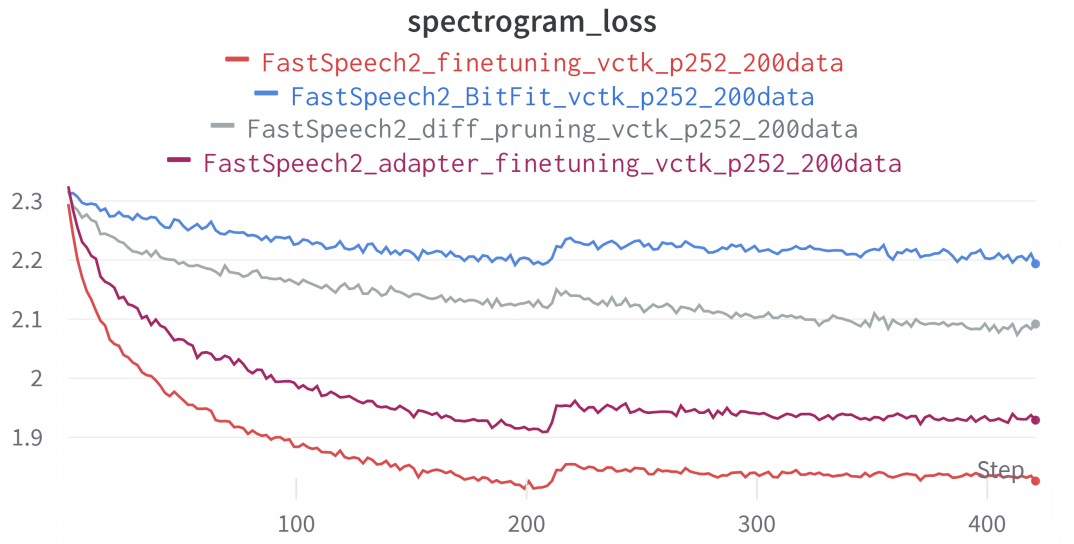

Figure 7. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p252 200data

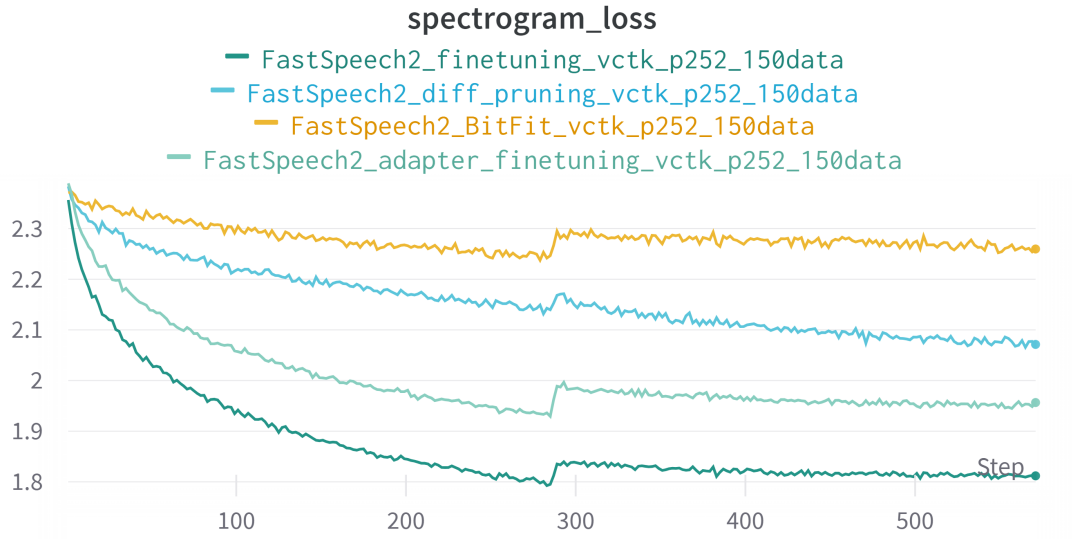

Figure 8. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p252 150data

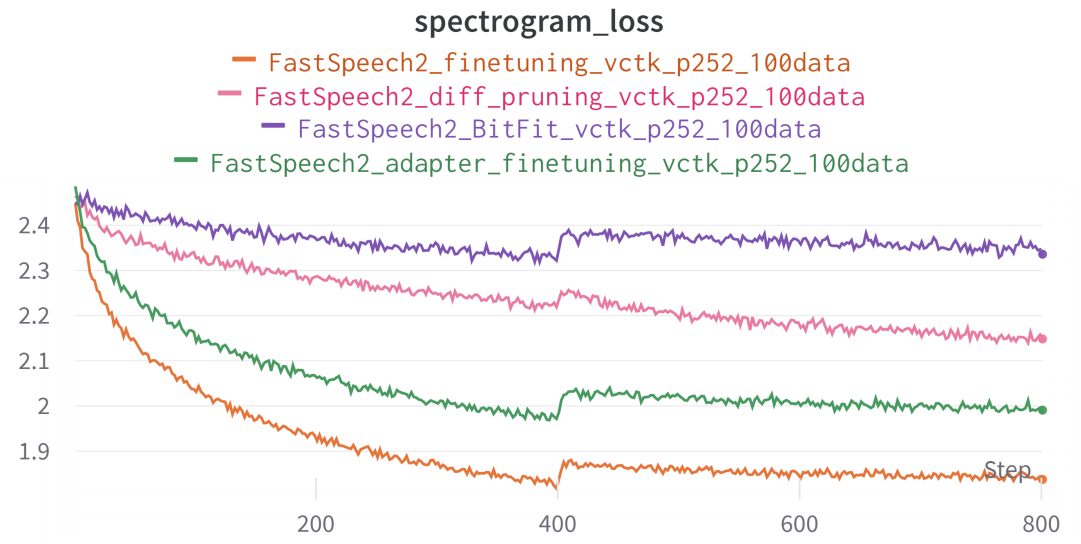

Figure 9. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p252 100data

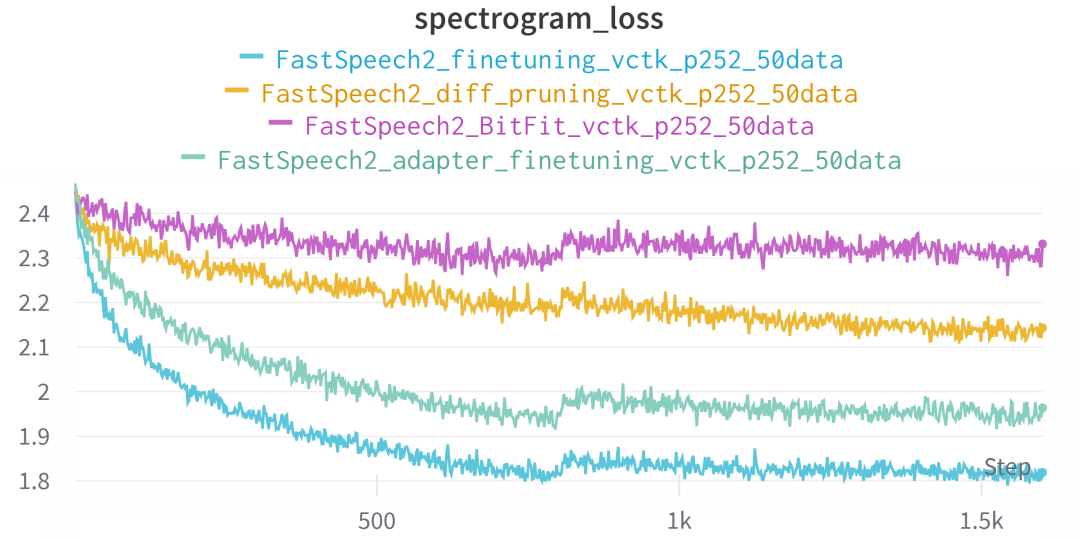

Figure 10. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p252 50data

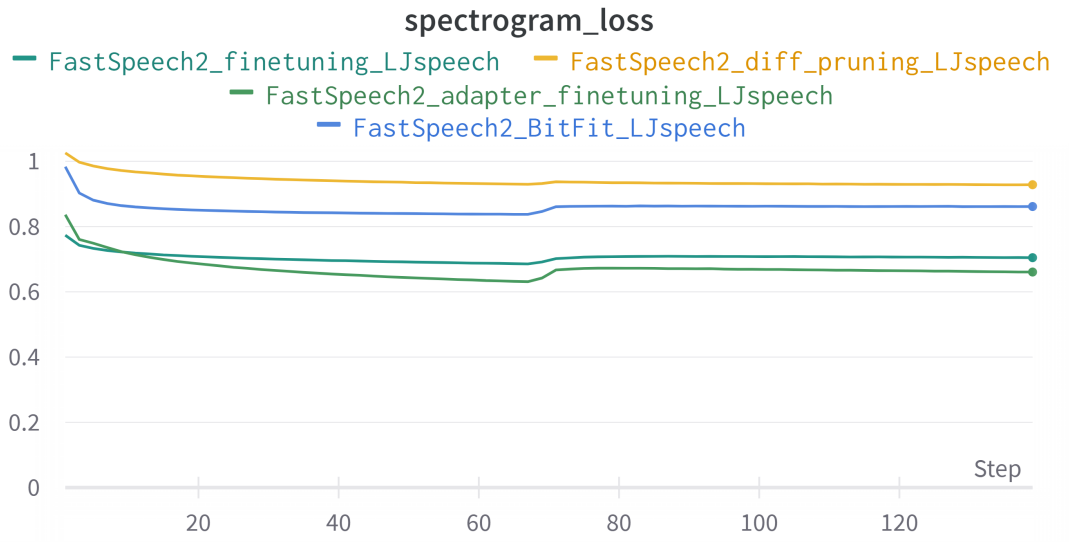

Figure 11. spectrogram loss of adapter, BitFit, diffpruning and fully finetuning trained with LJspeech dataset

