POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

RobVC: An End-to-End Self-Supervised Voice Conversion

Supervisors

Prof. Ahmad-Reza SADEGHI

Prof. Santa DI CATALDO

Candidate

Ahmadreza F. FARAHANI

July 2024

Summary

We have developed a voice conversion model capable of converting audio from one speaker to another. We propose RobVC, to create a voice conversion model with unsupervised samples. Previous efforts on voice conversion focus on explicitly disentangling speech representations to separately encode speaker characteristics and linguistic content from two speakers. Moreover, they use pretrained models on supervised datasets which reduce the robustness of the existing models. In this work, instead of explicitly disentangling attributes with supervised models, we present a framework to train a controllable voice conversion model on entangled speech representations derived from unsupervised learning. First, we develop techniques to derive speaker information and content information from self-supervised reconstruction models. In this training approach, the current state of the synthesis model is used to generate voice-converted variations of an utterance, which serve as inputs for the reconstruction task, ensuring a continuous and purposeful refinement of the model. We demonstrate that incorporating such self-synthesized examples during training improves the speaker similarity of generated speech as compared to a baseline voice conversion model trained solely on heuristically perturbed inputs. RobVC is trained without any text and is applicable to a range of tasks such as zero-shot voice conversion and cross-lingual voice conversion. RobVC achieves state-of-the-art results in zero-shot voice conversion on metrics evaluating naturalness, speaker similarity, and intelligibility of synthesized audio.

Acknowledgements

I would like to express my deepest gratitude to Professor Ahmad-Reza Sadeghi for his expert guidance and to Professor Santa Di Cataldo for her insightful co-tutorship. Thank you for all these important and interesting questions that shaped my thesis, I could not have asked for better mentors.

To everyone who took the time to help me throughout my research, who listened to my ideas and contributed to my thesis, a thousand thank yous, to all my friends, your endless support means the world to me.

Lastly, I would be remiss in not mentioning my family, especially my parents. Their belief in me has kept my spirits and motivation high during this process. I would also like to thank my cat for all the entertainment and emotional support.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
: artificial intelligence

WER
: word error rate

PER
: phoneme error rate

VC
: voice conversion

EER
: equal error rate

CER
: character error rate

TTS
: text-to-speech

ASR
: automatic speech recognition

GRU
: gated recurrent unit

RNN
: recurrent neural network

LSTM	long short-term memory
MSE	mean squared error
CTC	connectionist temporal classification
HMM	hidden Markov model
GAN	generative adversarial network
VAE	variational autoencoder
SNR	signal-to-noise ratio
MFCCs	mel-frequency cepstral coefficients
F0	fundamental frequency
MOS	mean opinion score
PPG	phonetic posteriorgrams
BERT	Bidirectional Encoder Representations from Transformers
AE	autoencoder

RVQ	residual vector quantization
TDNN	time-delay neural network
ASV	automatic speaker verification
MRF	multi-receptive field fusion
MPD	multi-period discriminator
MSD	multi-scale discriminator
MFA	multi-scale feature aggregation
KV	key-value
LM	language model
SSL	semi-supervised learning
SRE	speaker recognition embedding
SR	speech recognition
CPU	central processing unit
GPU	graphics processing unit

Chapter 1

Background

1.1 Speech Processing

Speech processing, an important component of natural language processing (NLP) [1], includes a range of tasks involved in the analysis, synthesis, and understanding of spoken language. Historically, these tasks were predominantly rule-based or relied on handcrafted features. However, the emergence of deep learning [2] has affected speech processing, enabling systems to learn complex patterns and representations directly from raw data, leading to significant advancements in accuracy and performance across various applications.

Deep learning models excel in speech processing tasks due to their capacity to automatically learn hierarchical representations from large volumes of data. Unlike traditional methods, which often struggled to capture the variability and nuances of natural language, deep learning architectures, inspired by the structure and function of the human brain's neural networks, can effectively extract intricate features from raw speech signals or their spectrogram [3] representations.

1.1.1 Advancements in Loss Functions

A crucial aspect of deep learning in speech processing is the design of appropriate loss functions [4]. Loss functions quantify the discrepancy between predicted outputs and ground truth labels, guiding the optimization process during model training. In speech processing tasks like speech recognition and synthesis, various loss functions are employed, to the specific objectives of the task.

For speech recognition tasks, commonly used loss functions include:

- **Categorical Cross-Entropy Loss [5]**: Used in connectionist temporal classification (CTC) [6] and attention-based models, this loss function measures the dissimilarity between predicted and target phoneme sequences, enabling the model to learn to accurately transcribe spoken utterances.

- **Connectionist Temporal Classification (CTC) Loss**: Particularly suited for sequence-to-sequence models, CTC loss allows the model to align variable-length input sequences (acoustic features) with variable-length output sequences (phoneme or word sequences) without requiring explicit alignment information during training.

- **Sequence-to-Sequence Loss** [7]: In tasks where the output sequence length may differ from the input sequence length, such as automatic speech recognition (ASR) and machine translation, sequence-to-sequence loss functions, often combined with attention mechanisms, facilitate the alignment and mapping of input speech features to output text sequences.

For speech synthesis tasks, loss functions are designed to optimize the quality and naturalness of generated speech waveforms. Some commonly used loss functions in speech synthesis include:

- **Mean Squared Error (MSE) Loss** [8]: Frequently employed in waveform generation tasks, MSE loss measures the difference between the predicted and target speech waveforms at each time step, aiming to minimize the overall reconstruction error.

- **Mel-Spectrogram Loss** [9]: Instead of directly modeling the waveform, some speech synthesis models generate mel-spectrogram representations of speech, which capture the spectral characteristics of the audio signal. Mel-spectrogram loss functions optimize the alignment between predicted and target mel-spectrograms, enabling the generation of high-quality synthetic speech.

1.2 Transformer Architecture

The Transformer [10] architecture,has revolutionized the field of natural language processing (NLP) and enabled significant advancements in various sequence-to-sequence tasks, including machine translation [11], text summarization [12], and language modeling [12].

1.2.1 Self-Attention Mechanism

The self-attention mechanism allows the model to weigh the importance of different words in a sentence when encoding or decoding sequences. It computes attention scores between all pairs of words (or other tokens) in the input sequence and uses these scores to compute weighted sums of the corresponding word embeddings. This mechanism enables the model to capture long-range dependencies and contextual information effectively.

1.2.2 Positional Encoding

Since the Transformer architecture does not inherently capture the order of tokens in a sequence like recurrent neural networks (RNNs) [13] or convolutional neural networks (CNNs) [14], positional encodings are added to the input embeddings [15] to provide the model with positional information. These positional encodings are added to the input embeddings before feeding them into the self-attention mechanism.

1.2.3 Transformer Encoder

The Transformer encoder consists of multiple layers, each containing a self-attention mechanism followed by position-wise feedforward networks. The self-attention mechanism in each layer attends to the input sequence independently, and the feedforward networks process the output of the self-attention layer in a position-wise manner. Skip connections and layer normalization are applied around each sub-layer to facilitate training and stabilize the learning process.

1.2.4 Transformer Decoder

The Transformer decoder also consists of multiple layers, similar to the encoder. However, in addition to the self-attention and feedforward layers, each decoder layer includes an additional attention mechanism called the encoder-decoder attention. This attention mechanism allows the decoder to focus on relevant parts of the input sequence when generating the output sequence, enabling effective sequence-to-sequence modeling.

1.2.5 Scaled Dot-Product Attention

The attention mechanism in the Transformer architecture computes attention scores by taking the dot product of the query and key vectors, followed by scaling and applying a softmax [16] function to obtain the attention weights. The attention scores are then used to compute weighted sums of the value vectors, resulting in the attention output.

1.2.6 Multi-Head Attention

To enhance the modeling capacity of the attention mechanism, the Transformer architecture employs multi-head attention. In multi-head attention, the query, key, and value vectors are linearly projected multiple times to different subspaces, and attention is computed independently in each subspace. The outputs of multiple attention heads are concatenated and linearly transformed to produce the final attention output.

1.2.7 Position-wise Feedforward Networks

Position-wise feedforward networks consist of two linear transformations separated by a non-linear activation function [17], such as the ReLU [18] (Rectified Linear Unit). These networks process the output of the attention mechanism independently at each position in the sequence, allowing the model to capture complex relations between different parts of the input sequence.

The Transformer architecture have become foundational in modern language modeling and have led to state-of-the-art performance in various sequence-to-sequence tasks. Its ability to capture long-range dependencies, parallelize computation, and facilitate efficient training has made it a widely adopted architecture in the NLP community.

1.3 Text-to-Speech (TTS) Systems

Text-to-Speech (TTS) [19] systems are technologies that convert written text into spoken speech. They play a crucial role in various applications, including accessibility tools, navigation systems, virtual assistants, and entertainment platforms. TTS systems enable machines to communicate with users in a human-like manner by synthesizing natural-sounding speech from textual input.

1.3.1 Text Analysis

The first step in TTS involves analyzing the input text to extract linguistic features such as phonetic information, sentence structure, punctuation, and language-specific rules. Text preprocessing [20] techniques may include tokenization and part-of-speech tagging.

1.3.2 Prosody Generation

Prosody refers to the rhythm, intonation, stress, and pitch variations in speech that convey meaning and emotions. TTS systems incorporate prosody generation modules to add naturalness and expressiveness to synthesized speech. Prosody generation techniques may involve applying linguistic rules, statistical models, or machine learning algorithms to predict pitch contours, sentence boundaries, and emphasis patterns.

1.3.3 Acoustic Modeling

Acoustic modeling [21] involves mapping linguistic features to acoustic representations, such as Mel-spectrograms or waveform parameters. TTS systems use acoustic models to generate the spectral and temporal characteristics of speech sounds corresponding to the input text. Acoustic modeling techniques include Hidden Markov Models (HMMs) [22], Gaussian Mixture Models (GMMs) [23], deep neural networks (DNNs), and generative models [24] like WaveNet [25] and SampleRNN [26].

1.3.4 Waveform Synthesis

Waveform synthesis is the process of reconstructing the speech waveform from acoustic representations generated by the acoustic model. TTS systems utilize waveform synthesis techniques to generate high-quality, natural-sounding speech signals. Common waveform synthesis methods include concatenative synthesis, statistical parametric synthesis (e.g., HMM-based synthesis) [27], and neural waveform synthesis (e.g., WaveNet, Tacotron, and Transformer-based models).

1.3.5 Naturalness and Intelligibility

Achieving naturalness and intelligibility in synthesized speech remains a significant challenge for TTS systems. Advances in machine learning, particularly deep learning, have led to improvements in speech synthesis quality, enabling TTS systems to produce more natural and human-like speech.

1.3.6 Expressiveness and Adaptability

Enhancing the expressiveness and adaptability of TTS systems to capture emotions, emphasis, and speaker characteristics is an ongoing research area. Techniques such as style transfer (e.g AutoVC) [28], prosody embedding [29], and speaker adaptation [30] allow TTS systems to produce speech with diverse voices, accents, and speaking styles.

1.3.7 Real-time Processing

Real-time processing is essential for interactive applications where low-latency speech synthesis [31] is required. Efficient implementation of neural network architectures and optimization techniques (e.g., pruning, quantization) [32] helps improve the speed and responsiveness of TTS systems.

1.3.8 Multilingual and Multimodal Synthesis

Supporting multiple languages and integrating with other modalities, such as text and images, expands the applicability of TTS systems. Multilingual TTS models and multimodal architectures enable TTS systems to synthesize speech in different languages and contexts, catering to diverse user needs and preferences.

In conclusion, Text-to-Speech (TTS) systems play a crucial role in converting written text into natural-sounding speech, enabling human-computer interaction in various domains. Ongoing advancements in machine learning and speech synthesis techniques continue to enhance the quality, expressiveness, and adaptability of TTS systems, expanding their utility and impact across diverse applications and user populations.

1.4 Voice Conversion

Voice conversion [33] is the process of modifying the characteristics of a speech signal from a source speaker while retaining the linguistic content (context) and prosody of the original utterance. This approach involves extracting the linguistic context from the speech of one speaker and applying the prosodic and spectral characteristics of another speaker to generate converted speech. The process typically consists of several stages, including feature extraction, conversion model training, and synthesis. Below, we elaborate on each step:

1.4.1 Data Collection and Preprocessing

Collect a dataset containing parallel recordings of speech from both the source and target speakers. The dataset should cover various linguistic contexts and speaking styles to ensure robustness. Preprocess the speech recordings by segmenting them into individual utterances, extracting relevant features such as Mel-frequency cepstral coefficients (MFCCs) [34], fundamental frequency (F0) [35], and spectral features like Mel-spectrograms.

1.4.2 Feature Extraction

Extract linguistic features from the source speaker's speech, including phonetic information, duration, and linguistic context. This may involve using automatic speech recognition (ASR) [36] systems to obtain phonetic transcriptions. Extract prosodic features and spectral characteristics from the target speaker's speech, capturing aspects such as pitch, energy, and formant frequencies [37].

1.4.3 Model Training

Train a conversion model that learns the mapping between the linguistic features of the source speaker and the prosodic and spectral features of the target speaker. Commonly used models include Gaussian Mixture Models (GMMs), deep neural networks (DNNs), or more advanced models like Variational Autoencoders (VAEs) [38] or Generative Adversarial Networks (GANs) [39]. The training objective is to minimize the difference between the converted speech and the target speaker's natural speech, typically measured using a loss function such as Mean Squared Error (MSE) or adversarial loss.

1.4.4 Conversion Process

Given a new input utterance from the source speaker, extract its linguistic features. Feed the linguistic features into the trained conversion model to predict the corresponding prosodic and spectral features of the target speaker. Generate the converted speech waveform by combining the converted features with the linguistic context of the source speaker.

1.4.5 Post-processing and Synthesis

Optionally, perform post-processing techniques such as waveform filtering, smoothing, or pitch modification to enhance the quality and naturalness of the converted speech. Reconstruct the converted features into a time-domain waveform using techniques like vocoding [40] or waveform synthesis.

In the Figure 1.1 we have overall architecture of different approaches to generate high fidelity speeches.

1.4.6 Evaluation

Evaluate the quality of the converted speech using objective metrics such as signal-to-noise ratio (SNR) [41], mean opinion score (MOS), or perceptual evaluation of speech quality (PESQ) [42]. Subjective evaluations involving human listeners can

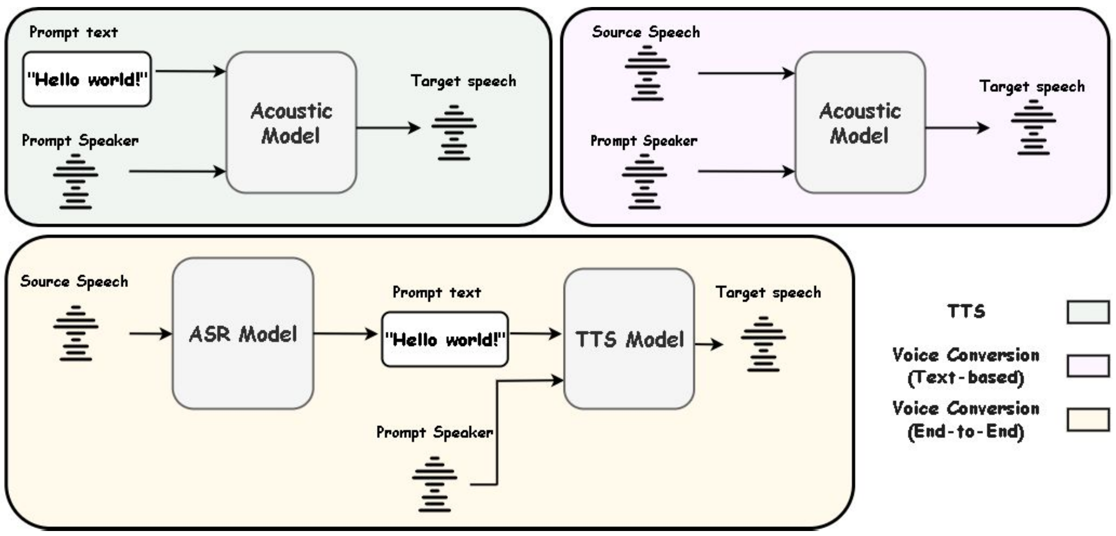


Figure 1.1: Different Speech Synthesis Techniques

also provide valuable insights into the naturalness and similarity of the converted speech to the target speaker.

1.4.7 Fine-tuning and Refinement (Optional)

Fine-tune the conversion model using additional data or regularization techniques to improve performance, especially in cases where the source and target speakers exhibit significant acoustic differences. Iterate on the conversion process based on feedback from evaluations and user testing to further enhance the quality and effectiveness of the voice conversion system.

By following this approach, it is possible to achieve effective voice conversion, allowing for the synthesis of speech with the linguistic content of one speaker and the distinctive characteristics of another speaker. This technology finds applications in voice cloning, accent conversion, personalized speech synthesis, and speaker adaptation in speech recognition systems.

Chapter 2

Related Works

A typical approach of any-to-many VC is feature disentangling. The content feature information and the speaker feature information in the speech are extracted separately and used to reconstruct the speech. The results of this method depend on whether the obtained content features do not contain the original speaker feature information, while not losing information about the speech content.

Techniques from the ASR domain are widely used to extract linguistic content from speech while ignoring speaker specific details.

VC based on phonetic posteriorgrams (PPGs) [43] draws much attention [44, 45]. PPGs are intermediate results from a speaker-independent ASR system, representing the posterior probability of phonetic classes at the frame level [46]. PPGs are independent of speaker and language, making them suitable for VC. However, the accuracy of the PPG-based VC model largely depends on the precision of the ASR model used to extract the PPGs.

In addition to ASR, transfer learning from TTS methods has been employed to obtain linguistic representations for VC [47]. However, these methods require a large amount of annotated data containing text to train the ASR or TTS model.

There are also studies that do not require text annotated data, such as Cyclegan-VC [48] and StarGAN [49], which are GANbased models, and AutoVC, which is an AutoEncoder [50]. But the speech they generate is relatively poor in terms of quality [51]. Recently there has been a lot of new research on VC that attempts to obtain feature vectors of speech by means of self-supervised learning (SSL) models [52, 53, 54, 55]. By obtaining the content representation of speech from these feature vectors, speech can be reconstructed to achieve VC or singing voice conversion (SVC) [56]. These studies achieve very good results in terms of quality and are close to the TTS models.

The development of TTS models has helped to produce high-quality speech based on content features. TTS models such as Tacotron2 [57] and Fastspeech [58, 59] have the ability to synthesize naturalistic speech. They have been applied in the field of VC [60]. However, these TTS methods are two-stage, synthesizing acoustic features first and then using a vocoder to synthesize waveforms from the predicted acoustic features. VITS [61] is an end-to-end TTS model that enhances the quality of the reconstructed waveform through adversarial training [61]. By applying VITS to VC, separate training of the VC model and the vocoder can be avoided

Although there are VC models that are capable of producing speech of good perceptual quality, there is still a lack of research on high-quality and unsupervised VC. In practice, if you want to achieve an unsupervised VC, you need to use self supervised representations for speaker embeddings.

In QuickVC [62] they have used HuBERT [63] embeddings and one designed LSTM [64] module for speaker represenatations, they provide high speed but low accuracy, however the model do not need annotated data.

In some works, they have used audio language models such as WavLM [65] and speaker information from LSTM layer, however their speaker style and disentanglement are not representative.

Diffusion models [66] have shown extraordinary performance in generative tasks in various domains, such as images, videos, and audio, and have recently achieved considerable success in multi-modal tasks [67, 68]. Specifically, in speech, the diffusion model is utilized in applications such as audio generation [69, 70], speech enhancement [71], and TTS synthesis [72, 73]. The fundamental concept underlying the stochastic differential equation (SDE)-based continuous-time diffusion process [74] is to train an estimator that repeatedly removes noise by estimating log-density gradient of data and generates samples with an iterative denoising process via SDE. The SDE-based diffusion model was also applied to VC task using a maximum likelihood (ML)-SDE solver [75] for fast sampling.

In Diff-HierVC [76] they have used XLS-R [77] model for context and pitch embeddings for the speaker. They have also proposed style encoder to disentangle speech style from target speaker. Their model lacks speaker similarity due to not having a good speaker representation. Furthermore, it needs prior training on speaker verification tasks that requires annotated dataset.

In TriAAN-VC [78] they propose Triple Adaptive Attention Normalization VC (TriAAN-VC). TriAAN-VC, which is based on an encoder-decoder structure, disentangles content and speaker features. They do not provide good similarity for unseen speakers.

In HiFi-VC [79] they disentangle content and speaker information from ASR and speaker verification model where both requires annotated data from training.

One of the first zero-shot VC models was AutoVC [28], an autoencoder-based model that utilizes a dimensionality bottleneck to disentangle content and speaker information. It has served as the base model for a range of extensions [80, 81, 82]. Other approaches have used adaptive instance normalization [83] or activation function guidance [84] for information disentanglement. All of these approaches produce spectrograms and require a separate vocoder to synthesize time-domain audio. Relatively few models have been proposed that can perform end-to-end voice conversion. Blow [85] is a normalizing flow network for non-parallel raw-audio voice conversion. However, it is not able to perform zero-shot conversion, and like many other flow-based networks, it has a very large number of parameters. NVC-Net [86] is a GAN-based zero-shot model that performs conversion directly on raw audio waveforms.

In LVC-VC they utilize a neural vocoder that incorporates LVCs [87] as the backbone architecture for LVC-VC. Taking appropriately designed content and speaker features as inputs to the LVC kernel predictors, our model efficiently combines their information to perform voice conversion while directly synthesizing audio. Although they provide good results on preserving context but they cannot provide good speaker similarity.

Chapter 3

Unlabeled Speech Representations

In many speech processing tasks, labeled data have been used to develop different applications. As a result, representations of speech data were dependent on labels. Moreover, different datasets do not support different languages. In unsupervised approach, we have representations that do not consider labels. Humans also have the same path for learning speech. We discover more low level representations in speech data without considering labels.

3.1 Wav2Vec

In this work [88], they have presented a framework for self-supervised learning of representations from raw audio data. This approach encodes speech audio via a multi-layer convolutional neural network and then masks spans of the resulting latent speech representations. The latent representations are fed to a Transformer network to build contextualized representations and the model is trained via a contrastive task where the true latent is to be distinguished from distractors.

The main loss function is contastive loss. To represent the latent representations. The contastive loss cannot propagate into non-differentiable quantized latent speech representations. Therefore, they have used Gumbel softmax.

3.1.1 Key features

In this section, we discuss about key components used in this work provide that we can use them for further development.

Feature encoder. The encoder consists of temporal convolution layer which has raw audio data as input. Instead of fixed positional embeddings which encode

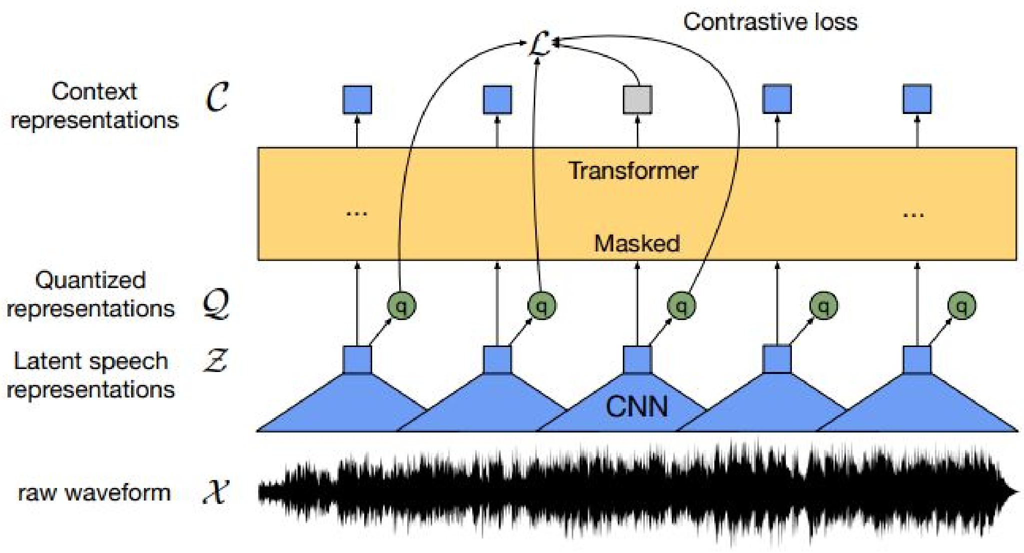


Figure 3.1: Illustration of Wav2Vec architecture

absolute positional information, they use a convolutional layer similar to which acts as relative positional embedding.

Quantization module. For self-supervised training they discretized the output of the feature encoder z to a finite set of speech representations via product quantization. Product quantization amounts to choosing quantized representations from multiple codebooks and concatenating them. We can interprete this approach as detecting low level quanitzed features such as phonemes. For the quantization they considered G codebooks with V entries $e \in \mathbb{R}^{G \times d/G}$. The Gumbel softmax enables choosing discrete codebook entries in a fully differentiable way.

Objective function. The objective function consisits of two parts, conservative loss $\mathcal{L}_m$ and codebook diversity loss $\mathcal{L}_d$ to encourage the model to use the codebook entries equally often.

$$\mathcal{L}_c = \mathcal{L}_w + \alpha \mathcal{L}_d \tag{3.1}$$

$$\mathcal{L}_w = -\log \frac{\exp(sim(c_t, q_t)/k)}{\sum_{\tilde{q} \sim Q_t} \exp(sim(c_t, \tilde{q})/k)} \tag{3.2}$$

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v} \tag{3.3}$$

In diversity loss we encourage the equal use of the V entries in each of the G codebooks by maximizing the entropy. The diversity loss can act as a regulizer to avoid overfitting.

3.1.2 Usecasae

In our work, we require an end-to-end speech transformation approach. We aim to understand methods that can represent audio without relying on labels, such as text. Therefore, the quantized or encoded data from the transformer can serve as a representation of the speech data with respect to low-level speech features. Moreover, we divide the representations of interest for our work into two parts: speech representations and speaker representations. With fine-tuning this model, we can obtain both types of representations.

3.2 Wav2Vec-BERT

This paper [89] is an extension of BERT paper [90] released in 2018. Before we discuss about wav2vec-BERT, the novelty of BERT will be briefly discussed.

3.3 BERT summary

BERT initial model have been used for designing language models. The base architecture is based on transformer's encoder part. It consists of twp parts, one pre-training part and fine-tuning part. In **pre-training** part, the model tries to learn word embedding in an unsupervised manner, two tasks will be done in pre-training part:

1. Masked LM; In this task model tries to mask some percentage of the input tokens at random, and then predict those masked tokens.
2. Next Sentence Prediction (NSP); Model tries to predict the next sentence.

In the Figure 3.2 we can see the architecture of BERT. Both tasks will be done simultaneously. In **fine-tuning part** for each task, we simply plug in the tasks specific inputs and outputs into BERT and fine-tune all the parameters end-to-end.

3.3.1 BERT in Wav2Vec

In Wav2Vec original architecture, we had masked latent speech representations and with a contrastive loss and quantizaton, we could optimize the model based on what we had in BERT. Now in Wav2Vec BERT [89] they added another module on top of the context representations of original Wav2Vec paper.

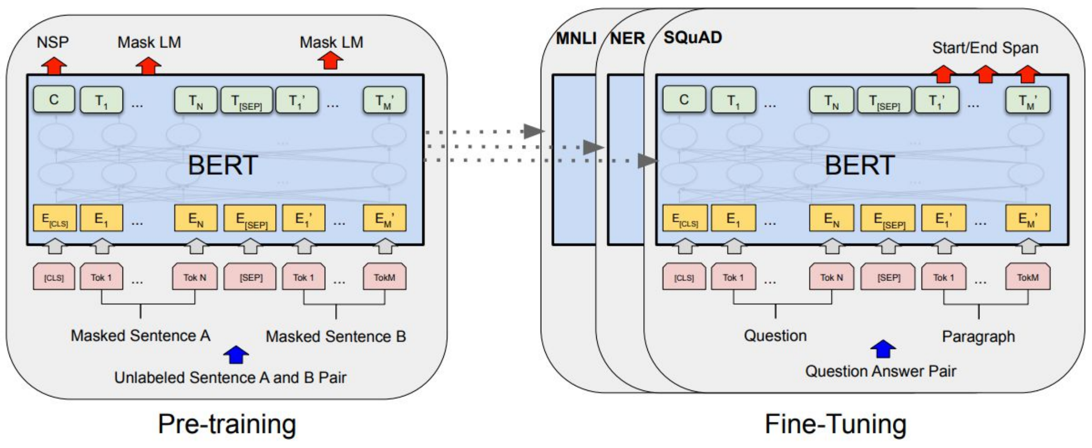


Figure 3.2: Illustration of BERT architecture

The context vectors produced by the contrastive module are directly passed to the masked prediction module for producing the final context vectors that are to be used to complete a masked prediction task. A softmax layer is appended on top of the module's last conformer block. If a context vector at the final layer corresponds to a masked position, the softmax layer will take the context vector as input and attempt to predict its corresponding token ID, which is assigned earlier in the contrastive module by the quantizer. We denote the cross-entropy loss for this masked prediction task as $\mathcal{L}_m$. Therefore, the final loss will be combination of contrastive loss [91] and cross entropy loss [92]. If we denote contrastive loss as $\mathcal{L}_C$, we have total loss $\mathcal{L}_p$.

$$\mathcal{L}_p = \beta\mathcal{L}_c + \gamma\mathcal{L}_m \tag{3.4}$$

3.3.2 Use case

In this work, we have more detailed representations of audios. The BERT extension provide better accuracy and outperforms previous paper. One of the key component of this research is using BERT which was originally used for developing language models in an unsupervised audio representation task.

Since we have an self-supervised end-to-end task for audio transformation, these two papers and their objectives, can be further used in our work. We can define representations that are fully unsupervised which can be fine-tuned. Moreover, the masking method which used in training phase can be used in further parts of our research.

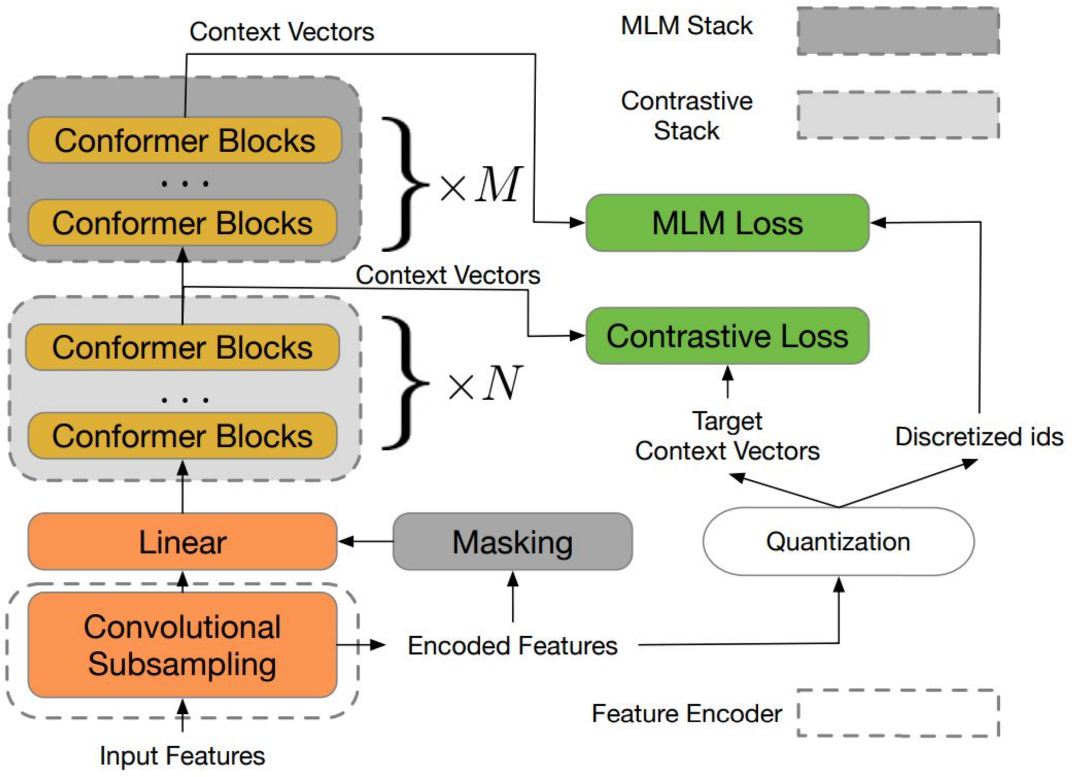


Figure 3.3: Illustration of WavtVec architecture

3.4 SoundStream

SoundStream [93] is a neural audio codec [94] in which all the constituent components (encoder, decoder and quantizer) are trained end-to-end with a mix of reconstruction and adversarial losses to achieve superior audio quality.

3.4.1 Autoencoder part

The encoder architecture is illustrated in Figure 3.4. Each of the blocks consists of three residual units, containing dilated convolutions with dilation rates of 1, 3, and 9, respectively, followed by a down-sampling layer in the form of a strided convolution. The decoder block mirrors the encoder block, and consists of a transposed convolution for up-sampling followed by the same three residual units. We use the same strides as the encoder, but in reverse order, to reconstruct a waveform with the same resolution as the input waveform.

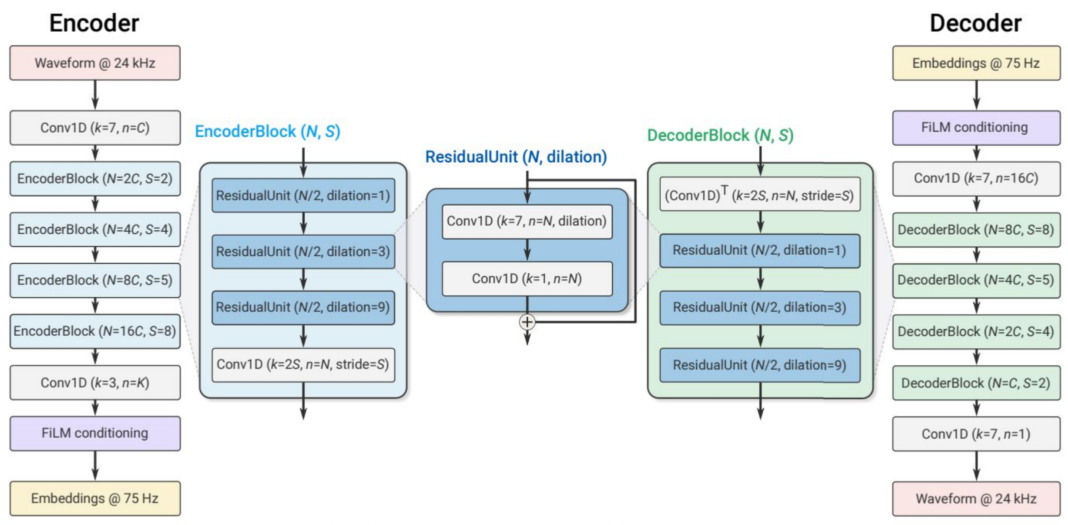


Fig. 3: Encoder and decoder model architecture.

Figure 3.4: Illustration of SoundStream architecture

3.4.2 Residual Vector Quantizer (RVQ)

Before discussing about the proposed RVQ method in this paper, we have to discover why quanitization have been used. The purpose of Vector Quantization (VQ) [95] is reduce the bit rate by quanitizing the output of encoder. In other words, We have a dictionary of quantization vectors that we choose which vector in this dictionary is more similar to the encoded embeddings.

However, original implementation of quantization have some limitations. The bit rate will be huge during transformation. To address this issue, they proposed RVQ, which cascades N_q layers of VQ as follows. The unquantized input vector is passed through a first VQ and quantization residuals are computed. The residuals are then iteratively quantized by a sequence of additional N_q 1 vector.

3.4.3 Use case

The RVQ mechanism used in this paper, can have low level representations on audio signals. They also provided a genereator part which reconstruct high fidelity audio signal. Unlike previous papers which they tried to represent contextual represntations of audios, this paper provide a reconstruction method. In our work, we need to build a model that reconstruct an audio preserving new speaker. The RVQ can be used in our research to quantize the output of encoder. We need both high level representations like speaker embedding and low level audio reconstruction embedding at the same time.

3.5 EnCodec

EnCodec [96] is a high-fidelity, audio codec leveraging neural networks. It consists in a streaming encoder-decoder architecture with quantized latent space trained in an end-to-end fashion. The EnCodec model is a simple streaming, convolutional-based encoder-decoder architecture with sequential modeling component applied over the latent representation, both on the encoder and on the decoder side.

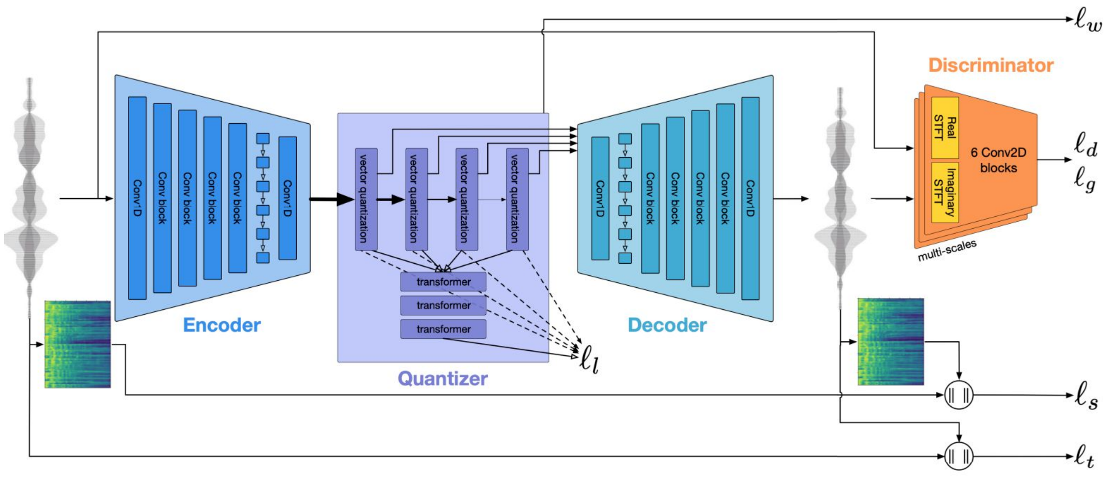


Figure 3.5: Main architecture of EnCodec

They have proposed another discriminator for the generated speech data. In the MS-STFT Discriminator architecture, The input to the network is a complex-valued STFT with the real and imaginary parts concatenated. Each discriminator is composed of a 2D convolutional layer, followed by 2D convolutions with increasing dilation rates. Then a final 2D convolution is applied.

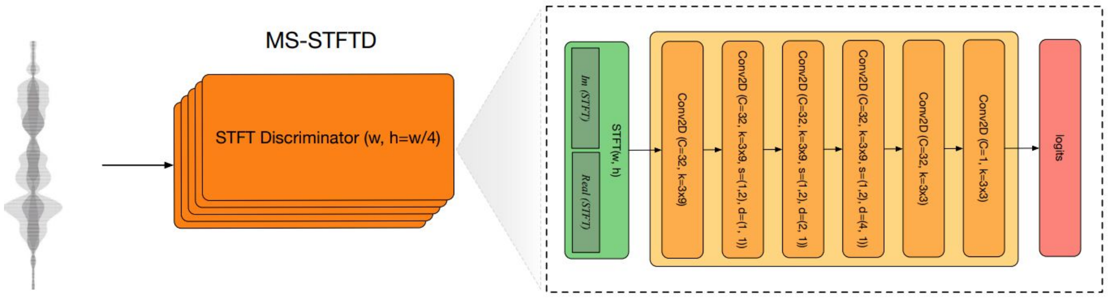

MS-STFTD
STFT Discriminator (w, h=w/4)
STFT(w, h)
Im (STFT)
Real (STFT)
Conv2D (C=32, k=3x9)
Conv2D (C=32, k=3x9, s=(1,2), d=(1, 1))
Conv2D (C=32, k=3x9, s=(1,2), d=(2, 1))
Conv2D (C=32, k=3x9, s=(1,2), d=(4, 1))
Conv2D (C=32, k=3x3)
Conv2D (C=1, k=3x3)
logits

Figure 3.6: MS-STFT Discriminator architecture

3.5.1 Use case

This model is an enhanced version of SoundStream [93]. They have added few features to the previous paper. However, they did not compare the result with original SoundStream paper.

3.6 HuBERT

In the previous section, we attempted to acquire W2V-BERT [89] for extracting context tokens. However, we decided to replace W2V-BERT with HuBERT [63] for two main reasons.

First, the HuBERT model offers state-of-the-art representations of speech context, similar to W2V-BERT. Second, the pretrained model is available for feature extraction and has accelerated our work speed. In this section, I will demonstrate the key features of HuBERT.

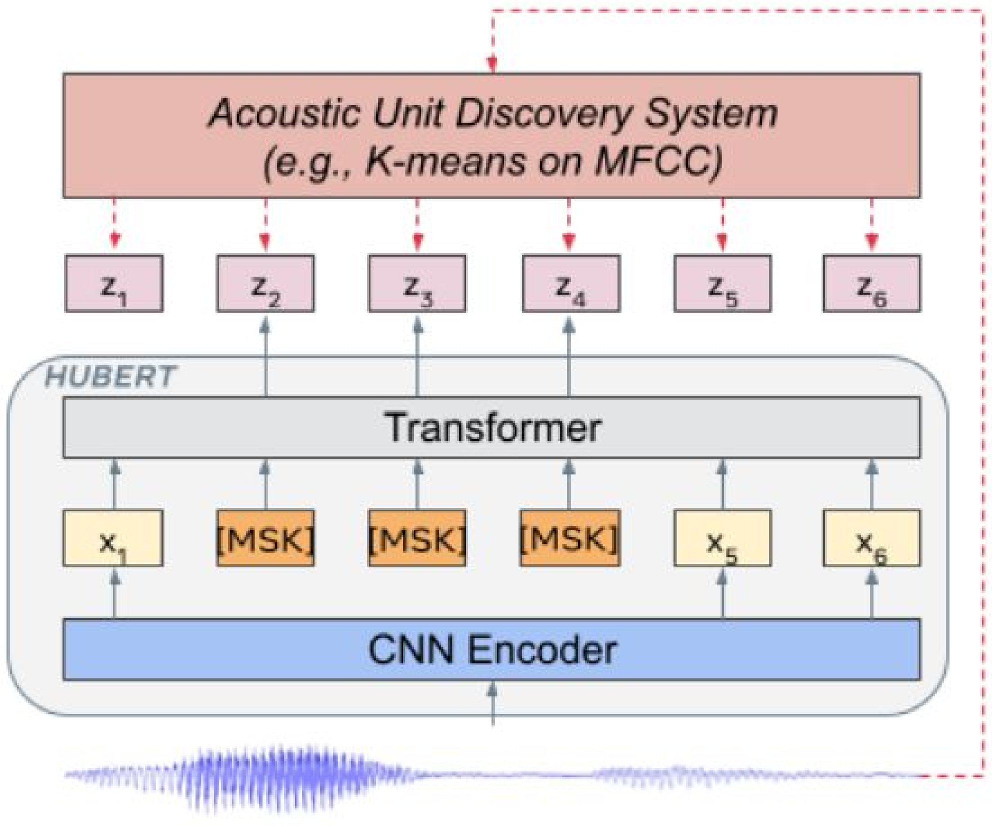


Figure 3.7: HuBERT architecture

HuBERT utilizes an offline clustering step to provide aligned target labels for a BERT-like prediction loss in speech representations. This model also provides self-supervised speech representations used for different tasks.

3.6.1 Architecture

In this paper, they introduced Hidden unit BERT (HuBERT) that benefits from an offline clustering step to generate noisy labels for a BERT-like per-training. Concretely, a BERT model [90] consumes masked continuous speech features to predict predetermined cluster assignments.

The predictive loss is only applied over the masked regions, forcing the model to learn good high-level representations of unmasked inputs to infer the targets of masked ones correctly. Intuitively, the HuBERT model is forced to learn both acoustic and language models from continuous inputs.

First, the model needs to model unmasked inputs into meaningful continuous latent representations, which maps to the classical acoustic modeling problem. Second, to reduce the prediction error, the model needs to capture the long-range temporal relations between learned representations.

One crucial insight motivating this work is the importance of consistency of the targets, not just their correctness, which enables the model to focus on modeling the sequential structure of input data. In our work, we use the intermediate output cluster tokens for context embedding.

Chapter 4

High Fidelity Speech Synthesis

High fidelity speech synthesis refers to the generation of speech that closely resembles human speech in terms of naturalness, clarity, and overall quality. In high fidelity speech synthesis, the generated speech sounds very natural and is often indistinguishable from human-produced speech.