5.1.3 Training time comparison

Since we have shown the trainable parameters in Table 2, we can conclude that adapter and BitFit[4] are the most parameter-efficient algorithms compared with others. During training, we found that, with the same training device, BitFit[4] uses the minimum training time to finetune the model. The adapter uses more time than Bitfit[4], but it only spends 0.74% of the full finetuning training time. Based on the algorithm of diff pruning, we can see that diff pruning should also do the full finetuning in the first step(in section 3.4). Therefore, the algorithm should spend more training time when doing the finetuning. The experiment verifies our hypothesis. In the experiment of finetuning trained with vctk p252 200 datasets, adapter finetuning finished the training task using 2291 seconds. BitFit spent only 1464 seconds finishing the task. In comparison, Diff pruning used 3410 seconds, and full finetuning cost 3108 seconds. Besides, the other datasets show the same result. BitFit and adapter are the most parameter-efficient and time-resolved algorithms in

transfer learning[2].

5.1.4 Generate the comparison audios

For each speaker in vctk, we randomly select 6 sentences, resulting in a testing set with a total of 72 comparison audios from 3 speakers, from whom there are four small sample groups. All audio is generaated to 48kHz. And we will compare the generated audio and the test audio from a real speaker. Unfortunately, the audios generated by diff pruning finetuned with vctk[35] small datasets(all 50, 100, 150, 200 data samples) are pure noise. In contrast, the adapter BitFit and finetuning algorithms have detectable results for further evaluation. And we also generate audios trained with LJspeech single speaker dataset [9]for the high resources comparison group. There are 6 test sentences selected from LJspeech [9]to generate the target audios. All audios are downsampled to 16kHz since the comparison test audios from LJspeech are 16kHz as well. By using enough data samples for training, the diff pruning algorithm can generate comparable samples to other proposed methods.

5.1.5 Cosine Similarity comparison

Cosine Similarity To evaluate the performance of the finetuning models, we will measure the cosine similarity[39] between the output audios generated by finetuned models and the target human audio samples. A high cosine similarity[39] means that the voice of the output audio sounds similar to the target training speaker's voice, indicating a good imitation of speakers. [40]
We use the function from Pytorch[41] to calculate the cosine similarity[39], expressed by $torch.nn.functional.cosine\_similarity(x1, x2, dim = 0)$ This function calculates the cosine similarity[39] between $x1$ and $x2$. $x1$ and $x2$ should be broadcastable to a common shape. dim represents the dimension in this common shape. Then the dim of the output will be squeezed, resulting in the tensor having one fewer dimension. The Cosine similarity is calculated with the following formula:

$$cosinesimilarity = \frac{x_1 \cdot x_2}{max(||x_1||_2 \cdot ||x_2||_2)} \tag{23}$$

$|| \cdot ||_2$ represents the Euclidean norm.[42]

We calculate all the cosine similarities[39] for the output audios trained by vctk, shown in Table 3. Since the diff pruning algorithm can not generate effective audio trained by low resource dataset, table 3 will not include cosine similarity for diff pruning. In comparison, we list the results for the adapter, BitFit, diff pruning, and full finetuning trained by high resource dataset (LJspeech) [9]in Table 4.

we can conclude that trained with low resources, BitFit, and adapter perform better than full finetuning compared with cosine similarity in most cases. But trained with high resources, the adapter algorithm has more advantages in cosine similarity.

We noticed that when training with low-resource datasets, the duration information will be influenced due to finetuning of the model. When training with full finetuning, the duration predictor parameters obtained in pre-training will be somewhat affected. Therefore, when generating audio signals, the resulting audios will be accelerated, affecting the audio quality. In contrast, the adapter algorithm freezes the variance predictor layer and adds new linear layers, which will be fully trained in the finetuning process. Therefore, the adapter can restore the speaker's speech information with low resources, and the speech duration, pitch, and energy are also relatively consistent. BitFit has the same idea by freezing the parameters of the weight matrix in each layer of Fastspeech2, which also includes variance predictor, and training the model by finetuning the bias vector. Most parameters in variance predictor layers obtained during pre-training are protected during finetuning, but a few parts are allowed to be modified. This method can encourage the generation of audio signals similar to that of the trained speaker.

The effect of the diff pruning algorithm in this experiment is not very satisfactory when training with small datasets. The generated voice signals are all noise signals. Meanwhile, the variance predictor layers cannot output effective duration information due to overfitting in the training process. However, Training results for diff pruning on the LJspeech dataset [9]are comparable to a certain extent because LJspeech is a high resources dataset. In the process of transfer learning[2] for diff

Cosine similarity(std.)	BitFit	full finetuning	adapter
vctk p230 50data	0.9374(±0.0336)	0.9334(±0.0268)	0.9470(±0.0224)
vctk p230 100data	0.9281(±0.0394)	0.9402(±0.0237)	0.9262(±0.0289)
vctk p230 150data	0.9414(±0.0193)	0.9389(±0.0313)	0.9396(±0.0313)
vctk p230 200data	0.9104(±0.0485)	0.9132(±0.0319)	0.9061(±0.0469)
vctk p252 50data	0.8668(±0.0549)	0.7831(±0.0581)	0.8001(±0.0547)
vctk p252 100data	0.8307(±0.0451)	0.7925(±0.0617)	0.8077(±0.0675)
vctk p252 150data	0.8955(±0.0275)	0.8529(±0.0564)	0.8632(±0.0365)
vctk p252 200data	0.8614(±0.0405)	0.7966(±0.0533)	0.7872(±0.0467)
vctk p254 50data	0.8247(±0.0359)	0.8046(±0.0472)	0.8090(±0.0394)
vctk p254 100data	0.8681(±0.0274)	0.7980(±0.0662)	0.8740(±0.0680)
vctk p254 150data	0.8765(±0.0217)	0.8219(±0.0463)	0.8511(±0.0469)
vctk p254 200data	0.8612(±0.0491)	0.7633(±0.0711)	0.8341(±0.0235)

Table 3. Cosine Similarity of BitFit, full fine-tuning and adapter trained by vctk

Cosine similarity(std.)	Bitfit	diff pruning	full finetuning	adapter
LJspeech	0.9501(±0.0078)	0.9124(±0.0153)	0.9410(±0.0169)	0.9274(±0.0177)

Table 4. Cosine Similarity of BitFit, diff pruning, full fine-tuning and adapter audios trained by LJspeech

pruning, Fastspeech2 models can be fully trained, thus avoiding overfitting to a certain extent. Therefore, under the training of high resources, the effectiveness of the diff pruning algorithm has also been proved. According to the result of cosine similarity[39] in Table 4, the BitFit algorithm has the best similarity trained with high resources, and its evaluation result is better than the full finetuning baseline. Although diff pruning and adapter have lower cosine similarity results than Bitfit, the output audios quality is acceptable.

5.1.6 Naturalness comparison

To measure the naturalness of the generated audio, a mean opinion score(MOS)[43] test is conducted. A good synthesized sample should have high quality in naturalness[44] and similarity with the target speaker. The naturalness evaluation[44] will be introduced in the next subsection. We intend to evaluate the mean opinion score[43] of naturalness[44] for the four mentioned algorithms in this part.

Mean opinion score Mean opinion score(MOS)[43] is an evaluation criterion used in the domain of telecommunications engineering, providing the overall quality of a system. It is the arithmetic average of all individuals' "values within a predefined range that the subject assigns to his perception of the quality performance of the system."[45] Such ratings are usually collected in subjective quality assessment tests but can also be estimated algorithmically.
MOS is a common metric for video audio and audiovisual quality assessment but is not limited to these ways. ITU-T[46] defines several ways to cite MOS in Recommendation ITU-TP.800.1[46], depending on whether scores are obtained from audiovisual, conversational, listening, talk, or video quality tests.
The Rating scales and mathematical definition: Mos is expressed as a single rational number, usually in the range 1-5, where 1 is the lowest perceived quality, and 5 is the highest perceived quality. Other MOS ranges are also acceptable, depending on the rating scale used in the base test. The Absolute Category Rating scale is very commonly used and maps ratings between Bad and Excellent to numbers between 1 and 5, as shown in Table 5.