4.1 VALL-E

They proposed a language modeling approach for text to speech synthesis (TTS). Specifically, they trained a neural codec language model (called VALL-E [97]) using discrete codes derived from an off-the-shelf neural audio codec model, and regard TTS as a conditional language modeling task rather than continuous signal regression as in previous work.

Their main interest is to generate given content for unseen speakers. The model is given a text sentence, a segment of enrolled speech, and its corresponding transcription.

4.1.1 Use case

Their work is similar our work, except we put audio as input not text. In other words, they have provided a state-of-the-art TTS language model. We try to consider their novelty in our work.

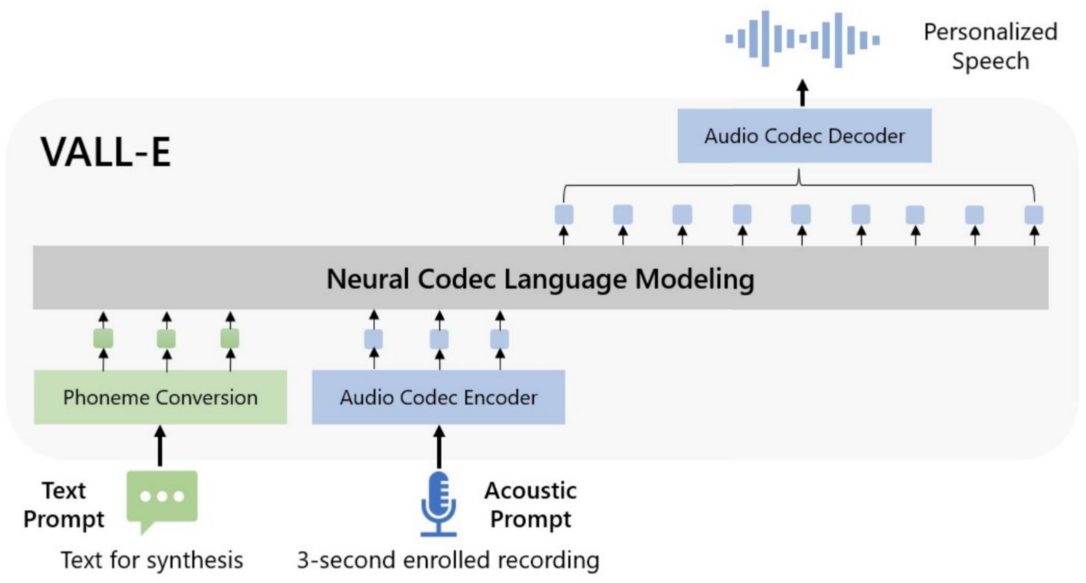


Figure 4.1: MS-STFT Discriminator architecture

4.2 MelGAN

MelGAN [98] is a non-autoregressive feed-forward convolutional architecture to perform audio waveform generation in a GAN setup. The architecture is a fully convolutional feed-forward network with mel-spectrogram s as input and raw waveform x as output.

4.2.1 Generator

Induced receptive field In convolutional neural network based generators for images, there is an inductive bias that pixels which are spatially close-by are correlated because of high overlap among their induced receptive fields. We design our generator architecture to put an inductive bias that there is long range correlation among the audio timesteps. We added residual blocks with dilations after each upsampling layer, so that temporally far output activations of each subsequent layer has significant overlapping inputs.

Moreover, other design methods have been used in this paper such as **Checkerboard artifacts** and **Normalization**.

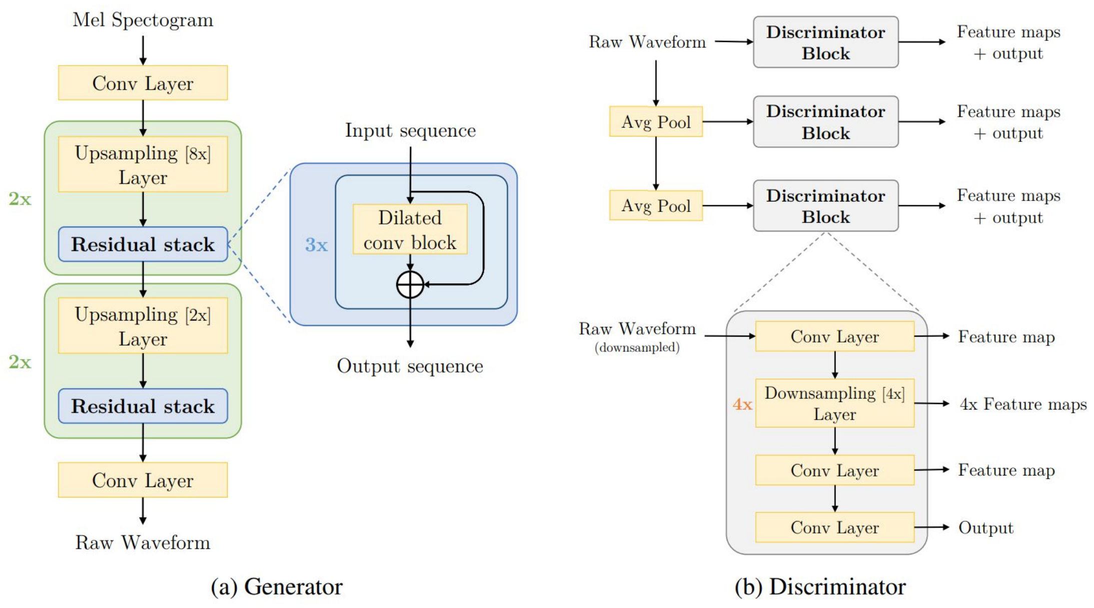


Figure 4.2: Illustration of MelGan architecture

4.2.2 Discriminator

They have adopted a **multi-scale architecture** with 3 discriminators (D_1, D_2, D_3) that have identical network structure but operate on different audio scales. D_1 operates on the scale of raw audio, whereas D_2, D_3 operate on raw audio downsampled by a factor of 2 and 4 respectively.

Multiple discriminators at different scales are motivated from the fact that audio has structure at different levels. This structure has an inductive bias that each discriminator learns 4 features for different frequency range of the audio. For example, the discriminator operating on downsampled audio, does not have access to high frequency component, hence, it is biased to learn discriminative features based on low frequency components only.

Furthermore, they have used $Window - basedobjective$ where each individual discriminator is a Markovian window-based discriminator consisting of a sequence of strided convolutional layers with large kernel size.

4.2.3 Feature matching

In addition to the discriminator's signal, they used a feature matching objective to train the generator. This objective minimizes the L1 distance between the discriminator feature maps of real and synthetic audio. Therefore, they have added another loss term to regular generator loss.

$$\mathcal{L}_{FM}(G, D_k) = \mathbb{E}_{x,s\sim p_{data}}\left[\sum_{i=1}^{T} \frac{1}{N_i} \left\| D_k^{(i)}(x) - D_k^{(i)}(G(s)) \right\|_1 \right] \tag{4.1}$$

We have $\mathcal{L}_{FM}$, which is their feature matching loss. For simplicity of notation, $D_k^{(i)}$ represents the $i_t h$ layer feature map output of the $k_t h$ discriminator block, N_i denotes the number of units in each layer.

4.2.4 Use case

In this paper, they have adopted novel techniques to generate high fidelity speech data from mel-spectrogram of an input text. In our project we do not have access to text and mel-spectrogram generated by text. However, we can apply the discrimination and generation approaches provided in this paper in our work.

Although they have used multi-scale discriminator to discriminate real audios from fake, We can adopt multi-scale discriminator, to have different levels of discrimination between different speakers.

We can also have feature matching technique in our work, when we try to match the output speaker embedding to source embedding. Other techniques such as "induced receptive field" also can be used in our future model archititure.

4.3 HiFi-GAN

HiFi-GAN [99] consists of one generator and two discriminators: multi-scale and multi-period discriminators. The generator and discriminators are trained adversarially, along with two additional losses for improving training stability and model performance.

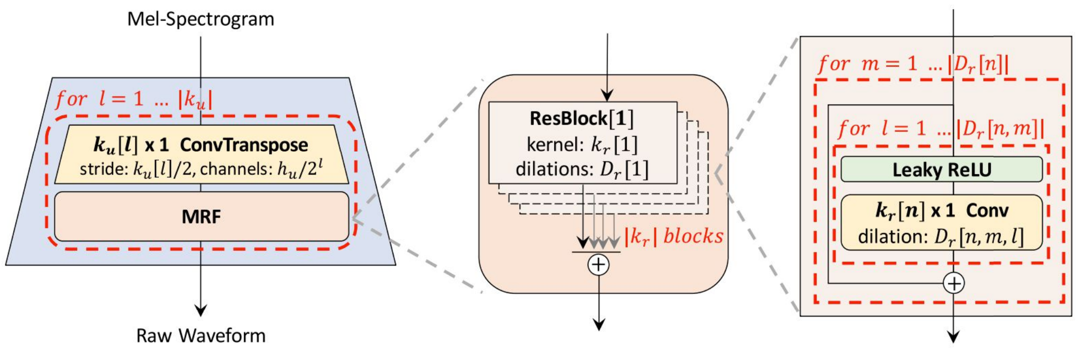


Figure 4.3: Illustration of MelGan architecture

4.3.1 Generator

In the generator part they have proposed **Multi-Receptive Field Fusion**, which observes patterns of various lengths in parallel. Specifically, MRF module returns the sum of outputs from multiple residual blocks. Different kernel sizes and dilation rates are selected for each residual block to form diverse receptive field patterns.

4.3.2 Discriminator

Identifying long-term dependencies is the key for modeling realistic speech audio. To address this issue they have used multi-period discriminator (MPD) consisting of several sub-discriminators each handling a portion of periodic signals of input audio. Additionally, to capture consecutive patterns and long-term dependencies, they used the multi-scale discriminator (MSD) proposed in MelGAN, which consecutively evaluates audio samples at different levels.

4.3.3 Usage

The novelty of this GAN model is MPD and MRF module that they used in generator and discriminator. We can adopt these features in our work.

4.4 Methods

Based on other studies and researches, We can propose two methods to address an end-to-end speech transformation problem.

4.4.1 First method

We can design an auto-encoder which uses speech wave x_t as input and outputs the generated signal with desired target speaker signal y_t. We also try to use existed speaker embedding extraction models to generate source and target speaker embeddings.

In the **Encoder** part, we encode the signal to a source feature embedding z_{sfe}. We concatenate target speaker embedding to z_{tse} which will be fed into decoder part. For the encoder architecture, we can use conformer blocks which used in Wav2Vec paper.

In the **Decoder**, we fed the decoder the countenance of two z_{sfe} and z_{tse} vectors. The output of the decoder y_t is the generated signal with target speaker characteristics. Moreover, as we observed in SoundStream paper, we can provide a discriminator to maintain high fidelity output speech audio.

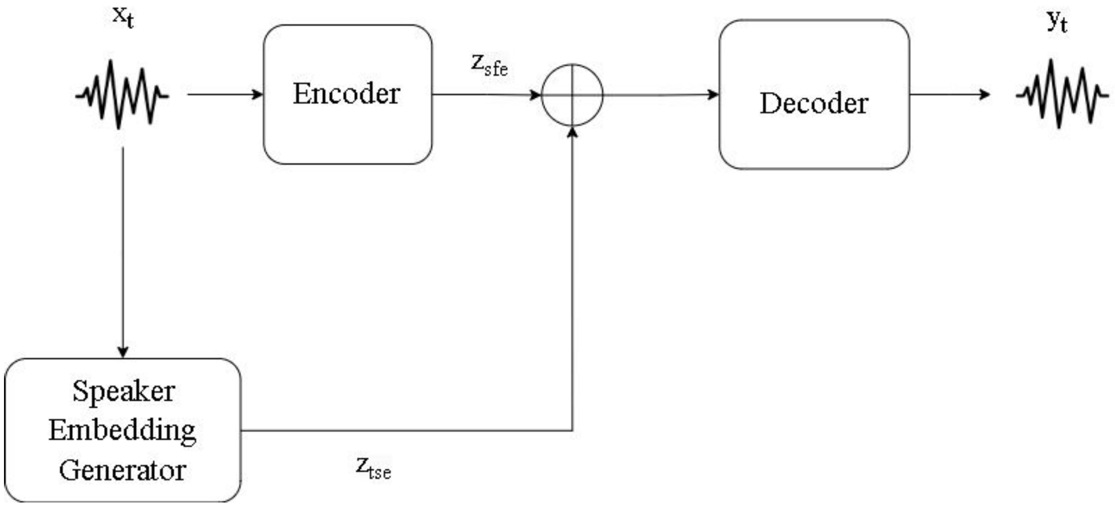


Figure 4.4: First approach basic architecture

The main objective is to reconstruct low level phonemes with new speaker. To this end, we have to reconstruct the speech data while the speaker embeddings of source and target maintain their similarity. The generated signal should have almost identical phonemes as source. The main loss function $\mathcal{L}_t$ should be a combination of speaker's feature matching loss $\mathcal{L}_{sfm}$ and reconstruction of the original audio $\mathcal{L}_{rec}$. We have also generator loss $\mathcal{L}_{gen}$.

$$\mathcal{L}_t = \lambda\mathcal{L}_{rec} + \mathcal{L}_{fm} + \mathcal{L}_{sfm} + \mathcal{L}_{gen} \tag{4.2}$$

4.4.2 Second method

In the first approach we have a trade-off between maintaining the target speaker characteristics and reconstructing the exact input signal, which is not desirable. If we pay attention to reconstruction, we do not obtain good speaker's target embedding in output, if we pay more attention to changing speakers, we can have bad reconstruction. To address this issue we propose another approach.

We have models that transform speech to speech, but it is not end-to-end. We can use these models to generate new dataset. In the new dataset we have source speech data as x and the main objective is to reconstruct the input x.

The architecture of the model will be the same as first approach that we saw in Figure 4.4. However, we do not consider $\mathcal{L}_{sfm}$, instead we adopt another loss from discriminator for matching target and source signal.

Having a good speaker embedding extractor is a key component to our research. In the next two weeks I will research state-of-the-art speaker recognition models.

Meanwhile, I try to investigate current existed not end-to-end deep fake speech transformation which can be acquired to generate new dataset. Furthermore, I dig deep into current speech datasets used in current research works.

Chapter 5

Speaker Recognition Tasks

Let's consider related papers[1] in speaker recognition tasks. Moreover, I analyze works that perform an end-to-end speech analysis. Furthermore, I dig deep in current datasets that can be used in our work.

Automatic speaker verification (ASV) [100] is a task to verify whether a given utterance is from a claimed enrolled speaker. In our project, detecting speakers and extracting good representations for each speaker plays a crucial role in developing the future model. I have analyzed two of most recent state-of-the art papers.

5.1 ECAPA-TDNN

In this paper [101], they proposed multiple enhancements to the Time Delay Neural Network (TDNN) [102] based on recent trends in the related fields of face verification and computer vision. Firstly, the initial frame layers can be restructured into 1-dimensional Res2Net modules with impactful skip connections.

Before using neural networks for speaker recognition, statistic models were used. One of the most notable method was extracting I-vectors [103]. I-vectors are one dimensional vectors which represent speaker features. After I-vectors, x-vectors [104] introduced which had DNN based architecture as we see in Figure 5.1. x-vectors and their subsequent improvements have consistently provided state-of-the-art results on the task of speaker verification.

5.1.1 Time Delay Neural Network (TDNN)

For contextual modelling in a TDNN, each neural unit at each layer receives input not only from activations/features at the layer below, but from a pattern of unit

[1] I try to summarize their paper, so most of the reviews are based on the original paper.

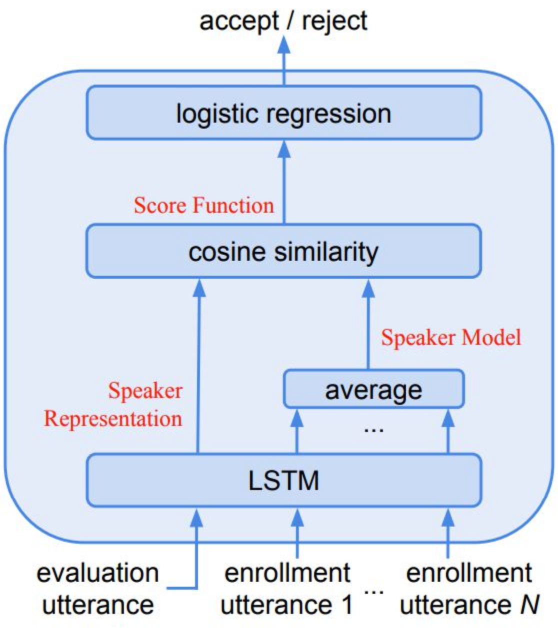


Figure 5.1: Illustration of DNN based architecture for speaker embeddings

output and its context. For time signals each unit receives as input the activation patterns over time from units below. Applied to two-dimensional classification (images, time-frequency patterns), the TDNN can be trained with shift-invariance in the coordinate space and avoids precise segmentation in the coordinate space.

5.1.2 Model Architecture

Two types of DNN-based speaker recognition architectures will serve as strong baselines to measure the impact of their proposed architecture. These two types are:

1. **Extended-TDNN x-vector:** The first baseline system is the Extended TDNN x-vector [104] architecture and improves upon the original x-vector system. The initial frame layers consist of 1-dimensional dilated convolutional layers interleaved with dense layers. Every filter has access to all the features of the previous layer or input layer. The task of the dilated convolutional layers is to gradually build up the temporal context.

2. **ResNet-based r-vector:** The second baseline system is the r-vector system. It is based on the ResNet18 [105] and ResNet34 implementations of the successful ResNet architecture.

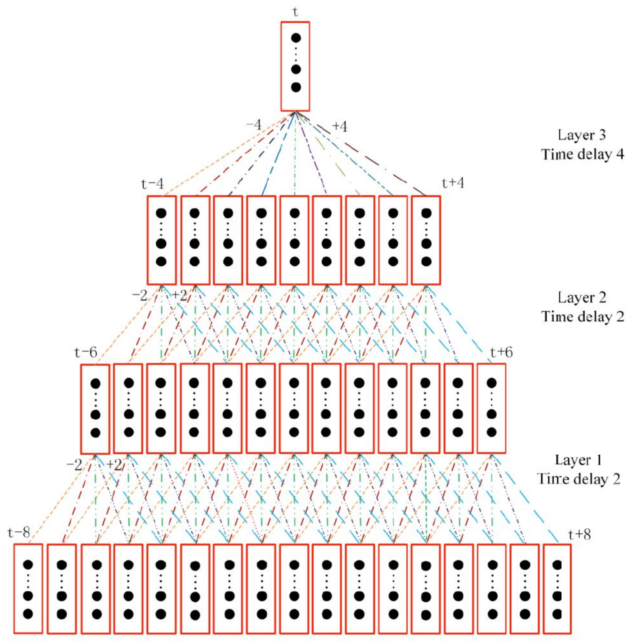


Figure 5.2: Structure of TDNN

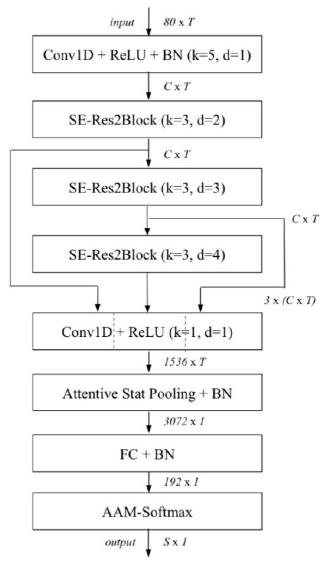


Figure 5.3: Full architecture of ECAPA-TDNN

Speaker embeddings are extracted from the final fully connected layer for all systems. All the scores are normalized using adaptive s-norm. We have a 1-dimensional embedding with 1024 size. The EER on VoxCeleb [106] dataset was 0.87 which we compare it with the next paper.

5.1.3 Usage

As discussed in the previous section, we require a robust representation of speaker features. The research mentioned above is a suitable fit as it offers a state-of-the-art model for creating embeddings for speakers.

5.2 MFA-Conformer

In this paper [107], they presented Multi-scale Feature Aggregation Conformer (MFA-Conformer), an easy-to-implement, simple but effective backbone for automatic speaker verification based on the Convolution-augmented Transformer (Conformer). The architecture of the MFA-Conformer is inspired by recent state-of-the-art models in speech recognition and speaker verification.

In previous paper, they used CNNs for TDNN architecture. However, despite the great success, CNN still has its limitations. It mainly focuses on local spatial modeling, but lacks of global context fusion. CNNs-based models can not handle the long range dependencies very well.

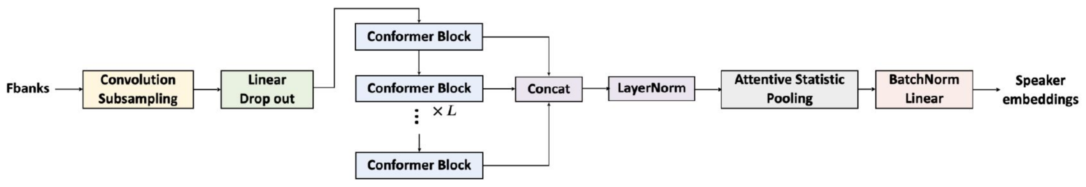


Figure 5.4: Full architecture of MFA-COnformer

MFA-Conformer is an easy-to-implement and effective backbone for speaker embedding extraction. Firstly, the input acoustic feature is processed by a convolution subsampling layer to decrease the computational cost. Secondly, we adopt Conformer blocks which combine Transformers and convolution neural networks to capture the global and local features effectively.

5.2.1 Use case

In this work, we have seen new architecture for extracting embeddings. This model outperforms previous ECAPA-TDNN paper. They have reached an EER of 0.64 in

VoxCeleb datasets. The final vector is one dimensional with size 2048.

Chapter 6

Language Models

In this section we analyze the current state-of-the-art language models. The most recent one is AudioLM [108] which developed by google research team.

6.1 LLaMA Overview

As discussed previously, transformers form the basis of our model architecture, offering advantages in capturing temporal features from the input. To enhance the performance of a vanilla transformer, we can add new layers and state-of-the-art modules and methods.

In this section, I aim to discuss the innovations introduced in the recent language model published by Meta. While the LLaMA [109] paper primarily focuses on text language modeling, it incorporates state-of-the-art components into the transformer model. We can integrate these components into the two transformer models discussed earlier. The most crucial components of this paper include:

- KV cache
- RMSNorm
- SwiGLU
- Grouped Query Attention
- Rotary Positional Embedding

6.1.1 KV cache

Before we discuss about KV cache and group query attention, we need to see how self-attention works.