The MOS is calculated as the arithmetic average of a single rating performed by a human subject on a given stimulus in a subjective quality assessment test. It is calculated by the formula:

$$MOS = \frac{\sum_{n=1}^{N} R_n}{N} \tag{24}$$

Where R are the individual ratings for a given stimulus by N subjects.

Rating	Label
5	Excellent
4	Good
3	Fair
2	Poor
1	Bad

Table 5. Absolute Category Rating scale in MOS

For the property of MOS Studies have shown that for categorical rating scales(such as ACR). Subjects do not perceive individual items equally. For example, the gap between good and fair may be larger than the gap between good and excellent. The perceived distance may also depend on the language into which the scale is translated. However, previous studies have failed to prove that scale translation significantly affects the results obtained.

There are several other biases in the typical way a MOS rating is obtained in addition to the non-linear scale problem described above. There is a "range equilibrium bias": subjects tend to give scores that span the entire rating scale during subjective experiments. Suppose the quality range presented is different. It is impossible to compare too different subjective tests. In other words, MOS is never an absolute quality measure but only relative to the tests that obtained it. For the above reasons, as well as several other environmental factors that affect the perceived quality of subjective tests: MOS values should only be reported if the environment in which they were collected is known and reported. Therefore, MOS values collected from different environments and test designs should not be directly compared.

Specifically, the guide provides the following suggestion: It is meaningless to directly compare the MOS values produced by individual experiments unless the experiments are explicitly designed for comparison, and even the data should be statistically analyzed to ensure that the comparison is valid.

Rating	Label	description
5	Excellent	the speech is broadcast-grade and very clear and fluent,
4	Good	There is no abnormal rhythm, relatively clear and fluent,
3	Fair	There are no serious prosodic errors, and very few syllables that were not quite clear
2	Poor	Occasionally, a few syllables are not quite clear, and there are some unusual prosodic patterns
1	Bad	It's not clear. It sounds like a simple concatenation of separate syllables

Table 6. The evaluaiton criterion for naturalness MOS

Specific experiment design for naturalness MOS calculation We made a questionnaire for the naturalness[44] MOS evaluation. Considering the time cost for each testee to finish the questionnaire, we control the questionnaire completion time within 30 minutes so that the test subjects can complete the questionnaire within an acceptable time range. Therefore, we selected the speech generated by the model trained by 50 data samples and 200 data samples of three speakers, p230, p252, and p254, from the vctk dataset for evaluation. Meanwhile, we also extracted 4 speech signals of each speaker as comparison samples and put them into the questionnaire. (Appendix C) Besides, we indicated the speech quality corresponding to each score in the MOS test in the questionnaire so that the test subjects could refer to it. The evaluation criterion for naturalness[44] MOS is recorded in Table 6.

MOS of Naturalness(std.)	BitFit	adapter	full finetuning	original speaker
vctk p230 50data	3.8553(±0.8595)	3.3616(±0.8139)	3.1316(±0.9885)	4.3289(±1.1122)
vctk p230 200data	3.8289(±0.8389)	3.6316(±0.9070)	3.4211(±0.9968)	4.2895(±1.1292)
vctk p252 50data	3.7500(±0.8347)	3.3421(±0.8877)	2.6711(±0.8228)	4.4079(±0.9957)
vctk p252 200data	3.9211(±0.9057)	3.4737(±0.7741)	3.0526(±0.9222)	4.4342(±1.0111)
vctk p254 50data	3.2247(±0.8422)	2.8526(±0.8749)	2.2500(±0.8660)	4.5132(±0.9591)
vctk p254 200data	3.3026(±0.7307)	2.5658(±0.7718)	1.9342(±0.8693)	4.4737(±0.9997)

Table 7. Naturalness of BitFit, adapter, full fine-tuning, and original speaker audios under low resource vctk dataset

MOS of Naturalness(std.)	BitFit	adapter	diff pruning	full finetuning	original speaker
LJspeech	3.3553(±0.8595)	3.5000(±0.9730)	3.4079(±0.9406)	3.5395(±0.8861)	4.4342(±1.1116)

Table 8. Naturalness of BitFit, adapter, full fine-tuning and original speaker audios under high resource LJspeech dataset

MOS results of naturnalness We found 19 subjects to complete the naturalness questionnaire. After that, we collected the test data and conducted statistical analysis to calculate the mean and standard deviation of each finetune model test. The results are shown in Table 7.

It can be seen that among the audio signals trained using the low resources dataset of the same speaker with vctk. BitFit and adapter methods obtain the highest MOS score, which means that the naturalness of the speech generated by BitFit and adapter under the small dataset is close to that of the original speaker. In addition, we can see that the audio naturalness generated by full finetuning trained with small datasets of different speakers is the lowest among the three methods(as a standard), which also confirms the effectiveness of the Bitfit and adapter model we use.

Besides, we also trained LJspeech(high resources dataset) to fine-tune the pre-trained model to compare adapter, BitFit, diff pruning, and full finetuning methods' naturalness of output audios. Since diff pruning can generate audio results in high resources training. We add the MOS evaluation of diff pruning to Table 8.

Judging from the results of the evaluation. Audios obtained by the adapter and fine-

tuning training model have the best naturalness. And their naturalness evaluations are close, which is better than the audios generated by BitFit and diff pruning.

5.1.7 Similarity comparison

In the same way as the measurement method of naturalness[44] MOS in the previous part, we also used 50 and 200 data samples selected by p230, p252, and p254 speakers in vctk to finetune the Fastspeech2 model using the four mentioned methods. Since diff pruning does not generate the hearable audios, this algorithm is excluded in the low resources similarity evaluation. A similarity comparison was made between these model-generated audio signals and 6 test samples randomly selected from each speaker data set. We also used 1-5 as the scale for the test subjects to choose from. In addition, we made a similarity evaluation for the algorithms implemented in LJspeech[9](diff pruning also included). Like the evaluation in vctk, we also choose 6 test audio samples to make the evaluation. The target audios are compared to the speeches generated by adapter, BitFit, diff pruning, and full finetuning.
Similarity has also been added to the test questionnaire. We can also provide testees with 1-5 options. This criterion is shown in Table 9.

MOS results of Similarity For similarity evaluation, we found 14 subjects to finish the questionnaire. After collecting all the test results, we conduct a statistical evaluation. The main purpose is to evaluate the model's performance by calculating and comparing the mean and variance of generated speech similarities for each training speaker and transfer learning method. The evaluation results are shown in Tables 10 and 11. It is shown in Table 10 that in the model using vctk single speaker dataset(low resource) for transfer learning, audios from BitFit and adapter have better speech similarity than full finetuning. And in most cases, BitFit has the highest value. As a standard comparison object, full finetuning does not perform well under the training of small datasets.
Relatively speaking, audios from the adapter and full finetuning have close similarity among all the models trained by LJspeech, and the similarity of the two models'

Rating	Label	description
5	Excellent	This is the same speaker in both audios
4	Good	I recognize the voice of the reference speaker, but something is a bit different
3	Fair	I recognize some characteristics of the voice of the reference speaker, but there is a clearly noticeable difference
2	Poor	The voices are somewhat similar, but I think these are different speakers
1	Bad	This are different speakers

Table 9. The evaluaiton criterion for similarity MOS

audio is higher than the audios from BitFit and diff pruning, as shown in Table 11. By analyzing the above similarity results, it can be conducted that the audio generated by BitFit and adapter is closer to the original speech under the training of low resource dataset. Besides, training with high resource datasets, the audios obtained from the adapter and full finetuning have better performance in similarity.

MOS of Similarity(std.)	BitFit	adapter	full finetuning
vctk p230 50data	3.0179(±1.2430)	3.0179(±1.3002)	2.7143(±1.1555)
vctk p230 200data	2.4821(±1.3347)	2.4464(±1.1896)	2.5357(±1.2499)
vctk p252 50data	2.6607(±1.2399)	2.8036(±1.3806)	2.4464(±1.2493)
vctk p252 200data	3.3036(±1.2638)	3.1607(±1.1875)	2.9107(±1.2251)
vctk p254 50data	2.5714(±1.1419)	2.3571(±1.1667)	1.9464(±0.9985)
vctk p254 200data	2.5893(±1.2177)	2.1429(±1.1025)	1.8571(±1.1510)

Table 10. Similarity of BitFit, adapter, full fine-tuning and original speaker audios under low resource vctk dataset

MOS of Similarity(std.)	BitFit	adapter	diff pruning	full finetuning
LJspeech	2.6607(±1.1951)	2.8036(±0.9614)	2.6071(±1.3708)	2.8571(±1.2274)

Table 11. Similarity of BitFit, adapter, full fine-tuning, and original speaker audios under high resource LJspeech dataset

6 Results Analysis

In section 5, we evaluate the training loss, cosine similarity(Non-subjective evaluation), and the naturalness and similarity(subjective evaluation) of the model-generated speech. In this section, we would like to introduce some conclusions and details.

1. The training of Diff pruning on a small dataset fails to generate effective results, which use vctk simple speaker dataset for transfer learning. The generated audios from diff pruning are pure noise, whose duration information extracted from the original text is also destroyed.