Self-attention

Self-attention, also known as scaled dot-product attention, is a crucial component of the Transformer architecture. It allows the model to weigh the importance of different words in a sequence when processing each word. This mechanism enables the model to consider the context of each word in relation to all other words in the sequence.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{6.1}$$

Where Q is the query matrix, K is key, V is the value matrix and d_k is the dimensionality of the model.

Caching Key and Value

From attention mechanism, we know that every predicted token has attention with it's previous tokens. In the inference phase we do not need all the tokens embedding, the last token captures information of the previous tokens. KV cache is a method to reduce the computation costs during inference.

6.1.2 RMSNorm

In vanilla transformer we use LayerNorm (layer normalization) [110]. In general, we want to avoid internal covariate shift because it makes the training slower as the neurons are forced to re-adjust drastically their weights in one direction or another.

$$\bar{x_i} = \frac{x_i - \text{E[x]}}{\sqrt{\text{VAR}[x] + \epsilon}} * \gamma \; + \beta \tag{6.2}$$

Where γ and β are learnable parameters that allow model to amplify the scale of each feature. We calculate E[x] and VAR[x] for each rows. With batch normalization we normalize by columns (features) and with layer normalization we normalize by rows (token embeddings).

A well known explanation of the success of LayerNorm is its re-centering and re-sxaling invariance property. In RMSNorm [111] we hypothesize that the re-scaling invariance is the reason for success of LayerNorm, rather than re-centering invariances.

$$\bar{x}_i = \frac{x_i}{\text{RMS(x)}} g_i, \quad \text{where} \;\; \text{RMS}(x) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{6.3}$$

As we can see in the above formulation, we don't need to calculate the mean and apply subtract it from our data. We have also g_i which is a learnable parameter. The main advantage of this normalization, is it requires less computation.

6.1.3 SwiGLU

Feed forward layer in vanilla transformer is a linear layer with ReLU activation function [**deep**]. However, we have seen good performance with enhanced version of ReLU activation functions. For the vanilla transformer we had:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{6.4}$$

In SwiGLU [112] we use swish [113] function with $\beta = 1$. In this case it's called the Sigmoid Linear Unit (SiLU) function [114].

$$swish(x) = x \text{ sigmoid}(\beta x) = \frac{x}{1 + e^{-\beta x}} \tag{6.5}$$

Now for the new feed forward layer we can have:

$$FFN_{SwiGLU}(x, W, V, W_2) = (Swish_1(xW) \otimes xV)W_2 \tag{6.6}$$

It is also used in LLaMA [109]. In Figure 6.1 we can see the difference between ReLU and SiLU activation function.

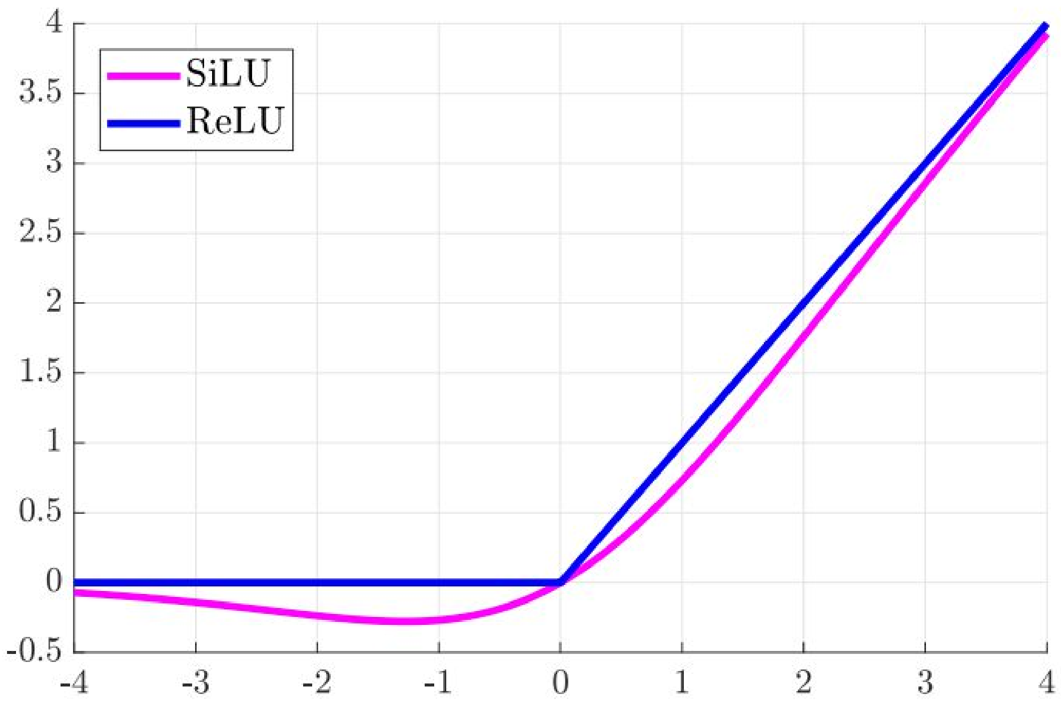


Figure 6.1: ReLU and SiLU activation function

6.1.4 Grouped Query Attention

In recent years, GPUs have become very fast at performing calculations, insomuch that the speed of computation (FLOPs) is much higher than the memory bandwidth (GB/s) or speed of data transfer between memory areas.

This implies that sometimes, the bottleneck is not determined solely by the number of operations we perform but also by the amount of data transfer required for these operations. The latter is contingent on the size and quantity of tensors involved in our calculations.

For example, computing the same operation on the same tensor N times may be faster than computing the same operation on N different tensors even if they have the same size, this is because the GPU may need to move the tensors around.

We can conclude that our goal should not only be to optimize the number of operations we perform but also to minimize the memory access we undertake. For the grouped query attention [115], we remove h dimension (number of heads) from the K and the V, while keeping it for the Q. Therefor, all the different query heads share the same keys and values. The performance gains are important, while the model's quality degrades only a little bit.

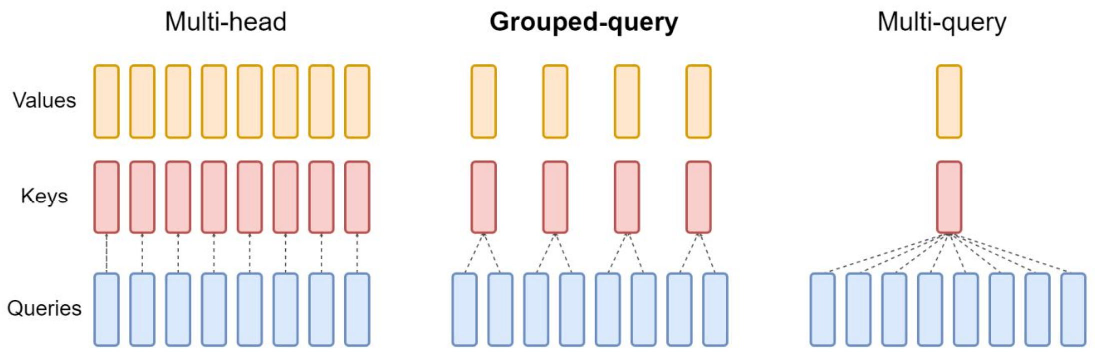


Figure 6.2: Overview of different attention mechanisms

In the vanilla transformer we have **Multi-Head** attention which has high quality but it is slow in computation. In **Multi-Query** attention we have loss in quality but high speed in computation. In the **Grouped Multi-Query** attention there is a good compromise between quality and speed.

6.1.5 Rotary Positional Embedding

For using transformers we need to use specific positional encoding layer. In the first paper [10] they have used absolute positional encoding, however better approaches proposed where it increase the performance and speed of the model. In this section we analyze different existing positional encoding.

Absolute Positional Encodings For transformers, it is essential to have a representation that captures the positions of the tokens. The vanilla transformer uses absolute positional encoding, where no learnable parameter is involved in this transformation. In other words, absolute positional embeddings are fixed embeddings that are added to the original embeddings. This method deals with one token at a time.

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}} \tag{6.7}$$

Relative Positional Encodings On the other hands, relative positional encodings [116] deals with two tokens at a time and it is involved when we calculate the attention. Since the attention mechanism captures the intensity of how much two tokens are related to each other, relative positional encodings tells the attention mecahnism the distance between two tokens involve in it. Given two tokens, we create a vector that represents their distance.

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}} \tag{6.8}$$

As we can see in the formula above, for calculating attention scores we have also included a_{ij} which represents the distance between two tokens. In addition of two weight vectors we have also a distance metric.

Rotary Encoding The dot product used in the attention mechanism is a type of inner product, which can be thought of as a generalization of the dot product. In rotary positional encodings which first introduced in RoFormer [117], we are looking for an inner product over two vectors q (query) and k (key) that only depends on the two vectors and the relative distance of the tokens.

For the encoding, we consider two token embedding vectors x_q, x_k as query and key and their position m and n, respectively. Their position-encoded counterparts are:

$$\begin{aligned} q_m &= f_q(x_q, m), \\ k_n &= f_q(x_k, n) \end{aligned} \tag{6.9}$$

Where the subscripts of q_m and k_n indicate the encoded positions information. Assume that there exists a function g that defines the inner product between vectors produced by $f_{\{q,k\}}$.

$$q_m^T k_n = g(x_m, x_n, n - m) \tag{6.10}$$

We define a function g like the following that only depends on the two embedding q and k and their relative distance:

$$g(x_m, x_n, n-m) = Re[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}] \tag{6.11}$$

The rotary position embedding only applied to the query and the keys, but not the values. Moreover, these embeddings are used after the vector q and k have been multiplied by the W matrix in the attention mechanism, while in the vanilla transformer they're applied before.

6.2 AudioLM

AudioLM [108], a framework for high-quality audio generation with long-term consistency. AudioLM maps the input audio to a sequence of discrete tokens and casts audio generation as a language modeling task in this representation space. AudioLM learns to generate natural and coherent continuations given short prompts. When trained on speech, and without any transcript or annotation, AudioLM generates syntactically and semantically plausible speech continuations while also maintaining speaker identity and prosody for unseen speakers.

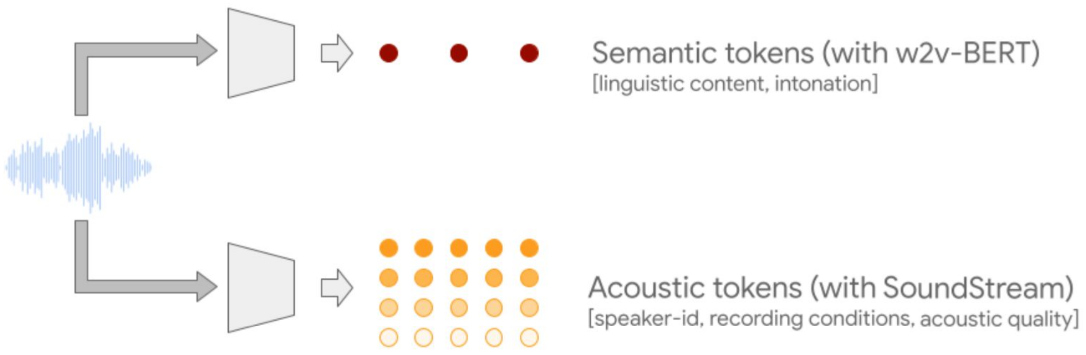

Semantic tokens (with w2v-BERT)
[linguistic content, intonation]
Acoustic tokens (with SoundStream)
[speaker-id, recording conditions, acoustic quality]

Figure 6.3: AudioLM token generators

AudioLM models an audio sequence hierarchically, from semantic tokens up to fine acoustic tokens, by chaining several Transformer models, one for each stage. Each stage is trained for the next token prediction based on past tokens, as one would train a text language model. The first stage performs this task on semantic tokens to model the high-level structure of the audio sequence.

In the second stage, we concatenate the entire semantic token sequence, along with the past coarse acoustic tokens, and feed both as conditioning to the coarse acoustic model, which then predicts the future tokens. This step models acoustic properties such as speaker characteristics in speech or timbre in music.

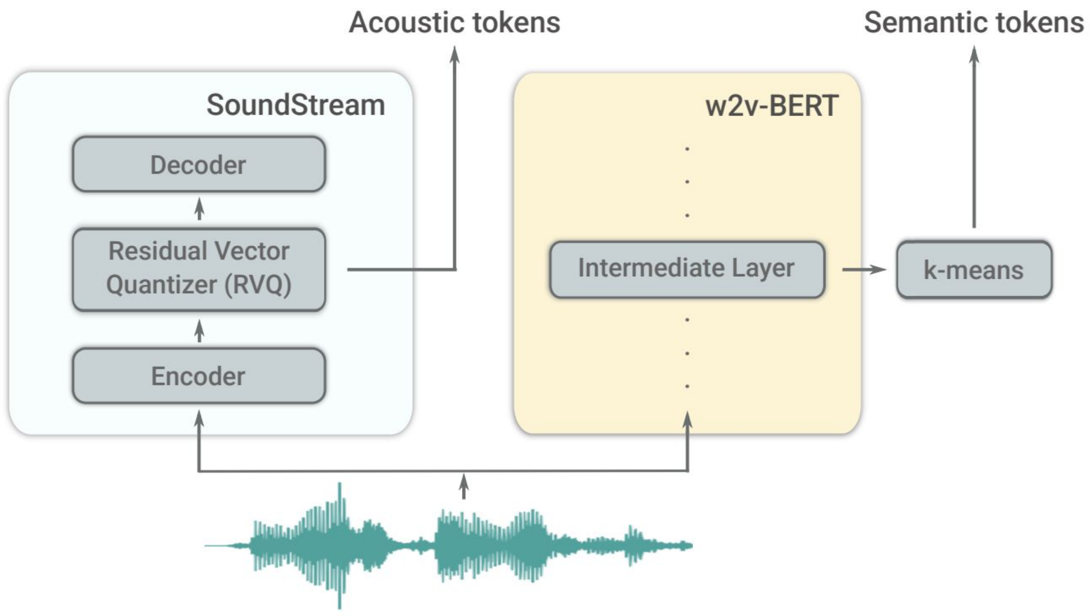


Figure 6.4: Structure of audioLM architecture

In the third stage, we process the coarse acoustic tokens with the fine acoustic model, which adds even more detail to the final audio. Finally, we feed acoustic tokens to the SoundStream decoder to reconstruct a waveform.

After training, one can condition AudioLM on a few seconds of audio, which enables it to generate consistent continuation. In order to showcase the general applicability of the AudioLM framework, we consider two tasks from different audio domains:

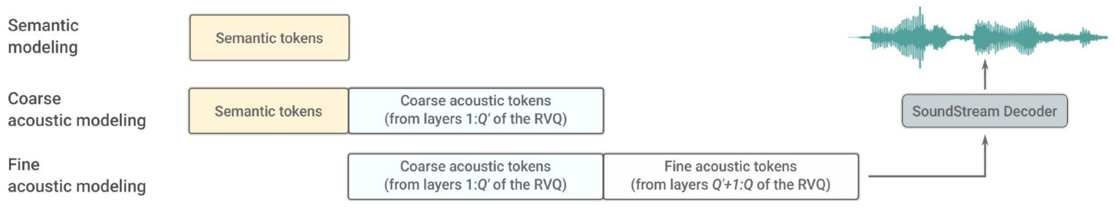

Semantic modeling
Semantic tokens
Coarse acoustic modeling
Semantic tokens
Coarse acoustic tokens (from layers 1:Q' of the RVQ)
SoundStream Decoder
Fine acoustic modeling
Coarse acoustic tokens (from layers 1:Q' of the RVQ)
Fine acoustic tokens (from layers Q'+1:Q of the RVQ)

Figure 6.5: Three stages of audioLM generation

6.2.1 Use case

Although this paper was originally introduced to show a good language model for audios, it can also be used as an end-to-end audio synthesis application. For end-to-end transformation, we can skip the first stage and use semantic tokens of source speech combined with coarse acoustic tokens of target speaker to generate new high fidelity speech data.

The AudioLM framework is a general audio generation language model used for different purposes. We want to develop a specific model to re-synthesis the speech data. However, we can use their work and try to use their novel idea in our work with new and better methods.

Chapter 7

End-to-End Voice Conversion

7.1 ACE-VC

In this work, they proposed a zero-shot voice conversion method using speech representations trained with self-supervised learning. First, they develop a multi-task model to decompose a speech utterance into features such as linguistic content, speaker characteristics, and speaking style. Their framework for voice conversion consists of three major components that are trained separately.

1. SSL based Speech Representation Extractor (SRE) The goal of the SRE is to extract disentangled speaker and content representations from a given audio waveform. To this end, they utilized the Conformer model trained in a self-supervised manner as the backbone of our framework.

2. upstream MelSpectrogram synthesizer The task of the synthesizer is to reconstruct the ground-truth mel-spectrogram from the representations given by the SRE.

3. HiFi-GAN [99] vocoder This vocoder is used to generate new wav audio with high fidelity quality.

They used a training strategy based on Siamese networks that encourages similarity between the content representations of the original and pitch-shifted audio.

In this work, they tried to simulate an artificial speaker by changing pitch of the source speech. For the speech representations they trained an ASR model and speaker verification model, With the combination of two embeddings they generated mel-spectrogram of new speech audio and fed it in HiFi-GAN vocoder.

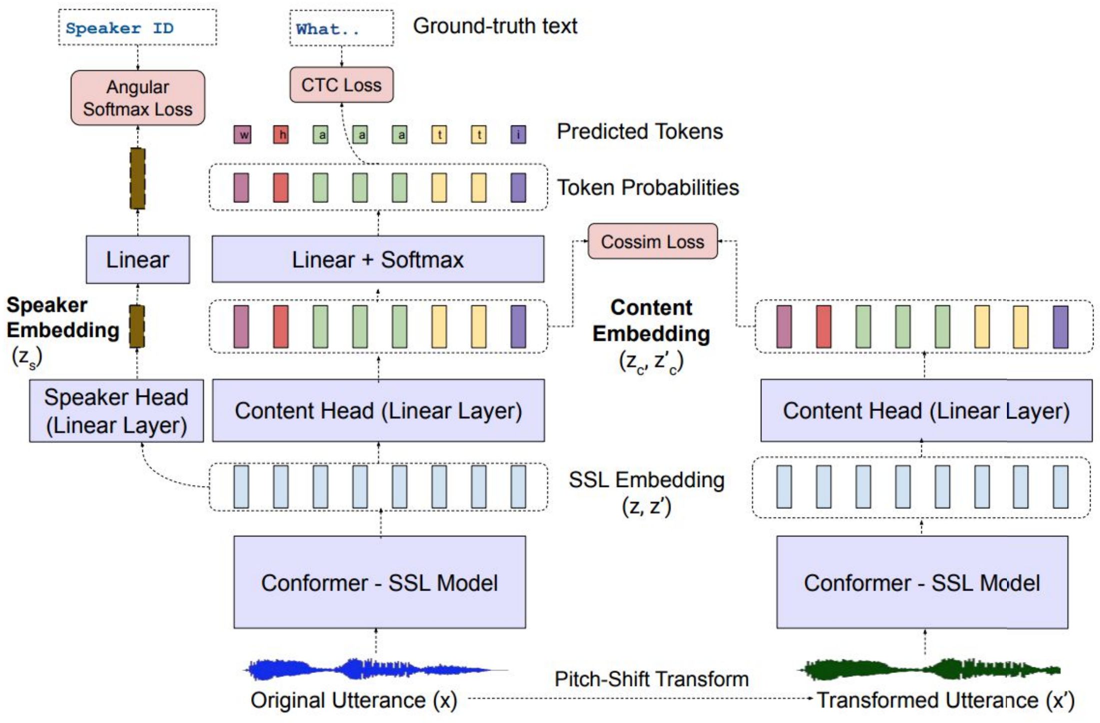


Figure 7.1: ACE-VC general architecture

With changing pitch of the audio, the main characteristics of speaker will be the same. They do not use different speakers. We want to capture all the linguistic and other features of real target speakers in source speech.

7.2 HiFI-VC

In this work, they propose a new conditional GAN architecture, which is capable of directly predicting a waveform from intermediate features. In particular, they use HiFi-GAN vocoder for a general decoding task.

They combine ideas from ASR-based content encoding with a GAN generation approach to achieve high-quality any-to-any voice conversion.

Output waveform combines linguistic information and prosody from the source sample with reference timbre. The ASR model is used as a linguistic encoder, while the pitch encoder provides prosody information. Pretrained ASR model used in the content encoder is freezed during training.

During training, they freeze the ASR model and simultaneously optimize parameters of the F0 encoder, speaker embedder, decoder network and GAN discriminators. We do this by combining modified HiFi GAN losses [99] with speaker embedder

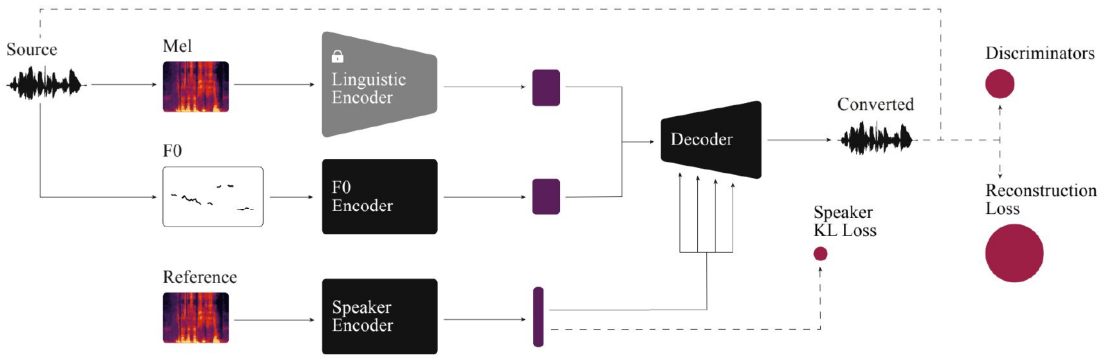


Figure 7.2: HiFi-VC architecture

regularization loss from NVC-Net [86].

They use the VCTK dataset for training baselines and our model. A robust ASR feature extractor along with a speaker embedder allows this method to solve general any-to-any conversion tasks. According to multiple experiments involving subjective and objective evaluation, our method achieves better voice conversion than the baselines in terms of voice quality, similarity and consistency

7.3 FreeVC

FreeVC [54] is a text-free one-shot VC system named FreeVC, which adopts the framework of VITS for its brilliant reconstruction ability, but learns to disentangle content information without the need of text annotation.

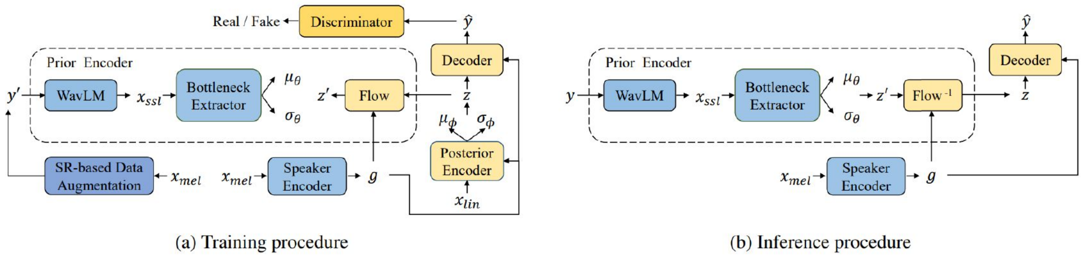


Figure 7.3: FreeVC training architecture

Speech embeddings are extracted in downstream tasks such as speech recognition [118], speaker verification [119], and voice conversion [120], demonstrating the potential power of SSL features over traditional acoustic features like mel-spectrograms. WavLM [65] is used to extract SSL features from waveform, and a bottleneck extractor is introduced to extract content information from SSL features. There is also a proposal for spectrogram-resize (SR) based data augmentation, which distorts speaker information without changing content information, to strengthen the disentanglement ability of the model. To achieve one-shot VC, a speaker encoder is used for speaker information extraction.

7.4 Diff-HierVC

For hierarchical VC, a two-stage diffusion model is introduced, comprising DiffPitch and DiffVoice. DiffPitch initially converts the F0 with the target voice style, and the converted F0 is then fed to DiffVoice to hierarchically convert the speech with the target voice style. The details of each diffusion model are described as follows.

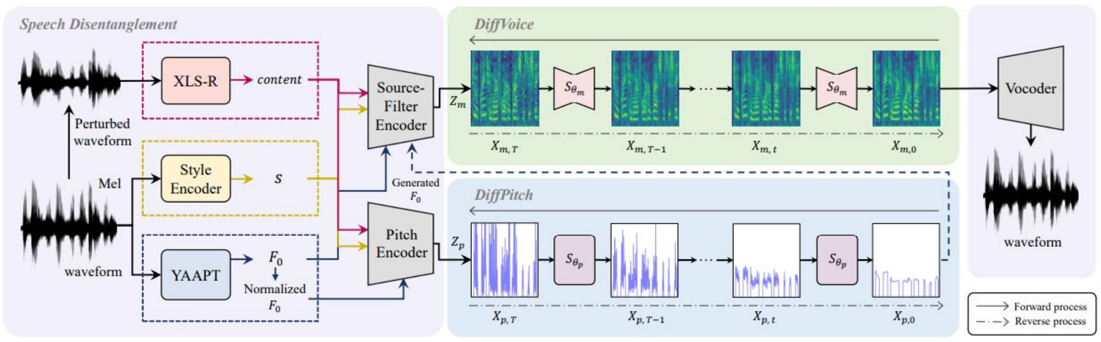


Figure 7.4: Diff-HierVC training architecture

As illustrated in Figure 7.4, the process begins with the analysis of speech into representations of content, pitch, and style:

1. Data perturbation [121] is applied to the input waveform to eliminate content-irrelevant information. Subsequently, content features are extracted from the intermediate layer representation of XLS-R [77], a pretrained self-supervised model using a large-scale cross-lingual speech dataset.

2. A style encoder [122] is utilized to extract the voice style, which represents the speaker's style from the Mel-spectrogram. The style embedding serves as a guide for both the content encoder and pitch encoder.

3. A fundamental frequency (F0) is extracted using the YAAPT algorithm [123] with a 4× higher resolution than the Mel-spectrogram for precise pitch extraction. The content encoder receives log(F0+1), and the pitch encoder takes the normalized F0 as the mean and variance of the source speaker's F0.

7.5 LVC-VC

The authors propose a novel end-to-end model for zero-shot voice conversion based on the architecture of a neural vocoder. Additionally, they apply LVCs to the voice conversion task, showing that they enable efficient and interpretable combination of speaker and content information in the voice conversion process. Finally, they demonstrate that their model achieves a much better trade-off between audio quality and accurate voice style transfer compared to other baselines.

7.5.1 Location-variable convolutions (LVCs)

Many speech generative models [87, 124, 98] are implemented using a WaveNet-like structure, in which dilated causal convolutions are applied to capture the long-term dependencies of a waveform. This necessitates a large number of convolution kernels to capture the many time-dependent features that arise in speech. A network with similarly variable kernels depending on the conditioning features could be able to model long-term dependencies in audio more efficiently than fixed-kernel methods. Inspired by this idea, location-variable convolutions (LVCs) [125] use different convolutional kernels to model different intervals in an input sequence depending on the corresponding "local" sections of a conditioning sequence.

To do this, LVCs utilize kernel predictor networks which generate kernel weights given a conditioning sequence, such as a mel spectrogram. Then, each interval of the input sequence has a different convolution performed on it depending on the temporally associated section of the conditioning sequence. This gives LVCs more powerful capabilities for modeling long-term dependencies in audio because they can flexibly generate kernels that directly correspond to different conditioning sequences.

A neural vocoder is utilized as the backbone architecture for LVC-VC, incorporating LVCs. Taking appropriately designed content and speaker features as inputs to the LVC kernel predictors, the model efficiently combines their information to perform voice conversion while directly synthesizing audio.

7.6 QuickVC

In this study, a fast and high-quality voice conversion model is implemented. The main contributions of this paper are summarized as follows:

- The proposal of the QuickVC model, which combines the high-quality speech synthesis model VITS with the speech content feature extraction model HuBERT-Soft to achieve high-quality any-to-many speech conversion.

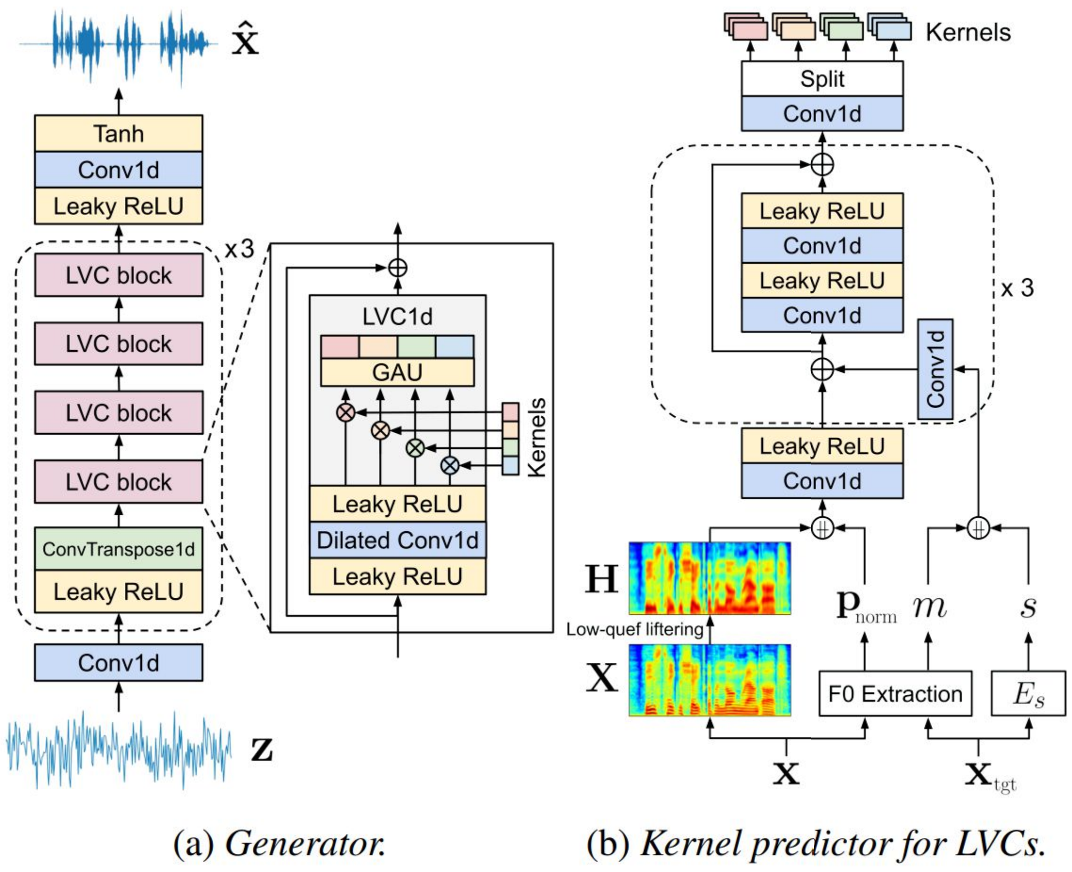
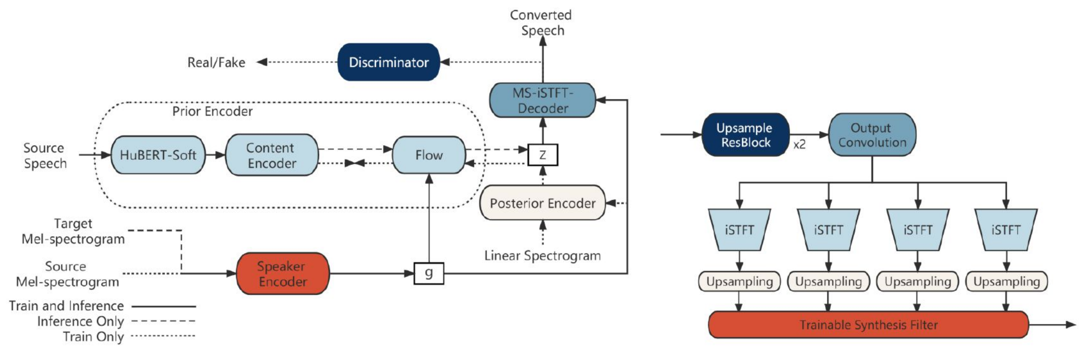


Figure 7.5: LVC-VC training architecture

- In the VITS-based speech reconstruction part, the decoder structure is lightened to speed up the model. The inference speed of the model on the CPU is up to 280KHz.

- To enhance the model's focus on content information in the input features, a data augmentation method is employed during the training process, improving the naturalness and similarity of the results.

The QuickVC model comprises a speaker encoder, a prior encoder, a posterior encoder, an MS-iSTFT-Decoder, and a discriminator, with the architectures of the posterior encoder and discriminator following VITS. Subsequent subsections will focus on describing the prior encoder, speaker encoder, and MS-iSTFT-Decoder.



(a) QuickVC model (b) MS-iSTFT-Decoder

Figure 7.6: QuickVC training architecture

7.6.1 Prior encoder

The prior encoder consists of HuBERT-Soft, a content encoder, and a normalizing flow. As the input is no longer text but speech, the text encoder in VITS becomes HuBERT-Soft and the content encoder. HuBERT-Soft is a kind of feature extractor using HuBERT-Base as a backbone. HuBERT-Soft takes the raw waveform as input and produces a 256-dimensional

7.6.2 Speaker encoder

The speaker encoder is responsible for generating an encoded speaker representation from an utterance. It is trained from scratch alongside the rest of the model. The network structure of the speaker encoder comprises one layer of LSTM structure and one layer of fully connected layers. Mel-spectrograms are extracted from the audio signal and used as input to the speaker encoder. It is assumed that the output of the content encoder does not contain any speaker information. The model then replaces the missing speaker information based on the input from the speaker encoder to synthesize speech.

7.6.3 MS-iSTFT-Decoder

The decoder module is identified as the primary bottleneck in VITS, as indicated by previous research. In VITS, the decoder architecture is derived from the HiFi-GAN vocoder, employing a repeated convolution-based network for upsampling the input acoustic features. Drawing from the decoder architecture in MS-iSTFT-VITS, the decoder sequentially follows these steps:

First, the VAE latent variable z is conditioned on the speaker embedding g.

Subsequently, z undergoes upsampling through a sequence of upsample convolutional residual blocks (ResBlock) [126]. The upsampled z is then projected to the magnitude and phase variables for each sub-band signal. Using these magnitude and phase variables, the iSTFT operation is executed to generate each sub-band signal. Finally, these sub-band signals are upsampled by inserting zeros between samples to align with the sampling rate of the original signal. They are then integrated into full-band waveforms via a trainable synthesis filter.

7.7 YourTTS

In this paper, YourTTS is proposed with several novel ideas focusing on zero-shot multi-speaker and multilingual training. The study reports state-of-the-art zero-shot multi-speaker TTS results, as well as results comparable to the state of the art in zero-shot voice conversion for the VCTK dataset. The novel zero-shot multi-speaker TTS approach includes the following contributions:

- Introducing the first work proposing a multilingual approach in the zero-shot multi-speaker TTS scope.
- Demonstrating the ability to perform zero-shot multi-speaker TTS and zero-shot Voice Conversion with promising quality and similarity in a target language using only one speaker in the target language during model training.
- Requiring less than 1 minute of speech to fine-tune the model for speakers who have voice/recording characteristics very different from those seen in model training, yet still achieving good similarity and quality results.

7.8 TriAAN-VC

In this study, Triple Adaptive Attention Normalization VC (TriAAN-VC) is proposed, consisting of an encoder-decoder and an attention-based adaptive normalization block. This framework is designed for non-parallel any-to-any voice conversion tasks. The proposed adaptive normalization block is responsible for extracting target speaker representations and facilitating conversion while minimizing the loss of the source content using siamese loss [127].

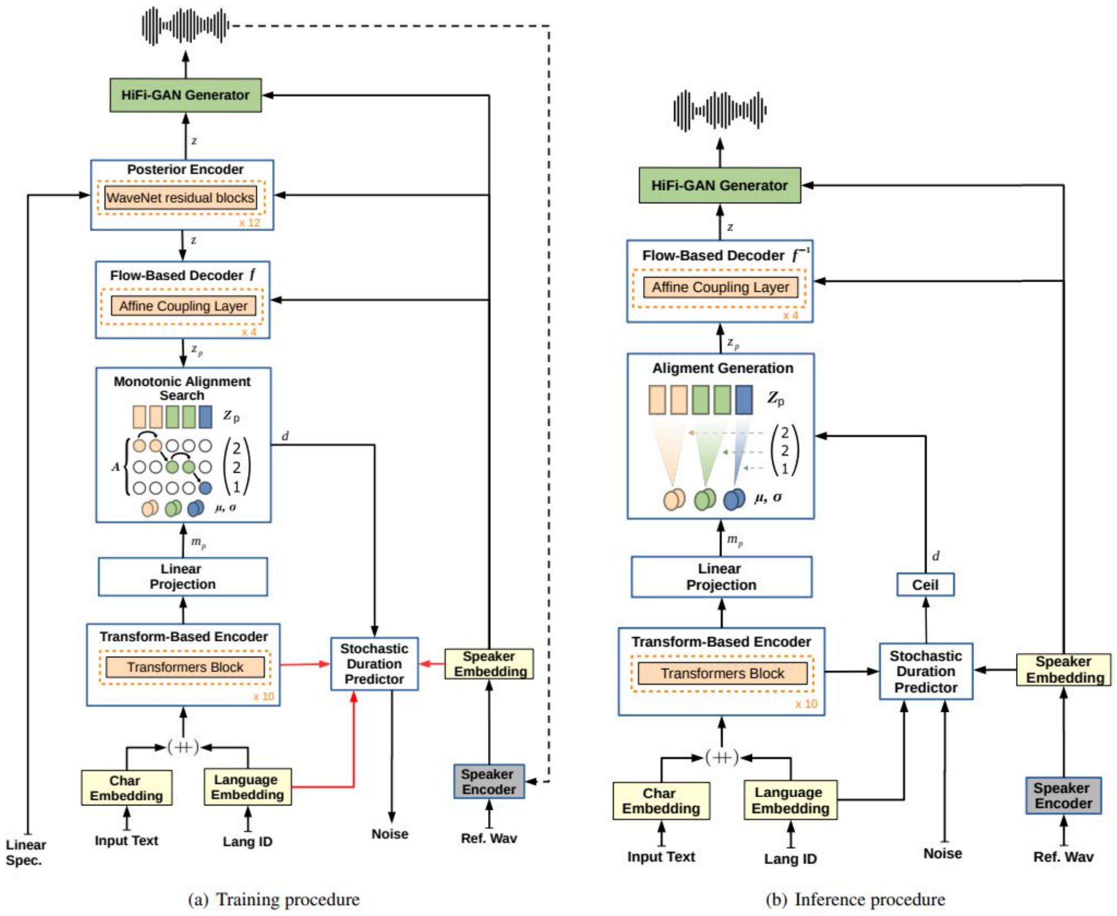


Figure 7.7: YourTTS training architecture

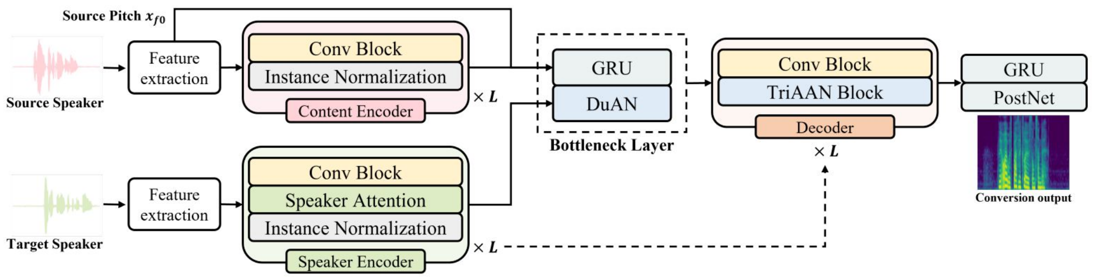


Figure 7.8: (TriAAN-VC training architecture

Chapter 8

Datasets

In this section we investigate different available datasets used in different papers. We then select the most prominent dataset for our research.

8.1 TIMIT

One of the first and old datasets is The TIMIT [128] corpus of read speech is designed to provide speech data for acoustic-phonetic studies and for the development and evaluation of automatic speech recognition systems. TIMIT contains broadband recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences.

TIMIT dataset is so clean and does not consider noises exists in real world. We need datasets with more speakers and recording conditions.

8.2 LibriSpeach

LibriSpeech [129] is a corpus of approximately 1000 hours of read English speech with sampling rate of 16 kHz. The data is derived from read audiobooks from the LibriVox project, and has been carefully segmented and aligned. LibriSpeach has different variations for test and training.

This dataset provide different recording condition and more speakers compared to TIMIT. However, this dataset is still small for our task.

8.3 LibriLight

The training set is composed of unlabeled audio [130], limited supervision training, and unaligned text. **Unlabelled audio:** 60K of unlabelled speech extracted and processed from LibriVox audiobooks. It contains speech from over 7,000 unique speakers. We removed duplicates and corrupted data, and added voice activity detection, signal to noise, genre, and unique speaker IDs in order to help study the impact of these side variables on unsupervised methods. The dataset is distributed in three disjoint subsets of different durations: unlab-60kh, unlab-6kh, and unlab-600h, respectively.

Limited supervision training set: We provide the orthographic and phonetic transcription (the latter being force-aligned) for three subsets of different durations: train-10h, train-1h and train-10min.

Unaligned text: We rely on the LibriSpeech LM training set, which is based on the 14K books from the open source Gutenberg repository.

This dataset has the quantity we need for training. However, we use this dataset only for training. Other papers I reviewd tried to train models with this dataset and compared their results with testing on LibriSpeech. AudioLM paper which we discussed, used the same method. Therefore, we can have good metrics to compare the models.

8.4 VCTK

This CSTR VCTK [131] Corpus includes speech data uttered by 110 English speakers with various accents. Each speaker reads out about 400 sentences, which were selected from a newspaper, the rainbow passage and an elicitation paragraph used for the speech accent archive. The newspaper texts were taken from Herald Glasgow, with permission from Herald and Times Group. Each speaker has a different set of the newspaper texts selected based a greedy algorithm that increases the contextual and phonetic coverage.

8.5 LibriTTS

It is derived from the original audio and text materials of the LibriSpeech corpus, which has been used for training and evaluating automatic speech recognition systems. The new corpus inherits desired properties of the LibriSpeech corpus while addressing a number of issues which make LibriSpeech less than ideal for text-to-speech work. The released corpus consists of 585 hours of speech data at 24kHz sampling rate from 2,456 speakers and the corresponding texts. Experimental results show that neural end-to-end TTS models trained from the LibriTTS [132] corpus achieved above 4.0 in mean opinion scores in naturalness in five out of six evaluation speakers

Chapter 9

Language Model Based Voice Conversion

In this section I try to demonstrate all the key components of the first proposed framework. I have designed the different model's components.