2. The performance of Finetuning algorithm is the worst trained with low resources datasets(vctk) among all the three transfer learning methods tested. The audio's duration information is interfered, resulting in the output audio's recitation speed becoming faster.

3. Among the models we trained with low resource dataset(vctk single speaker), BitFit and adapter have the best audio quality. We can conduct the above conclusion from the result of cosine similarity, MOS naturalness, and MOS similarity. Moreover, the audio duration information can be better restored under the training of low resource dataset, and the audio quality is better than our test criteria(full finetuning).

4. We also evaulate the model performance in transfer learning trained with a high-resource dataset(LJspeech). We found that the model using the diff pruning algorithm trained with the LJspeech dataset is surprisingly effective and can generate evaluable audio signals. Therefore we infer that the diff pruning algorithm could meet the overfitting problem trained with low resource dataset.

Of all the models trained using the LJspeech dataset. The audios generated by the models using the adapter and full finetuning methods have better MOS naturalness and similarity than the audios using BitFit and diff pruning.

5. In terms of all the model training processes, BitFit requires the least transfer learning training time when the same dataset is used for training. The adapter's training time is also excellent and obviously less than that of full finetuning, while

diff pruning has the most model training time.
Based on the above conclusions and the previous introduction of all transfer learning algorithms, we can conduct an effective analysis and infer some explanation.

Question1: Why does the diff pruning model trained with a low resource dataset fail to generate evaluable audios(pure noise), and the duration information is damaged?
Analysis: We can learn from the algorithm introduction of diff pruning[5] that when computing diff vector, diff pruning first implements full finetuning. Since all layers of Fastspeech2 can be trained in diff pruning. The parameter from the variance predictor in Fastspeech2 has great variation. However, due to insufficient training data samples from low-resource datasets. Variance predictor's training could meet the overfitting issue during transfer learning, resulting in the distortion of duration information. In addition, diff pruning trims diff vector. The hyperparameter alpha-group determines only some parameters in diff vector trainable by calculating the hard concrete reflection. (which is shown in 3.5.2). Therefore, diff pruning leads to overfitting when the model is trained with a low-resource dataset to a greater extent. In contrast, in training using LJspeech datasets, the overfitting problem of diff pruning can be effectively solved due to sufficient data. Besides, compared with the full finetuning method, diff pruning makes Fastspeech2 have fewer parameter changes in transfer learning.

Question2: Why do adapter and BitFit algorithms produce better audio when trained on small datasets?
Analysis: For the adapter[3], we freeze all the layers in Fastspeech2, including the encoder, variance predictor, and decoder. Besides, we insert the adapter layer in the middle. The information of small datasets can be fully learned in this additional layer, and the bottleneck structure will cause parameter validity. The model does not meet the overfitting issue in the transfer learning process since only a small part

of the parameter is trainable. In addition, because the variance predictor's freezing, duration, pitch, and energy information are not significantly damaged during transfer learning, the adapter performs well on low and high resources datasets.
For the Bitfit algorithm[4], since we freeze the weight matrix(key parameter) of all layers, only the bias vectors of all layers are trainable in the transfer learning process. So, it does not greatly change the pre-trained model when training with small datasets. Only a small part of the parameters are trainable. Therefore, the training with low-resource datasets does not result in the overfitting of the model. However, compared with high resources datasets training, the bias vector is not the key parameter in model training. Hence, the limitation of the trainable parameters results in low learning efficiency in transfer learning. We can see from the results of MOS naturalness and MOS similarity. BitFit migration-trained audio performed worse than the adapter on both assessments.

Question3: Why does BitFit take the least time in terms of training time, the adapter takes the second least time, followed by full finetuning and diff pruning?
Analysis: First, BitFit's trainable parameters are bias vectors, while the adapter's trainable parameters are weight matrix and bias vector(two linear layers). The trainable parameters of these two transfer learning methods are the least. Moreover, when the trainable parameters are in the same amount, the trainable parameters of the BitFit algorithm with bias vector update faster than the adapter. In comparing the training duration between full finetuning and diff pruning, diff pruning must first carry out full finetuning training in each step when calculating the diff vector. Besides, the magnitude pruning in the diff pruning algorithm will sort the parameters of each layer and select the top percentage of parameters to update. These mechanisms increase the training time, and also diff pruning adds the trainable parameter alpha. So, the training time will be significantly more than full finetuning.

7 Conclusions

In order to solve the overfitting problem of small dataset training in Fastspeech2, four transfer learning methods are proposed. We hope to first train the model with multi-speakers datasets to make the Fastspeech2 model learn the speech patterns of different speakers. Then, the transfer dataset is used for transfer learning so that the finetuned model can also imitate the tone, intonation, and other phonetic features of the target speaker. In this paper, we implement the transfer learning methods from NLP into text-to-speech tasks to achieve the mentioned goal. We adopt the algorithms of BitFit, adapter, and diff pruning for transfer learning, and we take full finetuning as the evaluation standard. After training, we also evaluate the generated audios obtained by the above transfer learning methods trained with high resources and low resources datasets in different data amounts.

For comparing the advantages and disadvantages of different models training with high resources and low resources datasets. We make the non-subjective and subjective evaluation for the generated audios from the transfer learning models. The non-subjective tests include evaluating training loss, training time, and cosine similarity. Subject evaluation is also an effective evaluation for text-to-speech. We obtained 19 subjective scoring samples for MOS naturalness evaluation and 14 subjective scoring samples for MOS similarity evaluation. Based on the evaluation results, we can conclude as follows:

1. Training with low resource dataset, the audios generated by the BitFit model have the best cosine similarity, MOS naturalness, and MOS similarity for different speakers, and the training time is also the shortest. Compared with the evaluation result of the full fine-tuning, BitFit has obvious advantages. However, under the training of the high resources dataset. The evaluation of multiple parameters of BitFit is not optimal, such as MOS similarity, MOS naturalness, and cosine similarity.

2. For the adapter algorithm, this model trained with high resources and low resources datasets can both produce good quality audio. Trained with low resources dataset, the MOS naturalness and MOS similarity evaluations of the adapter model are both superior to the full fine-tuning model. Meanwhile, the model trained with

high resources dataset has almost the same evaluation results as full finetuning. Surprisingly, the training time of the adapter is significantly shorter than that of full fine-tuning. So adapter can completely replace the full finetuning for text-to-speech tasks.

3. We have also introduced the diff pruning algorithm, and it has been found that the model trained with low resources datasets cannot generate effective audios, and the duration information of audios has been destroyed in the process of transfer learning. Therefore, we conclude that diff pruning is not suitable for transfer learning with low resources datasets. Surprisingly, we found that the diff pruning algorithm is effective when training high resources datasets. It is found that the audio signal generated by diff pruning has similar results as the BitFit in the evaluation of MOS naturalness and MOS similarity. However, in comparison, the model training time of diff pruning is the longest among all transfer learning methods and even higher than that of full finetuning. Therefore, Diff pruning is not an optimal choice in text-to-speech transfer learning.

4. As the assessment standard we adopt, we found that in the training process of low resources datasets, the duration information of the full finetuning model would be affected, resulting in faster speech speed and thus affecting the quality of speech, Therefore, the full finetuning algorithm is not suitable for text-to-speech training with low resources datasets, since full finetuning could meet the overfitting problem if the training steps are beyond its needs. Relatively speaking, the training results obtained by full finetuning using large datasets are excellent, achieving the best result in the evaluation of MOS naturalness and MOS similarity. But the disadvantage is that the training time is longer than that of the adapter.

Therefore, in this thesis, we can conclude that in the text-to-speech task based on the Fastspeech2 model, the BitFit, and adapter methods are the best choices if there are transfer learning tasks trained with low resources datasets. If the training dataset has high resources, adapter and full finetuning can be the first choice for transfer learning tasks.

8 Future work

Except for transfer learning, there are many other methods for imitating speakers trained with low resources datasets. We have drawn a conclusion that Adapter and BitFit models have better audio quality in transfer learning with small datasets. But it may be necessary to combine various methods to restore the speaking styles of the target speaker. The style embedding method[37] is recommended, which can help to improve the audio quality. In addition, there are many tasks in text-to-speech that transfer learning could be implemented into, such as finetuning models with small datasets to generate emotional audios and check how well the model can reproduce the emotion. In addition, the optimization of the hyperparameters is also a research field worth exploring. For example, the training of diff pruning algorithms is divided into two stages. The first stage is training using the diff pruning algorithm, and the second stage is the magnitude pruning. We recommend that the two training stage can be carried out alternately to see whether the quality of the speech generated by the model can be improved and the hyperparameters such as the percentage of the updated parameters in magnitude pruning can be modified to a lower value.
For the adapter algorithm, the total number of parameters of the adapter layer(bottleneck) could be modified according to the total number of parameters in the original model or according to the total number of samples in the training datasets to prevent the overfitting issue caused by excessive trainable parameters during finetuning.
For BitFit and the full fine-tuning algorithm, we can try to freeze the weight and bias parameters of the variance adapter layer in Fastspeech2 to prevent the duration, Pitch, and energy information in pre-training from being destroyed. See if this change makes for better transfer learning for Fastspeech2 on small datasets.