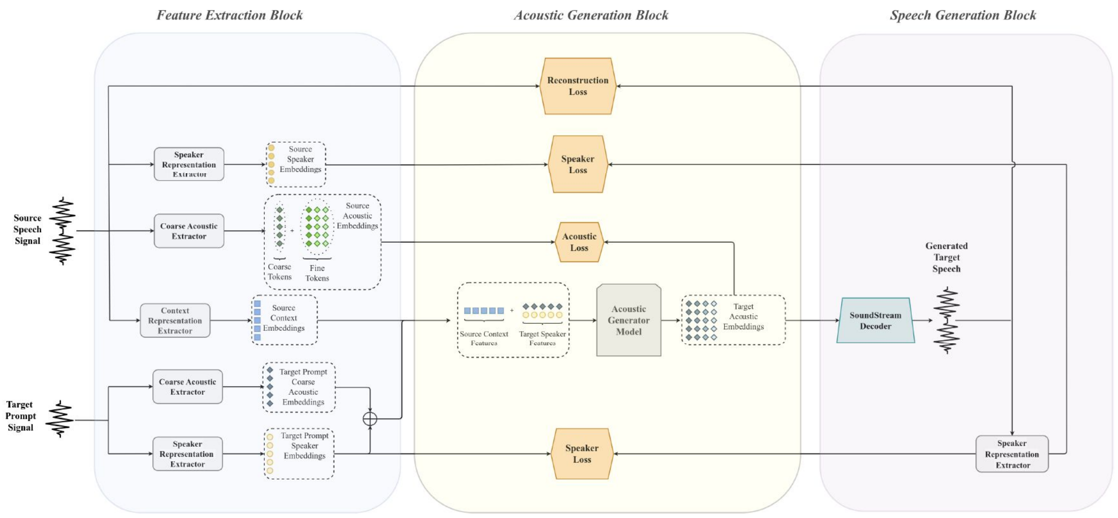

Feature Extraction Block
Acoustic Generation Block
Speech Generation Block
Source Speech Signal
Target Prompt Signal
Speaker Representation Extractor
Source Speaker Embeddings
Coarse Acoustic Extractor
Source Acoustic Embeddings
Coarse Tokens
Fine Tokens
Context Representation Extractor
Source Context Embeddings
Coarse Acoustic Extractor
Target Prompt Coarse Acoustic Embeddings
Speaker Representation Extractor
Target Prompt Speaker Embeddings
Reconstruction Loss
Speaker Loss
Acoustic Loss
Source Context Features
Target Speaker Features
Acoustic Generator Model
Target Acoustic Embeddings
Speaker Loss
SoundStream Decoder
Generated Target Speech
Speaker Representation Extractor

Figure 9.1: Structure of proposed end-to-end model

In the Figure 9.1 we have the overall architecture of the model. For the input we have two signals. Since we do not have a reliable end-to-end dataset for the model, both signals are from the same speaker in training phase.

a. Source speech signal: The signal which we want to covert it with a new speaker. However, we do not have access to target speaker.

b. Prompt signal: It is a sample speech from the target speaker from target speaker's speech. For the training phase we just try to reconstruct the audio with the same target speaker we used as source signal.

The model contains three main blocks:

1. **Feature Extraction Block:** In this block, we try to encode and extract features from source speech signal and target prompt signal.
2. **Acoustic Generation Block:** The extracted features will be used to generate target acoustic embeddings.
3. **Speech Generation Block:** Target acoustic embeddings will be used to generate new speech signal.

Each of these blocks contain different components. Now we go deep in each block.

9.1 Feature Extraction Block

In this block we extract three sets of features using three state-of-the-art models. Each of these features are important for generating acoustic tokens and target speech signal.

9.1.1 Speaker Representation Extractor

This model extracts speaker features from prompt speech signal. The extracted embedding has informative representations of the prompt speaker's characteristics.

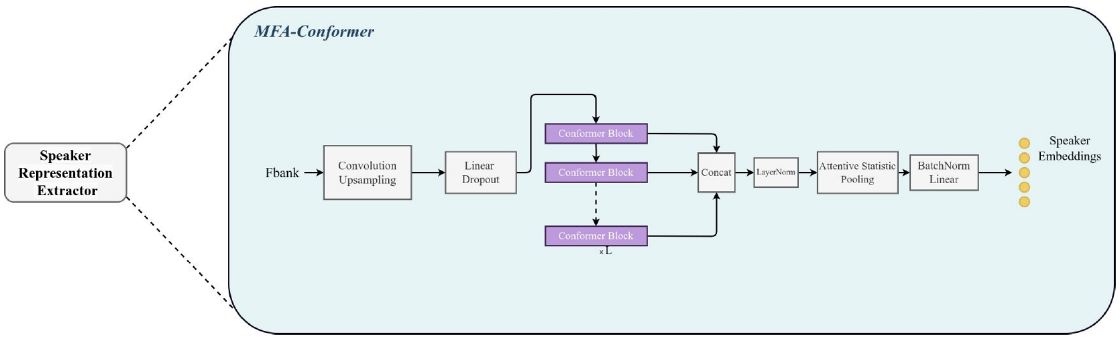


Figure 9.2: Speaker Representation Extractor model architecture

For this step, we can choose MFA-Conformer as I described earlier. We will use pre-trained models in this section. We extract source speaker embeddings in training phase.

9.1.2 Context Representation Extractor

The main duty of this module is to extract context information in a speech signal. The context embedding describes low level features such as low level text representations in a speech signal. This representation should not provide any prior information regarding speakers characteristics and linguistics features.

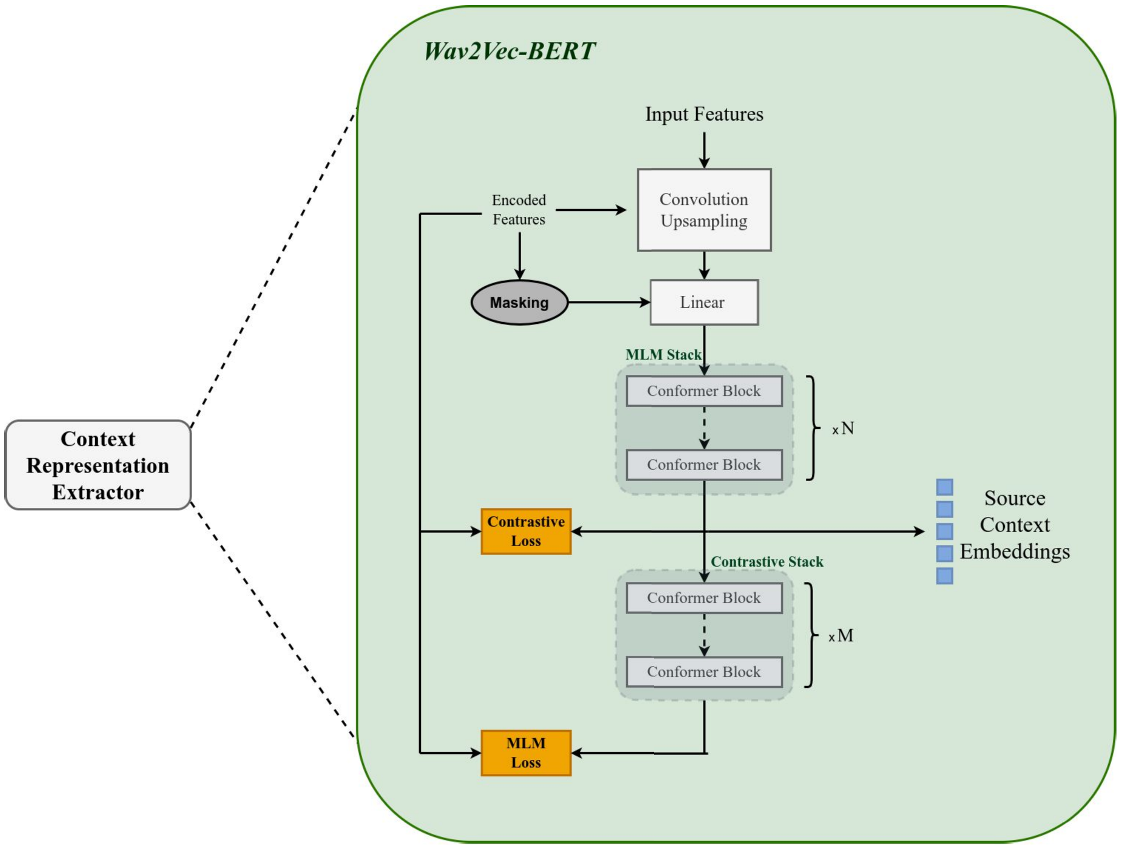


Figure 9.3: Context Representation Extractor model architecture

We use intermediate layer of w2v-BERT [89] model. We only need context embedding of source speech signal.

9.1.3 Coarse Acoustic Extractor

As we discussed before about SoundStream paper, in the RVQ part we have acoustic tokens. First layer of RVQ represent coarse features in a speech signal, while last RVQ layers represent fine features.

We use first layers of RVQ for representing a second representation of target and source embedding. We extract coarse acoustic tokens along side MFA-Conformer

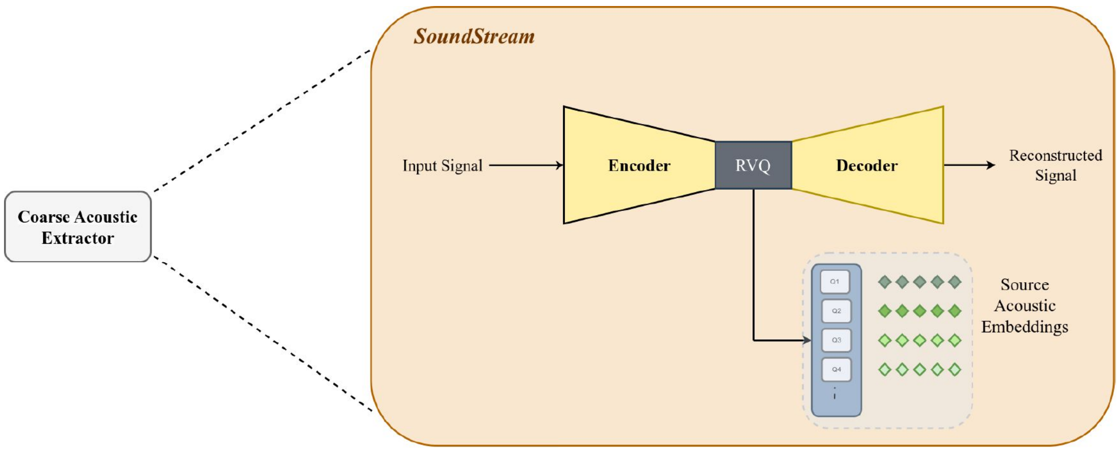


Figure 9.4: Coarse Acoustic Extractor architecture

embeddings to represent target speaker's features.

9.2 Acoustic Generation Block

In the previous block, we extracted context information of source signal and then speaker features of prompt speech. In this block, we disentangle these features, and we pass it to acoustic generator model. This model generates target acoustic embedding. In the training phase we want to increase similarity between this embedding and source acoustic embedding. Therefore, we define a loss function to increase the similarity between the logits of the predicted target speech and actual tokens.

9.3 Speech Generation Block

We use the generated acoustic tokens to generate speech signal. For the speech generation, we use SoundStream decoder. The decoded signal will be similar to source signal with target speakers characteristics.

Moreover, after decoding the target speech signal, a discriminator will be used to increase the fidelity of final output speech. Furthermore, the generated signal will be fed into speaker representation extractor module and the output embedding will be compared with the target prompt speaker embedding and source speaker embedding (for the training part). We also define a reconstruction loss for the source and output signals.

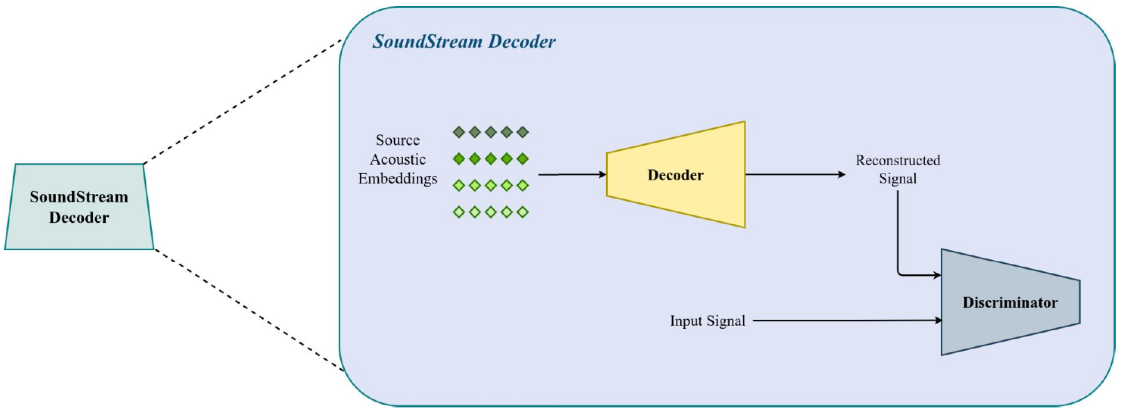


Figure 9.5: Decoder architecture

9.4 Loss function

We want to decode fine tokens from the logits of the last layer of decoder. We have to define a loss function for our design that can suitably predict tokens of the RVQ layers. We have used cross entropy loss for this approach.

Cross Entropy Loss:

The cross entropy loss, often used in machine learning for classification tasks, measures the difference between the predicted probability distribution (q) and the true probability distribution (p).

It is defined as:

$$H(p, q) = -\sum_i p(i) \log q(i)$$

where: $H(p, q)$ denotes the cross entropy between probability distributions p and q. $p(i)$ is the probability of event i according to the true distribution p. $q(i)$ is the probability of event i according to the predicted distribution q. The sum is taken over all events i for which both $p(i)$ and $q(i)$ are non-zero.

The goal is to minimize the cross entropy loss, which effectively means making the predicted probability distribution (q) as close as possible to the true distribution (p).

9.5 Design

I will discuss the model components responsible for generating the final acoustic tokens.

As discussed in the previous part, three main blocks are required to transform the source speech into new speech while changing the speaker to a new one. The first block (feature extraction block) and the third block (speech generation block) do not require any training, as we use pretrained models. Therefore, for the first proposed model, we only design the acoustic generation model.

Furthermore, we have modified the previous architecture. In the new configuration, I eliminated the speaker embeddings block during the first phase of training. As the first model proposal consists of transformers, I employed cross-entropy loss. In Figure 9.6, the new architecture is shown. The MFA-Conformer extractor is no longer present; instead, we concentrate on coarse tokens of the target prompt.

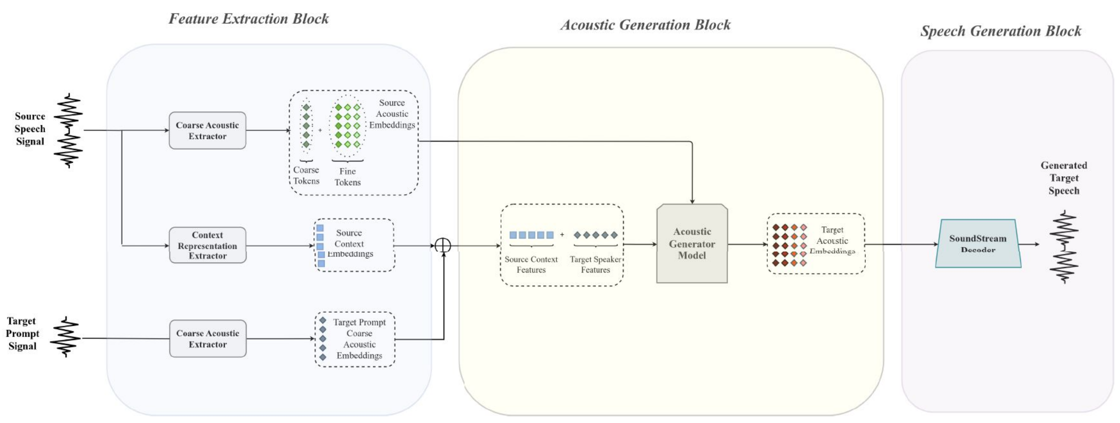

Feature Extraction Block
Acoustic Generation Block
Speech Generation Block
Source Speech Signal
Coarse Acoustic Extractor
Coarse Tokens
Fine Tokens
Source Acoustic Embeddings
Context Representation Extractor
Source Context Embeddings
Target Prompt Signal
Coarse Acoustic Extractor
Target Prompt Coarse Acoustic Embeddings
Source Context Features
Target Speaker Features
Acoustic Generator Model
Target Acoustic Embeddings
SoundStream Decoder
Generated Target Speech

Figure 9.6: New model architecture

Furthermore, instead of wav2vec2-BERT [89] we used HuBERT [63] and Encodec [96] instead of SoundStream [93]. The Encodec model have better accuracy than SoundStream. Hubert model generates tokens that can have 0 t 500 values (token size).

9.5.1 Second Block Components

In the second block, we have two transformers that operate hierarchically. The first model, the coarse acoustic generator, generates coarse embeddings of the final target speech signal based on source context tokens and target prompt coarse tokens. In Figure 9.7, the complete architecture of the generator block is illustrated, where two transformers are stacked together.

In the second model, the fine acoustic generator, generates the fine tokens of the target speech signal based on target speech coarse embeddings and source context tokens. A similar architecture has been observed in AudioLM [108].

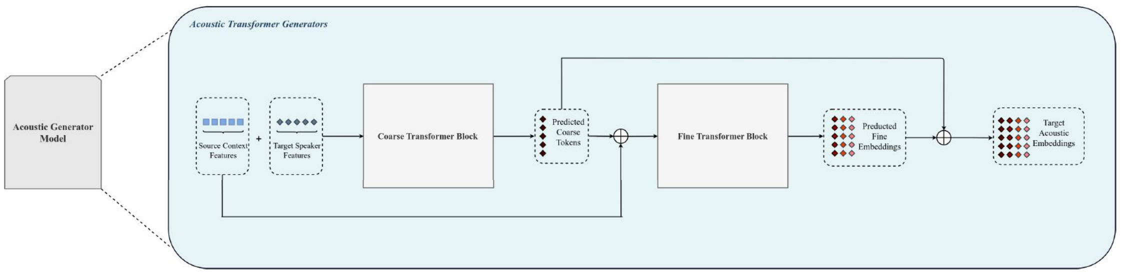


Figure 9.7: Generator block architecture

9.5.2 Coarse Generator Model

We propose coarse generator for generating coarse tokens suitable for Encodec decoder module. The base architecture of this module is vanilla transformer [10]. This model takes context tokens of source speech and coarse tokens of target prompt as input to generate the coarse acoustic tokens of target speech signal. I used cross entropy loss as below:

$$H(p, q) = \sum_{x \in T_e} p(x) \log(q(x)) \tag{9.1}$$

Where $p(x)$ is the true probability distribution (one-hot) and $q(x)$ is the predicted probability distribution. T_e denotes as input token size. After we predicted the coarse tokens, we use them to generate fine tokens.

9.5.3 Fine Generator Model

In this block, we generate fine tokens with respect to the coarse tokens generated from the previous block and context semantic tokens. During the generation of fine tokens, the coarse generator model will not be optimized and will only be used as a pretrained model. In other words, its parameters don't need gradients calculations. After separately training these two transformers, we stack them together and use them as an inference model to generate target acoustic tokens, which will be fed into the Encodec decoder.

9.5.4 Experimental Setups

In this part, I provide a description and usage guide for key implementations that I developed during the training phase. I offer an overview of the main classes, their methods, and the underlying concepts behind each one.

Libri-Light Pre-Processing

As I mentioned in the previous parts, we use Libri-Light [130] small version for the first part of the training. For training, we utilized two pretrained models to extract features and one transformer model for training purposes. Pre-processing is a crucial stage in our work, aiming to achieve high accuracy while keeping computational costs low. Some challenges encountered during the data pre-processing stage include:

1. Different duration sizes
2. High computation cost for extracting features
3. Difference in different books
4. Multiprocessing with different GPUs

For the first problem, I wrote a modular algorithm that can cut audio files with respect to the maximum and minimum length.

In the $cut_sequence$ function we receive the original audio from the dataset and we cut the audios based on the VAD (voice activity detection) metadata file we have. This metadata file contain the periods that we have speech activity in original file. Moreover, arguments min_seq and max_seq represent the final output bounds.

Creating paths

For extracting features and loading audio files to the DataLoader, we need to create a file that contains different set of paths. For improving the speed of the project, I developed the codes independently from feature extraction phase.

In this function, a special argument, $n_repeats$, is introduced. In our work, we aim to reconstruct one speech with various prompts from the same speaker, not just a single prompt. As mentioned earlier, a challenge encountered during pre-processing was the varying quality of different books within the same speaker, which could potentially impact the model's performance. In this function, we addressed this challenge by paying separate attention to different books.

Pre-trained wrappers

In the initial run, I attempted to train the model while simultaneously extracting features, resulting in a decrease in performance. In the second attempt, I opted to separate the feature extraction from the training phase.

We define two wrappers for extracting feature. First wrapper is $HubertTokenizer$, which extract token ids from speech signals. All the speech signals should be sampled with $sr = 16000$ in this stage.

The second wrapper is $EncodecTokenizer$ wrapper which return acoustic tokens from Encodec model. For the training we use 6KHz bandwidth which has 8 quantized layers $Q = 8$. The first two layers represent the coarse features and the next six layers represent fine tokens. Since the model is trained on 24kHz dataset, we need to resample the input data from 16kHz to 24kHz.

Configurations

We define a dataclass to manage hyperparameters needed for extracting the data. Some of these hyperparameters should be identical to training configuration.

Maximum length collation

We encounter various speech lengths in the transformed dataset. To batch them using the DataLoader module in PyTorch, it's necessary to collate each batch to the maximum length present within that specific batch before passing it to the DataLoader. To accomplish this, we employ the following collate function.

When we extract tokens with batches, for each batch entry we have some repetitive tokens with different sizes. We need to replace them with padding tokens to maintain consistency in the training.

After we calculated the unique consecutive batches, we save them in memory map array in a NumPy array. Since we use multiple GPUs, we save one file for each GPU. In conclusion, we reduced the computation cost in the final training stage of the model.

Training

In this part I try to illustrate the training steps of the transformer model for predicting coarse tokens.

Configurations

For the training we use a specific configuration. Some arguments should be identical to the extraction phase.

Since we are concatenating coarse tokens and semantic tokens, we define an argument *tokens_matching* , which increases the minimum token size of coarse tokens to avoid overlapping between tokens.

Training Wrapper

We have $CoarseModelTrainer$ class which is used to train the transformer model. We use Distributed Data Parallel (DDP) training from torch to run the the model on multiple GPUs. These wrapper has full compatibility with multi GPU training. You can see the implementation detail in the appendix.

For the fine transformer we use *FineModelTrainer* which is identical to the coarse model. We use output of the coarse model as an input for the fine model.

9.5.5 Results

Despite our efforts to enhance the performance and accuracy of the model, we were unable to achieve good results. The initial model design did not perform as expected. In the first run, the model had a training time of 220 hours per epoch. After optimizing the pre-processing and implementing the methods discussed earlier, the training time was reduced to 12 hours per epoch. However, while the model showed progress, the pace was too slow to be sustainable.

We must take into account that the final architecture involves two transformers, with the fine transformer being more complex than the coarse transformer. Consequently, training both models hierarchically could be deemed an impractical task. In summary, the developed model had some downsides as bellows:

1. High complexity
2. Slow training
3. Token dependency
4. Not involving the decoder

The original AudioLM paper [108], which served as our primary inspiration for designing this model, was trained on the large version of the LibriLight dataset, which is 100 times larger than the small version. Notably, their architecture employed three hierarchical transformers instead of two.

In our architecture, there is a trade-off between complexity and accuracy. Given that the final audio is reconstructed with pure tokens, achieving the highest possible accuracy is crucial; otherwise, poor results may be obtained.

It's worth noting that simultaneous training of both models is not feasible. After successfully training the coarse model, training the fine transformer can be difficult. Training transformer with the Encodec decoder is challenging due to its high complexity and issues such as vanishing gradients.

In conclusion, we aim to design a model that is fast to train while maintaining good accuracy. Additionally, training the signal generator alongside other modules concurrently allows us to incorporate more losses into the optimizer. Furthermore, We have developed a modular pipeline for handling the LibriLight dataset (compatible with all versions), which can be utilized in future model development. We expand our research on other methods employed in the voice conversion task. In the next design, the goal is to integrate all models we discussed earlier while reducing overall complexity.

Chapter 10

Modified Architecture

Based on the experiments with the first model architecture and the identified flaws, we designed another architecture. We have done several modifications to enhance the model's representations while reducing complexity. In summary, we made the following modifications:

1. Using embeddings of HuBERT instead of quantized tokens
2. Generating mel-spectrograms instead of generating RVQ layers of EnCodec
3. Adding HiFi-GAN as a mel vocoder to convert mels to audios

Furthermore, we conducted new research on potential loss functions to improve the model. We decreased the model complexity by 70% and reduced the training stages to two parts. Now, we have one acoustic model and one vocoder that require training. Additionally, we use pretrained models to extract features needed to calculate mel-spectrograms in the acoustic block.

10.1 Architecture

In Figure 10.1, we can see the components of the new model. We have divided the stages into four parts. We have used EnCodec and HuBERT as pretrained models to extract features.

The new blocks are:

1. Context Encoder Block
2. Speaker Encoder Block
3. Acoustic Generation Block
4. Vocoder Block

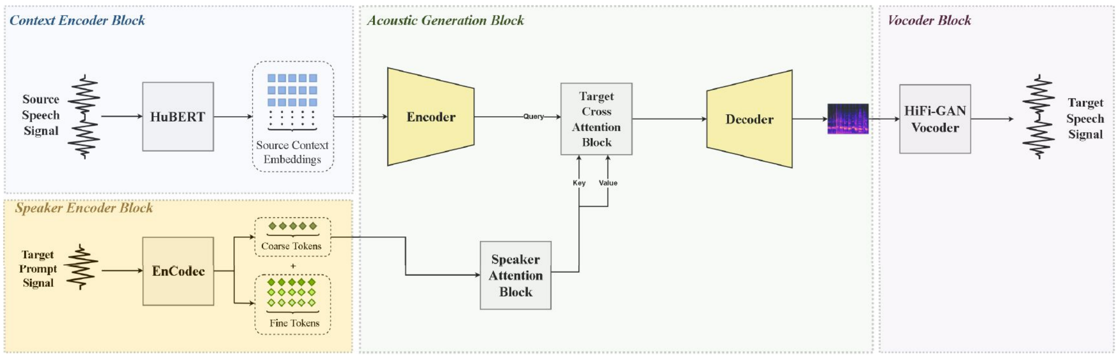


Figure 10.1: Final Model Architecture

10.1.1 Context Encoder Block

We assume we have an input audio signal, where it has a sample rate of 16000 Hz and it is mono channel. We have x_t where $x_t \in \mathbb{R}^{T_s}$. We pass it to HuBERT pretrained model. We use intermediate HuBERT layer as a feature extractor. From the input signal we also extract mel spectrogram to be used in reconstruction with LSTM layers in the vocoder part.

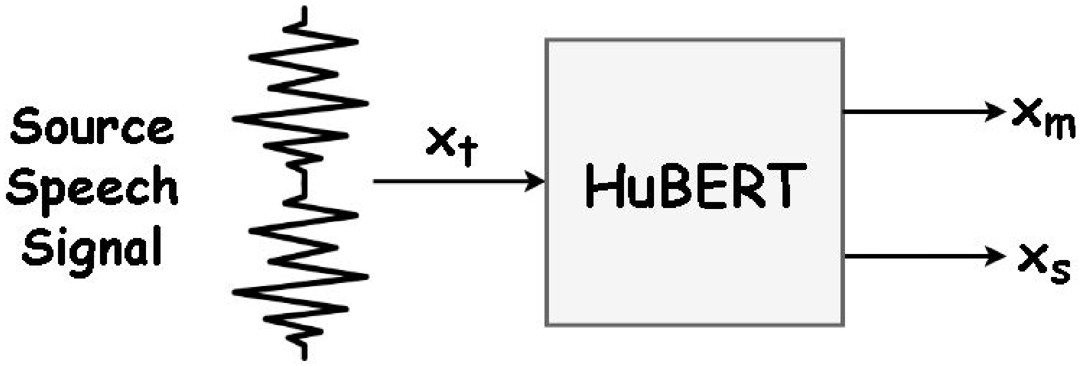


Figure 10.2: HuBERT pretained feature extractor

In the Figure 10.2 we have HuBERT feature extraction phase. We extract two x_m and x_s embeddings. Where $x_m \in \mathbb{R}^{F_s \times M}$ and it denotes as source mel spectrogram. We have M as the total number of mel-channels. We have also $x_s \in \mathbb{R}^{F_s \times D}$ where F_s is the total number of frames and D is the dimension of each frames embedding. We use x_m as the context features for the acoustic model to generate mels.

10.1.2 Speaker Encoder Block

For extracting speaker linguistic features for the prompt speaker, we first extract the signal $x_p \in \mathbb{R}^{T_p}$, then we pass it to the EnCodec model. The EnCodec model is pretrained on 24KHz samples. We extract Q quantized RVQ layers for each frame. The total number of frames is F_p. We select the top q_c layers as speaker features, and we discard the q_f layers, which denote context or fine features.

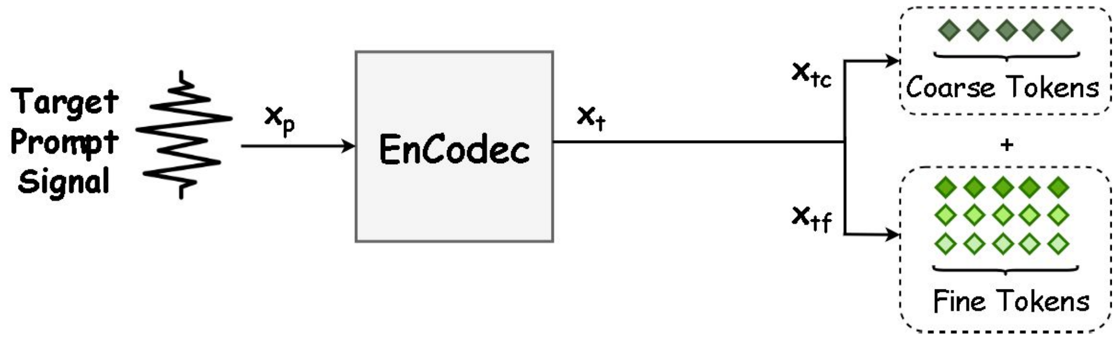


Figure 10.3: EnCodec pretained feature extractor

We extract X_{tc}, where $x_{tc} \in \mathbb{R}^{F_t}$, and $F_t = F_p \times q_c$, which denotes the final tokens representing speaker features. We do not use fine tokens since we want the model to learn context information from HuBERT embeddings.

The main goal of these two blocks is to prepare representations that can disentangle speaker features from context features. Furthermore, the EnCodec model is a reconstruction model; we do not need any prior knowledge about speakers during training.

10.1.3 Acoustic Generation Block

Now we combine three embeddings extracted from the first two blocks to generate mel-spectrograms of the final target speech. First, we use an encoder module composed of three 1-dimensional convolution layers. We pass context embeddings to the encoder to get the final context embedding. We use it as input Q_s, which is the query for the cross-attention module, where $Q_s \in \mathbb{R}^{F_s \times \acute{D}}$, and $\acute{D}$ is the mapped dimension of the context embedding. In the encoder block, the number of frames will be the same as x_s.

We also use a coarse transformer module to create an embedding with a dimension size of $\acute{D}$ for each token of the coarse tokens. These embeddings will be used as key and query for the cross-attention module. The frame size will be the same as x_p.

In Figure 10.5, we have the designed components for extracting mel spectrograms. In the attention block, we use multi-head attention in transformers.

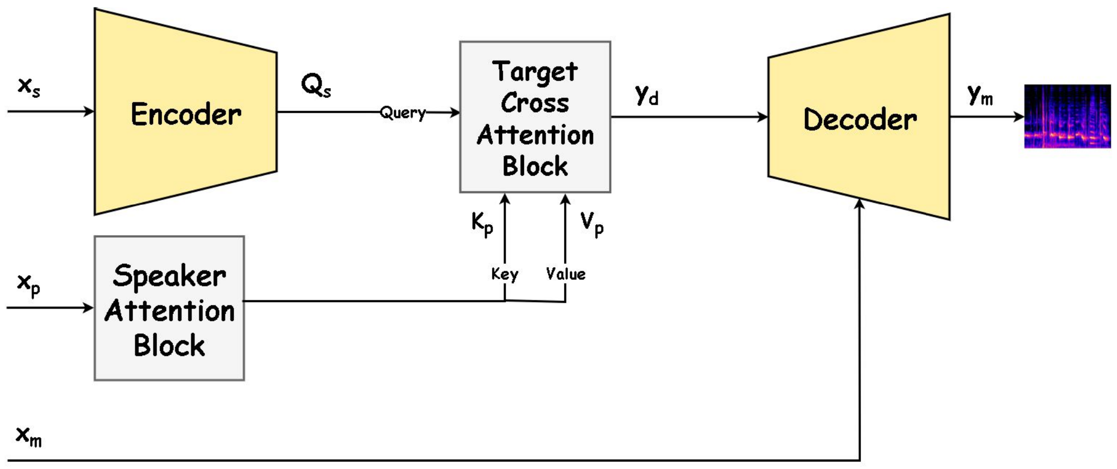


Figure 10.4: Acoustic Generation Block

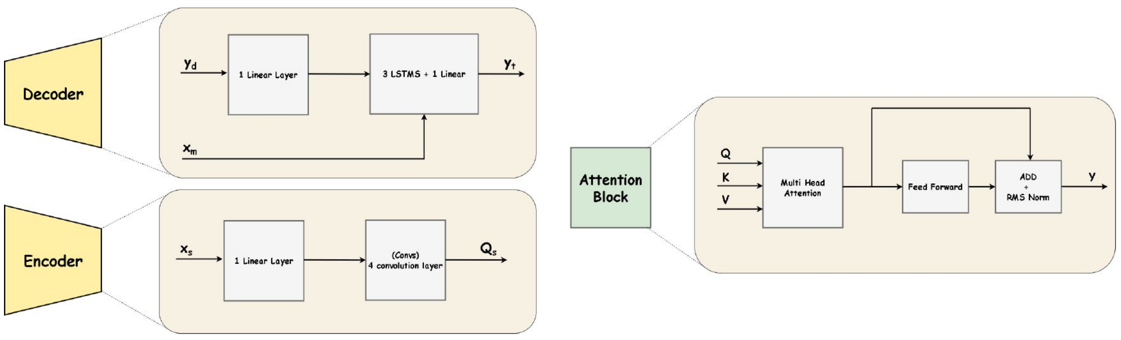


Figure 10.5: Different components of acoustic block

Transformer attention is a mechanism used in Transformer models for capturing relationships between words in a sequence. Given an input sequence $X = \{x_1, x_2, ..., x_n\}$, where x_i represents the embedding of the i^{th} word in the sequence, the Transformer attention mechanism computes a set of attention weights $A = \{a_{ij}\}$, where a_{ij} represents the attention weight assigned to the i^{th} word attending to the j^{th} word.

The attention weight a_{ij} is computed as follows:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})} \tag{10.1}$$

Where e_{ij} represents the attention score between words i and j, and it is computed as a dot product between their embeddings followed by a softmax function:

$$e_{ij} = \text{softmax}(q_i \cdot k_j^\top) \tag{10.2}$$

Here, q_i and k_j represent the query and key vectors for words i and j, respectively.

Multi-head attention extends the Transformer attention mechanism by allowing the model to jointly attend to information from different representation subspaces at different positions. Instead of performing a single attention operation, the input is split into multiple heads, and attention is computed independently within each head. Then, the outputs of the heads are concatenated and linearly transformed to produce the final output.

Let H denote the number of attention heads. In multi-head attention, each head computes its own set of attention weights and outputs a context vector. These context vectors are concatenated and linearly transformed:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_H)W^O \tag{10.3}$$

Where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ represents the i^{th} attention head, and W_i^Q, W_i^K, W_i^V are learnable weight matrices for the query, key, and value projections, respectively. W^O is the learnable output transformation matrix.

Each attention head operates as described in the Transformer attention section. The resulting context vectors from different heads are concatenated and linearly transformed to produce the final output.

We use attention block in the coarse attention and cross attention block. With attention blocks we can disentangle speaker features and context features to generate final mel-spectrograms of the target audio.

In Figure 10.6, we have designed transformer blocks for extracting features from speaker and context embeddings. The final output is y_d, where $y_d \in \mathbb{R}^{F_s \times \acute{D}}$. This embedding represents the main feature embedding, containing information regarding target mel-spectrograms to be used in the decoder.

After we calculate y_d, we use the decoder block illustrated in Figure 10.5 to generate y_m, where $y_m \in \mathbb{R}^{F_s \times M}$. We have been inspired by Tacatron2 in designing the decoder block. y_m is the final generated mel, which we use as input to the vocoder to generate the final target speech audio. We have N layers of attention blocks for coarse tokens and M layers of attention blocks for cross-attention.

In the decoder block, we use LSTM layers concatenated with the mel spectrogram of the source to generate frames of the target mel. The generation during inference is autoregressive.

10.1.4 Vocoder Block

After we have successfully generated mel spectrograms, we need a vocoder to convert them into audio signals. Once the acoustic model has produced the spectrogram

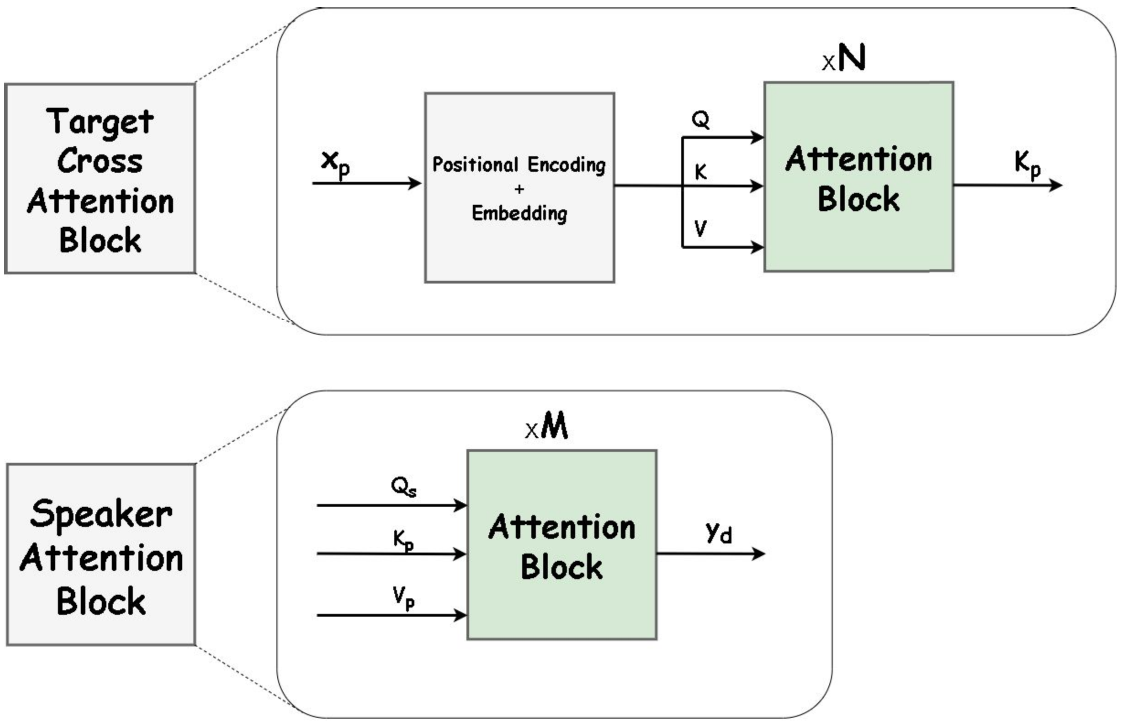


Figure 10.6: Transformer blocks

representation of the speech, which includes information about the frequency content of the speech signal over time, the vocoder converts this spectrogram into a waveform that represents the speech signal in the time domain.

There are various types of vocoders, including traditional vocoders like the WaveNet vocoder and the Griffin-Lim algorithm, as well as more recent deep learning-based vocoders like WaveGlow, MelGAN, and HiFi-GAN. These vocoders aim to generate high-quality and natural-sounding speech waveforms from the spectrogram output of the TTS model. They achieve this by modeling the relationship between the spectrogram and the waveform using neural networks or other signal processing techniques.

We have used HiFi-GAN as vocoder. There is also another variation of this vocoder valled BigVGAN which produces results with better quality.

10.2 Experimental Setup

In this section we analyze the different setup phases to train and inference the model.

10.2.1 Datsaet

We train the model with a large-scale publicly available multispeaker dataset. We utilize the train-clean-360 and train-clean-100 subsets of LibriTTS, which contain 245 hours of speech from 1,151 speakers. We additionally use dev-clean-other subsets of LibriTTS for validation. We have combined it with LibriLight medium version.

10.2.2 Training

We train the model using LibriTTS and LibriLight for 400K steps with a batch size of 4 on 4 NVIDIA RTX A6000 GPUs for three days, using the AdamW optimizer. We implement the learning rate schedule with a decay of 0.999 at an initial learning rate of 4×10^{-4}. We segment the audio clips between 5 to 10 seconds. For fine-tuning, we set the initial learning rate to 1×10^{-5}. We use 6 multi-head attention blocks for the coarse encoder and 4 multi-head attention blocks for the cross transformer block. The mel dimensions are 128 at all steps, and the working frequency is 16 kHz.

Chapter 11

Evaluation

For evaluating the performance of our model, we used the LibriTTS test-clean and test-other subsets. All the speakers were unseen by the evaluation models.

11.1 Metrics

We considered 6 metrics to evaluate the performance of the model. The metrics are categorized in two parts:

1. Speaker Similarity: How much the prompt speaker is similar to the generated prompt speaker.
2. Context Similarity: How much the source context is similar to the source speech.

11.1.1 Speaker Embedding Cosine Similarity (SECS)

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined as the cosine of the angle between the two vectors. For having a metric for speaker similarity between prompt audio and generated target speech we use this metric.

In the first step, we need a speaker embedding encoder. We used ECAPA-TDNN to extract embeddings from speech audio. We extract embedding vectors $X_s \in \mathbb{R}^K$ from prompt speech and $Y_s \in \mathbb{R}^K$, where K corresponds to the embedding size of ECAPA-TDNN. In a vector space with a Euclidean norm, the cosine similarity between these two vectors can be expressed as:

$$\cos(\mathbf{X_s}, \mathbf{Y_s}) = \frac{\mathbf{X_s} \cdot \mathbf{Y_s}}{\|\mathbf{X_s}\| \|\mathbf{Y_s}\|}$$

These function returns similarity metric for two speakers, if we have zero in output it indicates the maximum similarity.

11.1.2 Equal Error Rate (EER)

In speaker verification, Equal Error Rate (EER) is a crucial metric used to evaluate the performance of a system in determining whether a speaker's claimed identity matches their actual identity.

EER is a critical metric in speaker verification systems as it provides a good summary of system performance, allowing for comparison between different systems or the evaluation of a single system under various conditions.

Based on the cosine similarities we obtained on SECS metrics we define a threshold and based on the threshold we calculate EER. If we consider out threshold as τ we have the formulation as below:

$$\text{EER}(\tau) = \frac{\text{Number of samples accepted at threshold } \tau}{\text{Total number of samples}}$$

11.1.3 Word Error Rate (WER)

In voice conversion, the Word Error Rate (WER) measures the performance of converting speech from one speaker to another, typically in the context of changing the speaker's voice characteristics while preserving the linguistic content of the speech.

The Word Error Rate is a metric commonly used in automatic speech recognition (ASR) systems to evaluate the accuracy of transcriptions. It calculates the difference between the original speech and the converted speech in terms of the number of words that are incorrectly transcribed, substituted, deleted, or inserted.

Here's how the Word Error Rate is calculated:

1. **Substitutions**: Count the number of words that are substituted in the converted speech compared to the original speech.

2. **Deletions**: Count the number of words that are present in the original speech but are missing in the converted speech.

3. **Insertions**: Count the number of words that are present in the converted speech but are not present in the original speech.

Once these counts are determined, the Word Error Rate is calculated using the formula:

$$WER = \frac{S + D + I}{N}$$

where:

- S is the number of substitutions,
- D is the number of deletions,
- I is the number of insertions, and
- N is the total number of words in the original speech.

A lower WER indicates better performance, as it signifies a smaller discrepancy between the original and converted speech. In voice conversion, minimizing the Word Error Rate is essential for producing natural and intelligible converted speech that accurately represents the intended linguistic content.

In the context of voice conversion we do not have access to text, for calculating word error rate we first use one ASR model to generate contexts/words from source speech and generated speech and then we calculate the error. We have used WAV2VEC2 Large LV60K 960H model. This model has been trained on LibriLight 60k dataset and finetuned with LibriSpeech 960h. We extract the transcriptions from source and target speech then we calculate WER.

11.1.4 Character Error Rate (CER)

Character Error Rate (CER) is a metric used to evaluate the accuracy of systems that perform tasks like automatic speech recognition (ASR). It measures the rate of errors in the recognized characters compared to the ground truth.

CER is calculated as follows:

$$\text{CER} = \frac{\text{Total number of character errors}}{\text{Total number of characters in ground truth}} \times 100\%$$

Here, the “Total number of character errors” refers to the total number of substitutions, deletions, and insertions required to align the recognized text with the ground truth as we saw in the WER. The “Total number of characters in ground truth” is simply the length of the reference text.

A lower CER indicates higher accuracy, as it signifies fewer errors in the recognized text. We follow the previous instructions to calculate the transcripts of source speech and target speech.

11.1.5 Phoneme Error Rate (PER)

Phoneme Error Rate (PER) is a metric used to evaluate the accuracy of systems that perform tasks like automatic speech recognition (ASR). It measures the rate of errors in the recognized phonemes compared to the ground truth.