Appendix A Spectrogram loss during training using vctk P230 and vctk P254, and LJspeech

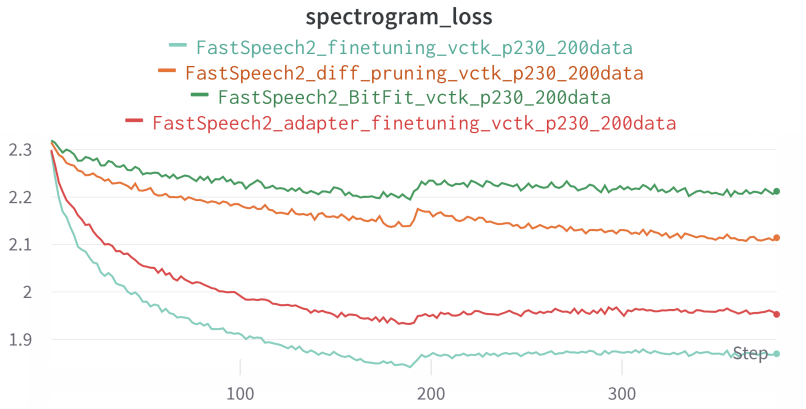

Figure 12. spectrogram loss of adapter, BitFit, diffpruning and fully finetuning trained with vctk p230 200data

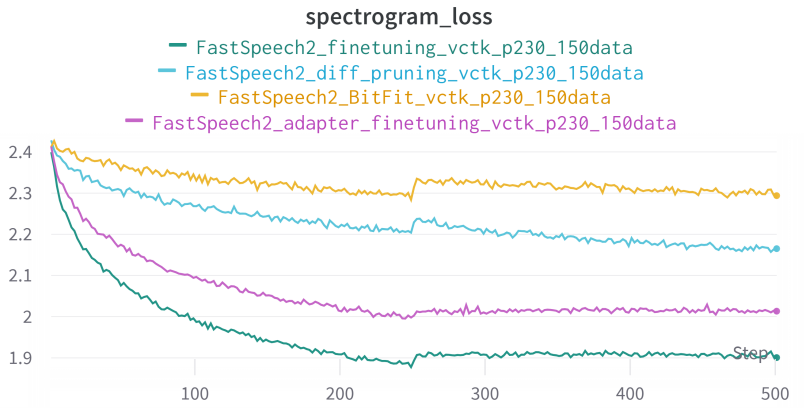

Figure 13. spectrogram loss of adapter, BitFit, diffpruning and fully finetuning trained with vctk p230 150data

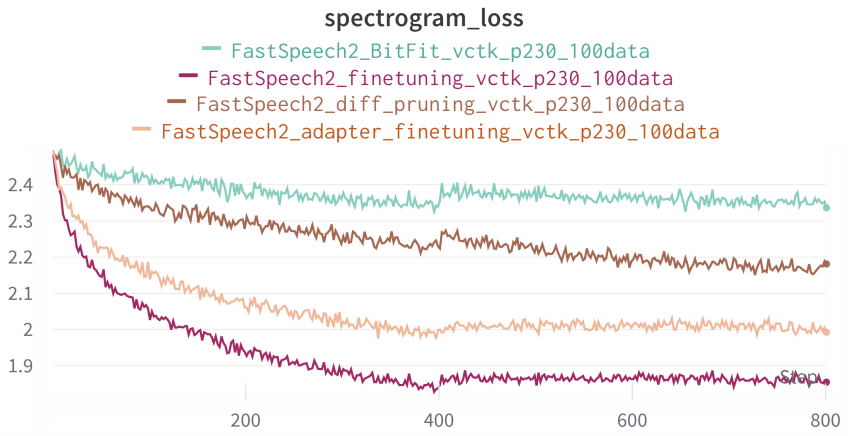

Figure 14. spectrogram loss of adapter, BitFit, diffpruning and fully finetuning trained with vctk p230 100data

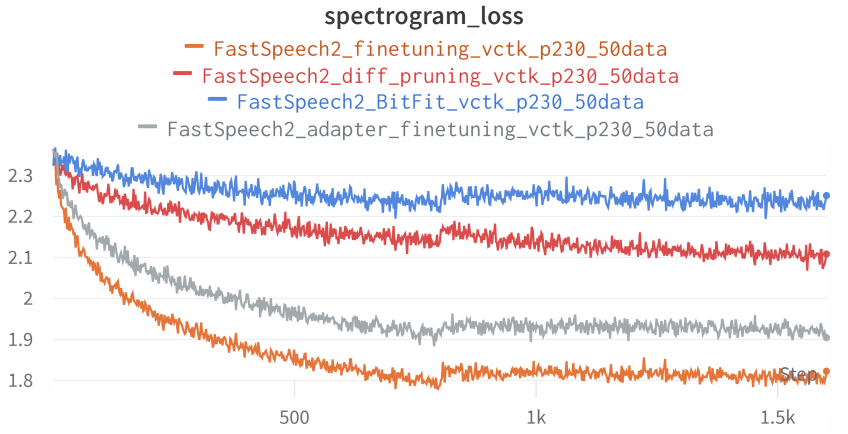

Figure 15. spectrogram loss of adapter, BitFit, diffpruning and fully finetuning trained with vctk p230 50data

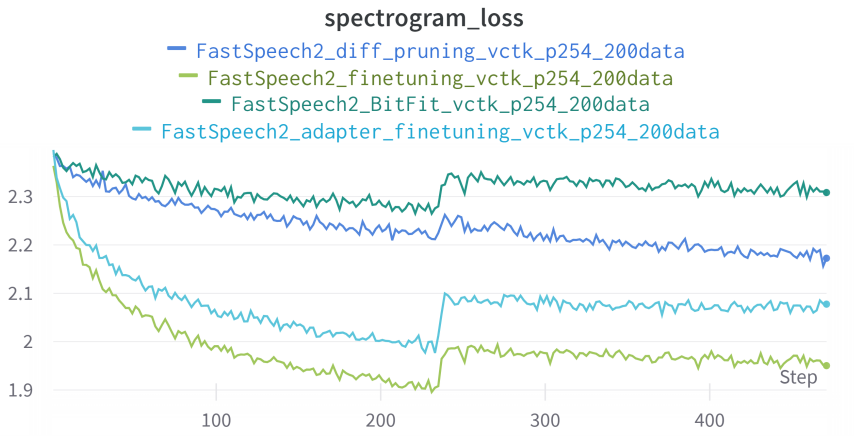

Figure 16. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p254 200data

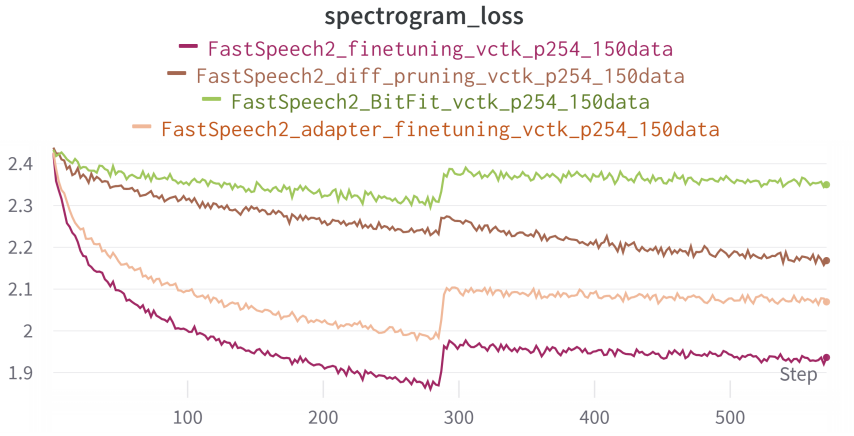

Figure 17. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p254 150data

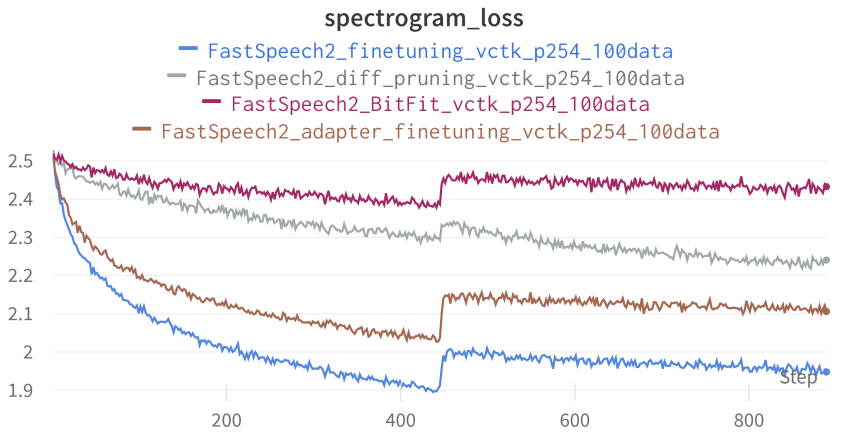

Figure 18. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p254 100data

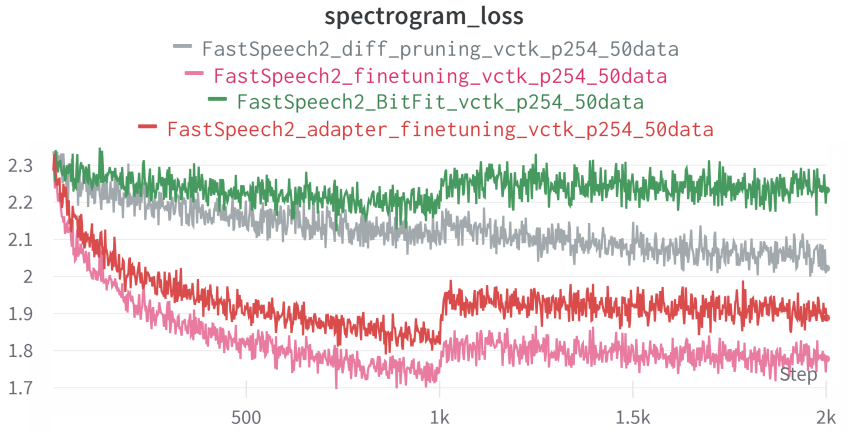

Figure 19. spectrogram loss of adapter, BitFit, diffpruning and fully fine-tuning trained with vctk p254 50data

Appendix B Mel-spectrogram of all the models

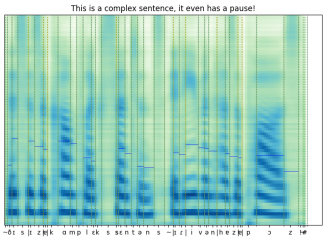

Figure 20. adapter vctk p230 50data

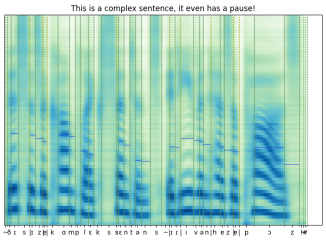

Figure 21. BitFit vctk p230 50data

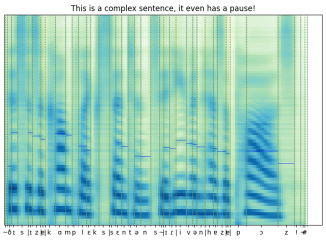

Figure 22. diff pruning vctk p230 50data

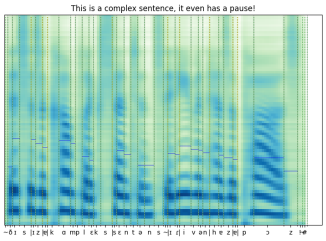

Figure 23. finetuning vctk p230 50data

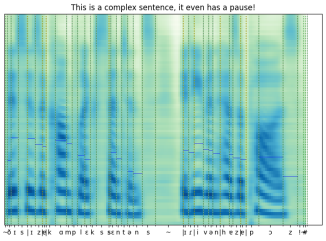

Figure 24. adapter vctk p230 100data

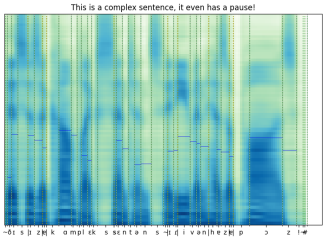

Figure 25. BitFit vctk p230 100data

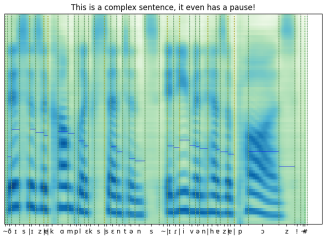

Figure 26. diff pruning vctk p230 100data

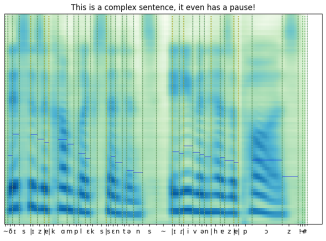

Figure 27. finetuning vctk p230 100data

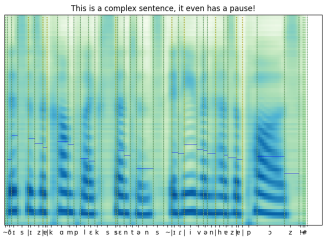

Figure 28. adapter vctk p230 150data

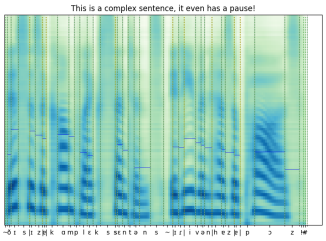

Figure 29. BitFit vctk p230 150data

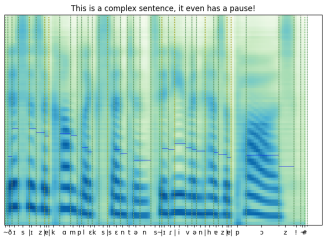

Figure 30. diff pruning vctk p230 150data

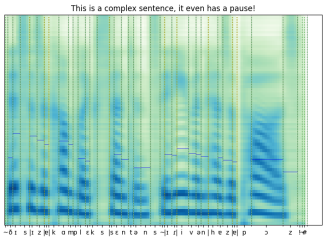

Figure 31. finetuning vctk p230 150data

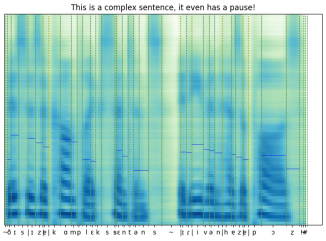

Figure 32. adapter vctk p230 200data

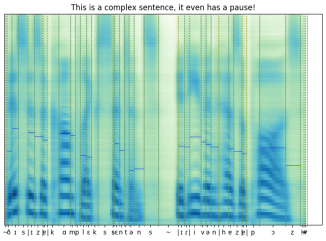

Figure 33. BitFit vctk p230 200data

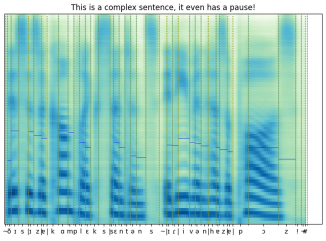

Figure 34. diff pruning vctk p230 200data

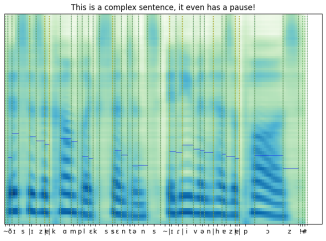

Figure 35. finetuning vctk p230 200data

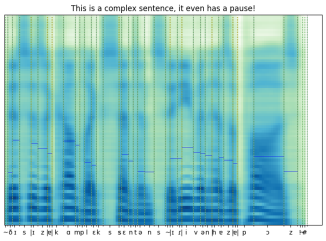

Figure 36. adapter vctk p252 50data

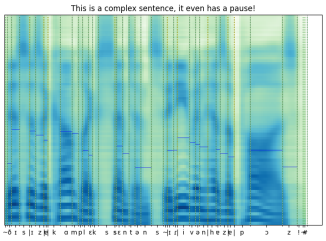

Figure 37. BitFit vctk p252 50data

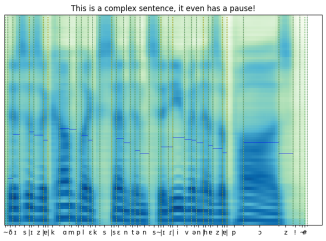

Figure 38. diff pruning vctk p252 50data

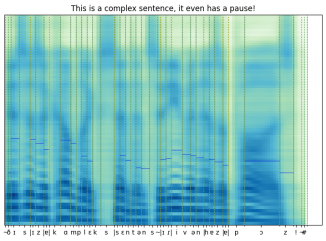

Figure 39. finetuning vctk p252 50data

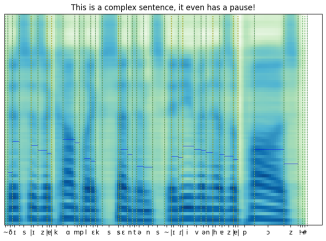

Figure 40. adapter vctk p252 100data

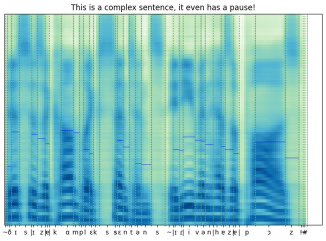

Figure 41. BitFit vctk p252 100data

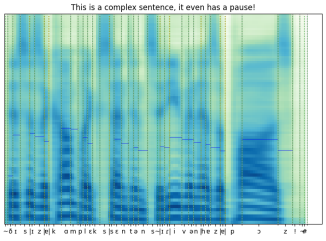

Figure 42. diff pruning vctk p252 100data

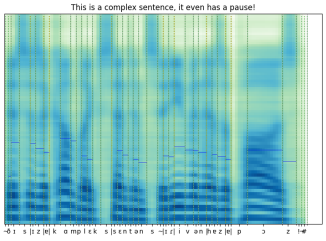

Figure 43. finetuning vctk p252 100data

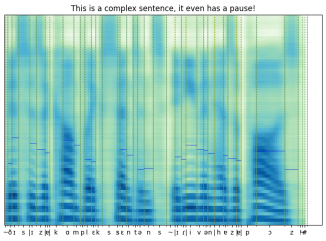

Figure 44. adapter vctk p252 150data

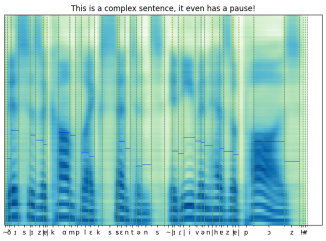

Figure 45. BitFit vctk p252 150data

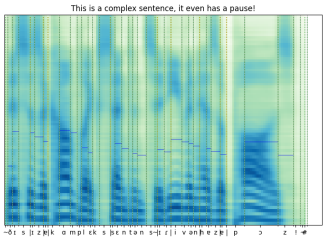

Figure 46. diff pruning vctk p252 150data

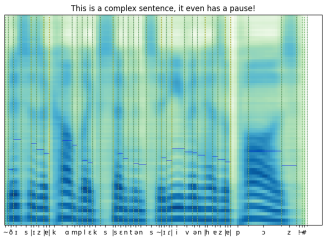

Figure 47. finetuning vctk p252 150data

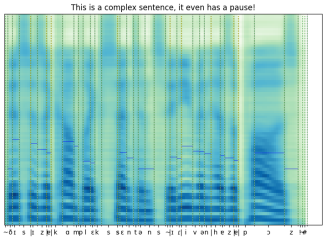

Figure 48. adapter vctk p252 200data

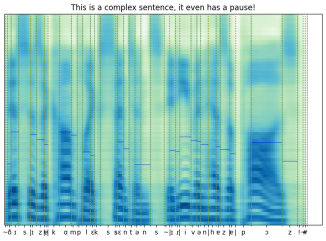

Figure 49. BitFit vctk p252 200data

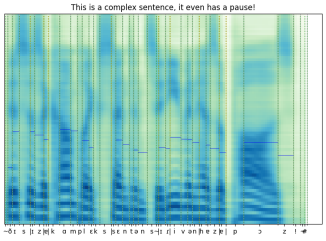

Figure 50. diff pruning vctk p252 200data

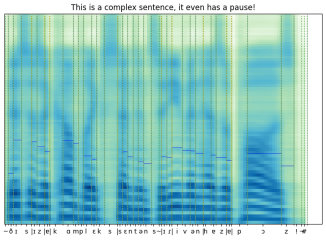

Figure 51. finetuning vctk p252 200data

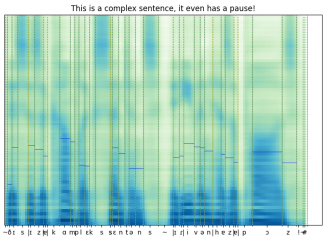

Figure 52. adapter vctk p254 50data

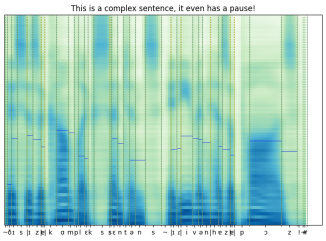

Figure 53. BitFit vctk p254 50data

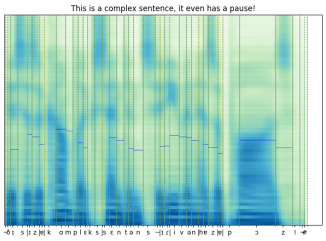

Figure 54. diff pruning vctk p254 50data

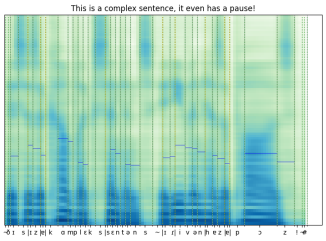

Figure 55. finetuning vctk p254 50data

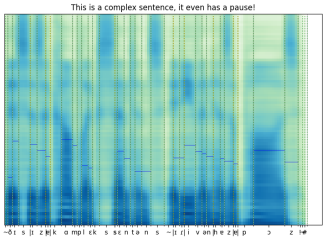

Figure 56. adapter vctk p254 100data

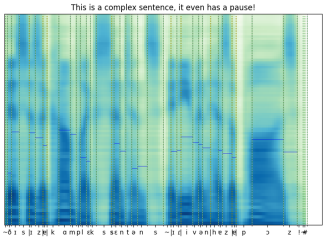

Figure 57. BitFit vctk p254 100data

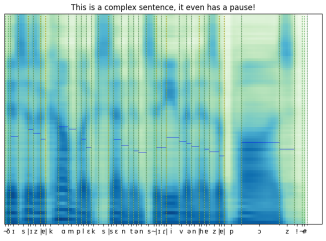

Figure 58. diff pruning vctk p254 100data

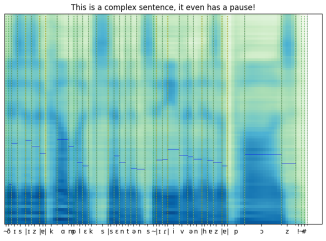

Figure 59. finetuning vctk p254 100data

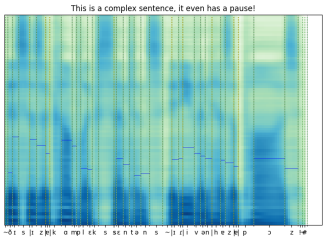

Figure 60. adapter vctk p254 150data

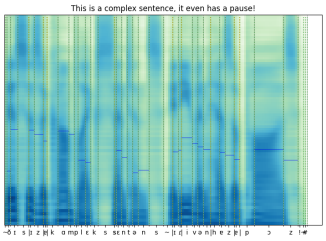

Figure 61. BitFit vctk p254 150data

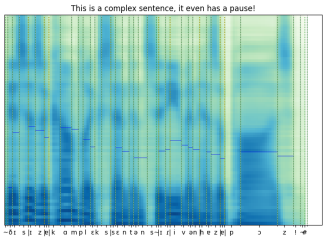

Figure 62. diff pruning vctk p254 150data

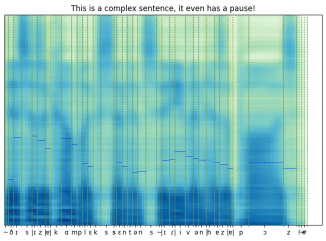

Figure 63. finetuning vctk p254 150data

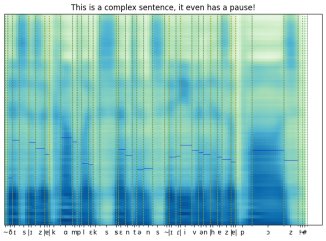

Figure 64. adapter vctk p254 200data

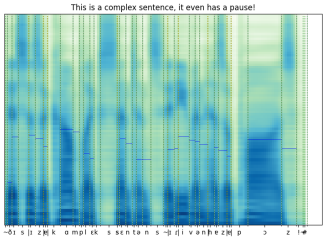

Figure 65. BitFit vctk p254 200data

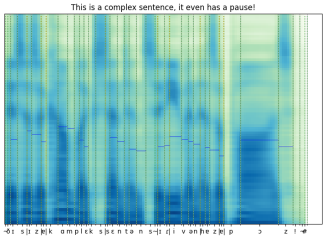

Figure 66. diff pruning vctk p254 200data

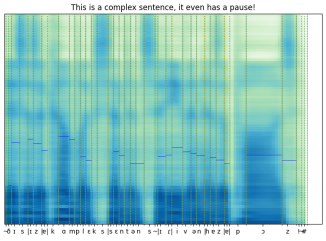

Figure 67. finetuning vctk p254 200data

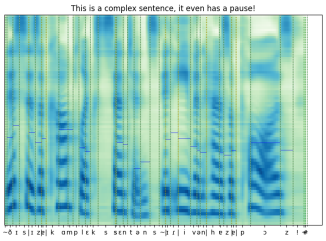

Figure 68. adapter LJspeech

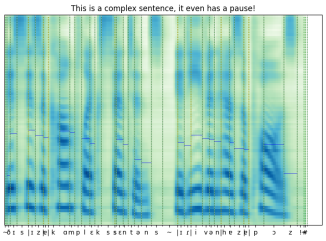

Figure 69. BitFit LJspeech

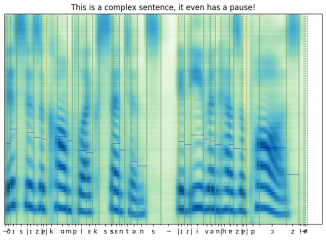

Figure 70. diff pruning LJspeech

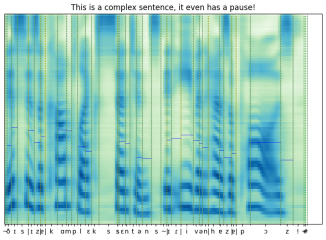

Figure 71. finetuning LJspeech

Appendix C Test samples selected from vctk and LJspeech

For each speaker, we randomly select 6 sentences as test samples to generate target comparison audios to evaluate cosine similarity. And the sub-datasets (50,100,150,200) share the same test samples. The selected test samples for cosine similarity evaluation are attached in Table 12 and Table 13.

And we need to evaluate the MOS of naturalness and similarity for all the mentioned models, we selected 4 sentences for each speaker. And the sub-datasets (50, 200) share the same test samples. The selected test samples for naturalness and similarity evaluation are attached in Tables 14 and 15.

vctk P230/P252/P254	test samples
P230/P252/P254 003	Six spoons of fresh snow peas, five thick slabs of blue cheese, and maybe a snack for her brother Bob.
P230/P252/P254 006	When the sunlight strikes raindrops in the air, they act as a prism and form a rainbow.
P230/P252/P254 008	These take the shape of a long round arch, with its path high above, and its two ends apparently beyond the horizon.
P230/P252/P254 014	To the Hebrews it was a token that there would be no more universal floods.
P230/P252/P254 018	Aristotle thought that the rainbow was caused by the reflection of the sun's rays by the rain.
P230/P252/P254 021	The difference in the rainbow depends considerably upon the size of the drops, and the width of the colored band increases as the size of the drops increases.

Table 12. Test samples selected from vctk P230/P252/P254 to evaluate cosine similarity

LJspeech	test samples
LJ001-0005	the invention of movable metal letters in the middle of the fifteenth century may justly be considered as the invention of the art of printing.
LJ001-00014	And it was a matter of course that in the Middle Ages, when the craftsmen took care that beautiful form should always be a part of their productions whatever they were,
LJ001-0025	imitates a much freer hand, simpler, rounder, and less spiky, and therefore far pleasanter and easier to read.
LJ001-0038	while in fourteen seventy at Paris Udalric Gering and his associates turned out the first books printed in France, also in Roman character.
LJ001-0046	their type is on the lines of the German and French rather than of the Roman printers.
LJ001-0050	and though the famous family of Aldus restored its technical excellence, rejecting battered letters,

Table 13. Test samples selected from LJspeech to evaluate cosine similarity

vctk P230/P252/P254	test samples
P230/P252/P254 008	These take the shape of a long round arch, with its path high above, and its two ends apparently beyond the horizon.
P230/P252/P254 014	To the Hebrews it was a token that there would be no more universal floods.
P230/P252/P254 018	Aristotle thought that the rainbow was caused by the reflection of the sun's rays by the rain.
P230/P252/P254 021	The difference in the rainbow depends considerably upon the size of the drops, and the width of the colored band increases as the size of the drops increases.

Table 14. Test samples selected from vctk P230/P252/P254 to evaluate MOS of naturalness and similarity

LJspeech	test samples
LJ001-0005	the invention of movable metal letters in the middle of the fifteenth century may justly be considered as the invention of the art of printing.
LJ001-00014	And it was a matter of course that in the Middle Ages, when the craftsmen took care that beautiful form should always be a part of their productions whatever they were,
LJ001-0025	imitates a much freer hand, simpler, rounder, and less spiky, and therefore far pleasanter and easier to read.
LJ001-0038	while in fourteen seventy at Paris Udalric Gering and his associates turned out the first books printed in France, also in Roman character.

Table 15. Test samples selected from LJspeech to evaluate MOS of naturalness and similarity

References

[1] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2):48–57, 2014.

[2] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[3] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[4] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*, 2021.

[5] Demi Guo, Alexander M Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning. *arXiv preprint arXiv:2012.07463*, 2020.

[6] Masaaki Nagahara. *Sparsity methods for systems and control*. now Publishers, 2020.

[7] Yi Ren, Chenxu Hu, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech 2: Fast and high-quality end-to-end text to speech. *arXiv preprint arXiv:2006.04558*, 2020.

[8] Miikka Silfverberg, Lingshuang Jack Mao, and Mans Hulden. Sound analogies with phoneme embeddings. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2018*, pages 136–144, 2018.

[9] Keith Ito and Linda Johnson. The ljspeech dataset. 2017. *URL https://keithito.com/LJ-Speech-Dataset*, 2017.

[10] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[11] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

[12] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech: Fast, robust and controllable text to speech. *Advances in neural information processing systems*, 32, 2019.

[13] Kun Song, Heyang Xue, Xinsheng Wang, Jian Cong, Yongmao Zhang, Lei Xie, Bing Yang, Xiong Zhang, and Dan Su. Adavits: Tiny vits for low computing resource speaker adaptation. *arXiv preprint arXiv:2206.00208*, 2022.

[14] Zhaoyang Niu, Guoqiang Zhong, and Hui Yu. A review on the attention mechanism of deep learning. *Neurocomputing*, 452:48–62, 2021.

[15] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

[16] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5457–5466, 2018.

[17] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.

[18] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *Advances in Neural Information Processing Systems*, 34:15908–15919, 2021.

[19] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[20] Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. Montreal forced aligner: Trainable text-speech alignment using kaldi. In *Interspeech*, volume 2017, pages 498–502, 2017.

[21] Robert Dale. Gpt-3: What's it good for? *Natural Language Engineering*, 27(1):113–118, 2021.

[22] Guolin Ke, Di He, and Tie-Yan Liu. Rethinking positional encoding in language pre-training. *arXiv preprint arXiv:2006.15595*, 2020.

[23] Anupama Ray, Sai Rajeswar, and Santanu Chaudhury. Text recognition using deep blstm networks. In *2015 eighth international conference on advances in pattern recognition (ICAPR)*, pages 1–6. IEEE, 2015.

[24] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[25] Hans Marmolin. Subjective mse measures. *IEEE transactions on systems, man, and cybernetics*, 16(3):486–489, 1986.

[26] Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

[27] Dengsheng Zhang and Dengsheng Zhang. Wavelet transform. *Fundamentals of Image Data Mining: Analysis, Features, Classification and Retrieval*, pages 35–44, 2019.

[28] Yuhang Duan, Honghui Li, Mengqi He, and Dongdong Zhao. A bigru autoencoder remaining useful life prediction scheme with attention mechanism and skip connection. *IEEE Sensors Journal*, 21(9):10905–10914, 2021.

[29] Heiga Zen, Viet Dang, Rob Clark, Yu Zhang, Ron J Weiss, Ye Jia, Zhifeng Chen, and Yonghui Wu. Libritts: A corpus derived from librispeech for text-to-speech. *arXiv preprint arXiv:1904.02882*, 2019.

[30] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[31] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[32] Lukas Hauzenberger. Modular diffpruning for bias mitigation/eingereicht von lukas hauzenberger. 2023.

[33] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[34] Jodi Kearns. Librivox: Free public domain audiobooks. *Reference Reviews*, 28(1):7–8, 2014.

[35] Christophe Veaux, Junichi Yamagishi, Kirsten MacDonald, et al. Cstr vctk corpus: English multi-speaker corpus for cstr voice cloning toolkit. *University of Edinburgh. The Centre for Speech Technology Research (CSTR)*, 2017.

[36] Hwancheol Jeong, Sangbaek Lee, Weonjong Lee, Jeonghwan Pak, Jangho Kim, and Juhyun Chung. Performance of gtx titan x gpus and code optimization. *arXiv preprint arXiv:1511.00088*, 2015.

[37] Yuxuan Wang, Daisy Stanton, Yu Zhang, RJ-Skerry Ryan, Eric Battenberg, Joel Shor, Ying Xiao, Ye Jia, Fei Ren, and Rif A Saurous. Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis. In *International Conference on Machine Learning*, pages 5180–5189. PMLR, 2018.

[38] Jiaqi Su, Zeyu Jin, and Adam Finkelstein. Hifi-gan: High-fidelity denoising and dereverberation based on speech deep features in adversarial networks. *arXiv preprint arXiv:2006.05694*, 2020.

[39] Peipei Xia, Li Zhang, and Fanzhang Li. Learning similarity with cosine similarity ensemble. *Information sciences*, 307:39–52, 2015.

[40] Florian Lux, Julia Koch, and Ngoc Thang Vu. Low-resource multilingual and zero-shot multispeaker tts. *arXiv preprint arXiv:2210.12223*, 2022.

[41] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: Solution for Edge Computing Applications*, pages 87–104, 2021.

[42] M Emre Celebi, Fatih Celiker, and Hassan A Kingravi. On euclidean norm approximations. *Pattern Recognition*, 44(2):278–283, 2011.

[43] Robert C Streijl, Stefan Winkler, and David S Hands. Mean opinion score (mos) revisited: methods and applications, limitations and alternatives. *Multimedia Systems*, 22(2):213–227, 2016.

[44] Jay E Anderson. A conceptual framework for evaluating and quantifying naturalness. *Conservation biology*, 5(3):347–352, 1991.

[45] ITU-T. Vocabulary for performance, quality of service and quality of experience, 2017.

[46] Standardization Sector and OF Itu. Itu-t, 2013.