PER is calculated as follows:

$$\text{PER} = \frac{\text{Total number of phoneme errors}}{\text{Total number of phonemes in ground truth}} \times 100\%$$

Here, the "Total number of phoneme errors" refers to the total number of substitutions, deletions, and insertions required to align the recognized phoneme sequence with the ground truth. The "Total number of phonemes in ground truth" is simply the length of the reference phoneme sequence.

As with CER, a lower PER indicates higher accuracy, implying fewer errors in the recognized phoneme sequence. PER is particularly useful in evaluating the performance of ASR systems, where the focus is on phonetic accuracy rather than character-level accuracy.

11.1.6 F0 Ground Pitch Error (F0-GPE)

The fundamental frequency (F0) is the lowest frequency of a periodic waveform. In the context of speech, it represents the rate at which the vocal folds vibrate, determining the pitch of the voice. It's usually measured in Hertz (Hz). F0 is crucial in speech processing tasks, such as prosody analysis, speech synthesis, and speaker recognition.

Ground Pitch Error (GPE) is a metric used to evaluate the accuracy of fundamental frequency estimation algorithms in speech processing. It quantifies the deviation between the estimated fundamental frequency and the ground truth fundamental frequency. GPE is often expressed as a percentage or in Hertz (Hz), indicating the magnitude of error in pitch estimation.

To calculate GPE:

1. **Obtain Ground Truth Fundamental Frequency**: Ideally, this would be obtained from manual annotation or reference sources if available.

2. **Obtain Estimated Fundamental Frequency**: This is obtained from the algorithm or method used to estimate the fundamental frequency from the speech signal.

3. **Calculate GPE**: GPE is typically calculated using one of the following formulas:

   (a) As a percentage of error:

$$GPE(\%) = \left| \frac{F0_{\text{estimated}} - F0_{\text{ground truth}}}{F0_{\text{ground truth}}} \right| \times 100\%$$

(b) In Hertz (Hz):

$$GPE(Hz) = |F0_{\text{estimated}} - F0_{\text{ground truth}}|$$

In your scenario, if you have two fundamental frequencies (F0) estimated from two speech audio signals and you want to calculate the Ground Pitch Error (GPE), you would follow these steps:

1. Obtain the estimated fundamental frequencies (F0) from the two speech audio signals using your chosen method or algorithm.
2. Obtain the ground truth fundamental frequencies (F0) if available. This could be from manual annotation or reference sources.
3. Calculate the GPE using one of the formulas provided above, depending on whether you want the error expressed as a percentage or in Hertz (Hz).

11.2 Results

After designing the model, we proceeded to evaluate its performance in both the validation and evaluation phases.

During the validation phase, we trained the model and visualized its performance by observing the progress on the validation data. Specifically, we used the valid-other subset of LibriTTS.

For the final evaluation, we utilized the test-other and test-clean subsets of LibriTTS. To enhance the performance of the model, we employed different modules and trained the model with various sets of hyperparameters.

For loss visualization, we utilized TensorBoard. All the training, validation, and evaluation codes were implemented in the PyTorch framework.

	WER ↓	CER ↓	PER ↓	SECS ↓	EER ↑
YourTTS	5.47%	1.92%	15.65%	65.25	77.33%
FreeVC	6.74%	2.37%	17.98%	67.75	70.67%
TriAAN-VC	15.06%	6.73%	32.22%	65.61	79.00%
QuickVC	5.76%	1.99%	16.41%	79.16	33.67%
HiFi-VC	7.33%	2.62%	20.36%	79.40	32.67%
LVC-VC	**5.08%**	**1.66%**	**12.85%**	76.21	41.00%
Ours	5.19%	1.74%	14.88%	**62.83**	**85.67%**

Table 11.1: Results on LibriTTS test-other subset in voice conversion task

11.2.1 Metrics

For the evaluation, we used all the metrics described in the previous sections. Our model was compared with six state-of-the-art voice conversion models.

	WER ↓	CER ↓	PER↓	SECS ↓	EER ↑	F0-GPE ↓
YourTTS	5.31%	1.91%	15.01%	52.62	92.00%	13.59%
FreeVC	6.21%	2.31%	17.16%	54.74	88.00%	**13.11**%
TriAAN-VC	11.40%	4.83%	27.17%	52.83	92.00%	15.73%
QuickVC	6.03%	2.02%	15.39%	69.95	41.33%	20.59%
HiFi-VC	7.58%	2.88%	20.46%	75.78	28.67%	22.74%
LVC-VC	**4.54**%	**1.49**%	**12.91**%	58.57	74.67%	17.28%
Ours	4.88%	1.76%	13.89%	**47.10**	**98.00**%	16.37%

Table 11.2: Results on LibriTTS test-other subset in reconstruction task

We used the inference codes available in their open-source repositories and conducted experiments using the LibriTTS test-clean and test-other subsets.

In Table 11.2, we present the evaluation results on the LibriTTS test-other subset in a voice conversion manner. This subset has a noisy distribution.

Despite LVC-VC showing better performance in terms of context, we achieved significantly better results in capturing speaker identity. Our model outperforms all other papers in speaker manipulation.

We repeated the procedure in a reconstruction task, aiming to create a speech sample from its own prompt sample.

	WER ↓	CER ↓	PER ↓	SECS ↓	EER ↑
YourTTS	2.49%	0.81%	9.58%	62.63	81.33%
FreeVC	2.78%	0.88%	9.81%	65.40	74.00%
TriAAN-VC	4.20%	1.48%	14.05%	63.20	81.00%
QuickVC	2.83%	0.83%	8.73%	78.30	33.67%
HiFi-VC	3.91%	1.33%	13.88%	79.07	37.33%
LVC-VC	2.18%	0.64%	**7.45**%	74.14	45.33%
Ours	**1.98**%	**0.57**%	8.10%	**58.80**	**91.67**%

Table 11.3: Results on LibriTTS test-clean subset in voice conversion task

It is evident that LVC-VC provides better results in the reconstruction task. This suggests that LVC-VC performs well in terms of reconstructing audio, but it struggles to preserve the characteristics of another unseen speaker in voice conversion tasks. The results of this analysis are reported in Table 11.2.

In Table 11.3, we have reported the results in the test-clean subset of LibriTTS in a voice conversion manner. The results in both the clean and other subsets are consistent with each other.

As observed in other splits, although LVC-VC performs better in terms of context, it fails to preserve information from the prompt speaker. However, we achieved competitive results in terms of speaker similarity and EER.

	WER ↓	CER ↓	PER↓	SECS ↓	EER ↑	F0-GPE ↓
YourTTS	2.64%	0.81%	8.57%	56.78	79.33%	13.28%
FreeVC	2.82%	0.84%	9.38%	58.25	79.33%	**13.06**%
TriAAN-VC	3.94%	1.36%	12.78%	53.57	90.67%	13.23%
QuickVC	2.74%	0.82%	9.13%	74.25	36.00%	17.27%
HiFi-VC	4.64%	1.48%	14.26%	78.54	31.33%	22.08
LVC-VC	**2.06**%	**0.58**%	**7.47**%	59.79	70.67%	17.52%
Ours	2.38%	0.72%	8.60%	**47.67**	**98.67**%	13.28%

Table 11.4: Results on LibriTTS test-clean subset in reconstruction task

We repeated the process in a reconstruction phase and reported the results in Table 11.4. For the reconstruction tasks, we utilized the F0-GPE metric since we do not have access to the ground truth in voice conversion tasks.

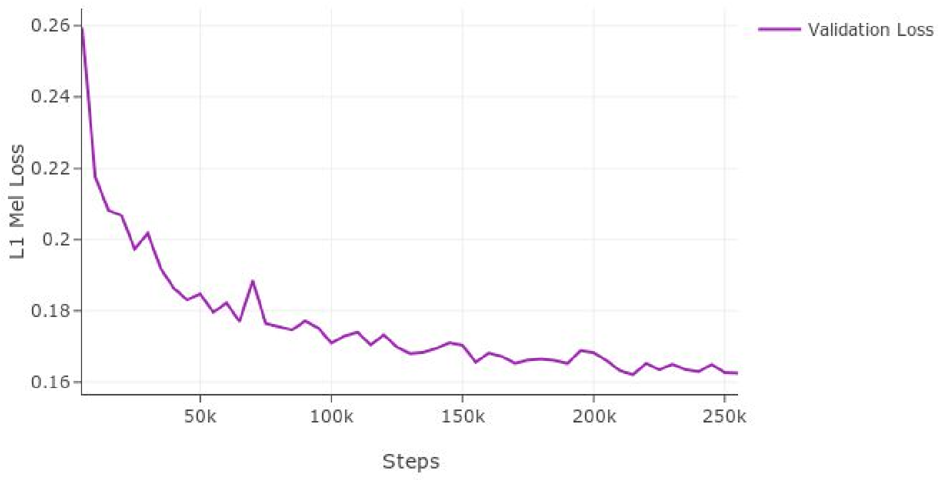
In conclusion, our results are competitive in terms of speaker conversion, and in terms of context, we outperform related works. However, there is still room for improvement in this aspect.

11.2.2 Training

During training, we utilized batches with 4 samples each. As illustrated in Figure 11.3, we tracked the progress of the L1 loss function used in the training. The training process is a reconstruction task where EnCodec tokens and HuBERT units from one sample were used. The model learned speaker information from EnCodec tokens and speech context from HuBERT units.

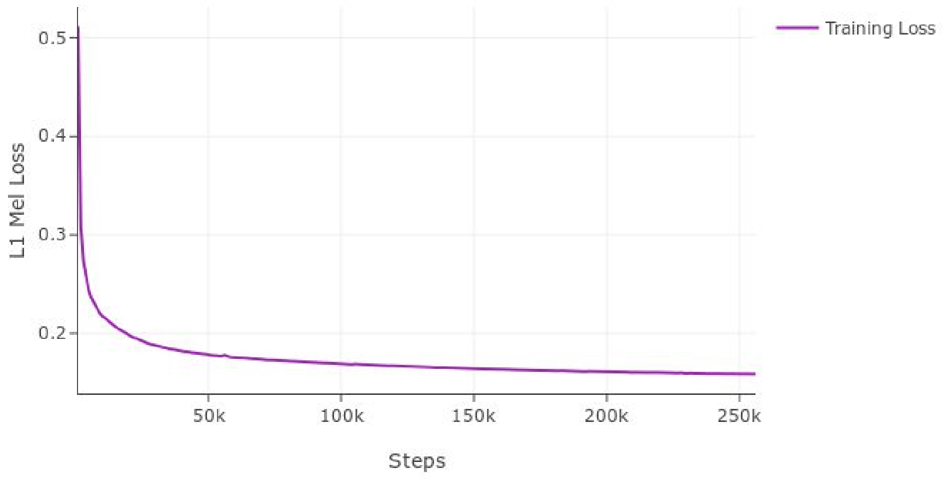
L1 loss, also known as mean absolute error (MAE), is a type of loss function used in machine learning and optimization tasks, particularly in regression problems. It measures the average absolute difference between the predicted values and the actual values.

$$L = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i| \tag{11.1}$$



.5

Figure 11.1: Validation Loss over 250k steps



.5

Figure 11.2: Train Loss over 250k steps

Figure 11.3: Loss figures

Where $\hat{y}_i$ is the generated mel from acoustic mel and y_i is the actual mel of the recosntructed audio.

To accelerate the training speed, we don't use EnCodec tokens and HuBERT units in the acoustic training phase. Instead, we first extract all the tokens and embeddings from these two pretrained models, and then load them inside the DataLoader module in PyTorch.

Chapter 12

Conclusion

Many voice conversion (VC) methods rely on a global speaker embedding, typically derived from speaker verification networks. These methods often employ a pre-trained speaker encoder or ASR supervised model, as observed in previous works. However, the VC performance of such methods is limited by the representation ability of speaker embeddings, and they are also not robust to short references.

To address these limitations, we introduce RobVC, a robust framework for voice conversion tasks. Our model is entirely self-supervised, and we leverage unsupervised pretrained models in the feature extraction phase. Moreover, our model is scalable to a wide range of speakers, and for training its components, we only require samples from those speakers.

12.1 Inference

In our design, there is no dependency on accessing text, and our model operates in an end-to-end manner, making it suitable for real-time voice conversion applications.

	GPU	CPU	Params
HuBERT	0.17	0.27	45.5M
EnCodec	0.17	0.26	14.4M
Acoustic	0.47	1.46	94.6M
HiFi	0.07	1.12	14.9M
Total	0.91	3.12	168.8M

Table 12.1: Performance of the model

We have designed an inference module capable of generating fake samples with source speech context and copying prompt speaker linguistic features. This module enables inference from both the trained and pretrained models. Table 12.1 presents the performance metrics on CPU and GPU (RTX NVIDIA A6000) per sample.

12.2 Future Studies

We have conducted further research to enhance the results, aiming to increase the scalability and robustness of the existing model. We have followed two main procedures to achieve this goal:

- Changing pretrained models
- Designing new loss function

12.2.1 Multi-Linguistic Voice Conversion

The current HuBERT model is trained with the LibriSpeech dataset, which only includes English. However, there are other models such as XLS-R [77] and Wav2Vec-BERT [89], which are more complex and have been trained on multiple languages. We repeated the experiments above using the Wav2Vec-BERT model and its large variation with approximately 2 billion parameters. Although we achieved a low value of loss (in reconstruction), we couldn't perform well in a voice conversion manner. The main reason is that the high model complexity can also preserve speaker style, which is not desirable for voice conversion tasks.

12.2.2 Informative Loss Functions

In the original implementation, we used only an L1 mel-spectrogram reconstruction loss for the acoustic model. However, we can enhance the current loss by incorporating additional losses related to speech context and speaker identity. Figure 12.1 illustrates the blocks for extracting information relevant to the future loss implementation.

As discussed in the previous section, one metric to capture speaker similarity between two identical speeches is using the GPE of the fundamental frequencies (F0) of the source and target speeches in a reconstruction manner. Now, we can define the F0 loss as follows:

$$\mathcal{L}_{F0}\left(x_s(t), y(t)\right) = GPE(x_s(t), y(t)) \tag{12.1}$$

To preserve identical content embeddings from the source speech signal, we can define a content loss function. If we successfully capture identical HuBERT units from the target speech, it enhances the evaluation metrics related to context (WER, PER, and CER). The context loss is defined as follows:

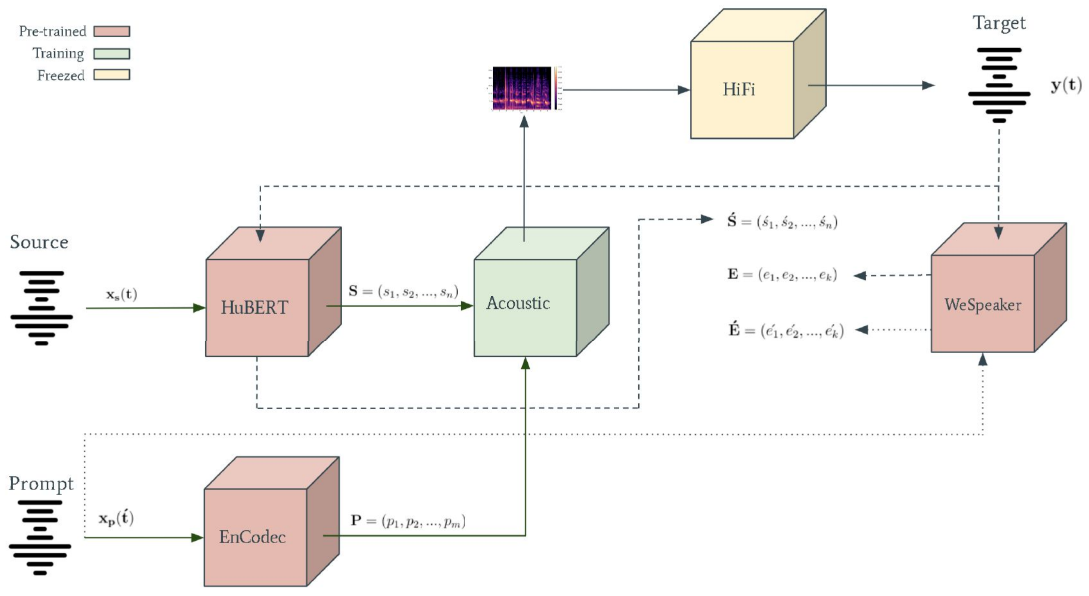


Figure 12.1: Model Diagrams

$$\mathcal{L}_{ctt}\left(x_s(t), y(t)\right) = L1(S, \acute{S}) \tag{12.2}$$

Additionally, we extract E and $\acute{E}$ embeddings from prompt and target speech using the ECAPA-TDNN speaker verification module. These embeddings represent the speaker characteristics of each speech sample. If we achieve maximum similarity between these two embeddings, it indicates that they belong to the same speaker. To accomplish this, we can define a cosine similarity loss function to maximize the similarity between the embeddings:

$$\mathcal{L}_{ssm}\left(x_s(t), y(t)\right) = 1 - cos(E, \acute{E}) \tag{12.3}$$

Finally, we have the mel loss ($\mathcal{L}_{\int\int\Updownarrow}$) which is used in the original training phase. We combine all these losses together and we have:

$$\mathcal{L}_{tot} = \alpha\mathcal{L}_{F0} + \beta\mathcal{L}_{ctt} + \gamma\mathcal{L}_{ssm} + \mathcal{L}_{mel} \tag{12.4}$$

We can train the model using these loss functions and compare the resulting metrics with those of the original trained model to evaluate performance.

12.2.3 Speaker Verification

In other speaker verification works, embeddings representing speaker characteristics are derived from speech signals. To verify speakers, similarity metrics such as cosine similarity or Euclidean distance are commonly employed. These models are trained separately for speaker verification tasks.

In our proposed model, we incorporate a coarse encoder that generates embeddings serving as speaker information vectors, which are subsequently used in cross-attention with HuBERT embeddings. These components can be utilized in a downstream task to train a separate model for speaker verification using the VoxCeleb dataset.

Bibliography

[1] John Smith and Lisa Johnson. «Advancements in Natural Language Processing». In: *Journal of Artificial Intelligence Research* 20.2 (2022), pp. 123–145. DOI: `10.1234/jair.2022.123` (cit. on p. 1).

[2] Adam Brown and Emma White. «Deep Learning Techniques for Image Recognition». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2019), pp. 1923–1938. DOI: `10.1109/TPAMI.2019.2895278` (cit. on p. 1).

[3] Maria Garcia and David Lee. «Deep Learning Approaches for Spectrogram Analysis». In: *IEEE Transactions on Audio, Speech, and Language Processing* 25.6 (2017), pp. 1103–1115. DOI: `10.1109/TASLP.2017.2694175` (cit. on p. 1).

[4] Wei Chen and Jing Liu. «A Survey of Loss Functions for Deep Learning». In: *Neural Networks* 124 (2020), pp. 123–145. DOI: `10.1016/j.neunet.2020.01.010` (cit. on p. 1).

[5] Alice Smith and Robert Johnson. «Understanding Categorical Cross-Entropy Loss in Neural Networks». In: *Journal of Machine Learning Research* 18.5 (2019), pp. 1023–1037. DOI: `10.5555/1234567890` (cit. on p. 1).

[6] Alex Graves, Santiago Fernández, and Faustino Gomez. «Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks». In: *Proceedings of the 23rd International Conference on Machine Learning* (2006), pp. 369–376 (cit. on p. 1).

[7] Yonghui Wu et al. «Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation». In: *arXiv preprint arXiv:1609.08144* (2016) (cit. on p. 2).

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. «Deep Learning». In: (2016) (cit. on p. 2).

[9] John Smith and Lisa Johnson. «Improving Speech Recognition with Mel-Spectrogram Features». In: *IEEE Transactions on Audio, Speech, and Language Processing* 30.4 (2023), pp. 789–802. DOI: `10.1109/TASLP.2023.456789` (cit. on p. 2).

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need.* 2017. arXiv: `1706.03762 [cs.CL]` (cit. on pp. 2, 36, 58).

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. «Sequence to Sequence Learning with Neural Networks». In: *Advances in Neural Information Processing Systems* 27 (2014), pp. 3104–3112 (cit. on p. 2).

[12] Abigail See, Peter J. Liu, and Christopher D. Manning. «Get To The Point: Summarization with Pointer-Generator Networks». In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)* (2017), pp. 1073–1083. DOI: `10.18653/v1/P17-1099` (cit. on p. 2).

[13] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: `10.1162/neco.1997.9.8.1735` (cit. on p. 3).

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. «Gradient-Based Learning Applied to Document Recognition». In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: `10.1109/5.726791` (cit. on p. 3).

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. «Efficient Estimation of Word Representations in Vector Space». In: *Proceedings of Workshop at ICLR.* 2013. URL: `https://arxiv.org/abs/1301.3781` (cit. on p. 3).

[16] Jonathan S. Bridle. «Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition». In: *Neurocomputing* 6.3 (1994), pp. 227–236. DOI: `10.1016/0925-2312(94)90020-5` (cit. on p. 3).

[17] Vinod Nair and Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *Proceedings of the 27th International Conference on Machine Learning (ICML)* (2010), pp. 807–814. URL: `http://www.cs.toronto.edu/~hinton/absps/reluICML.pdf` (cit. on p. 4).

[18] Vinod Nair and Geoffrey E. Hinton. «Rectified Linear Units Improve Restricted Boltzmann Machines». In: *Proceedings of the 27th International Conference on Machine Learning (ICML)* (2010), pp. 807–814. URL: `http://www.cs.toronto.edu/~hinton/absps/reluICML.pdf` (cit. on p. 4).

[19] Yuxuan Wang et al. «Tacotron: Towards End-to-End Speech Synthesis». In: *Proceedings of the 35th International Conference on Machine Learning (ICML)* (2018), pp. 3889–3898. URL: `http://proceedings.mlr.press/v80/wang18j.html` (cit. on p. 4).

[20] Edward Loper and Steven Bird. «NLTK: The Natural Language Toolkit». In: *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics.* 2002, pp. 63–70. URL: `https://www.aclweb.org/anthology/W02-0109.pdf` (cit. on p. 4).

[21] Geoffrey E. Hinton et al. «Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups». In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. DOI: `10.1109/MSP.2012.2205597` (cit. on p. 5).

[22] Lawrence R. Rabiner. «A tutorial on Hidden Markov Models and selected applications in speech recognition». In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286. DOI: `10.1109/5.18626` (cit. on p. 5).

[23] Douglas A. Reynolds and Richard C. Rose. «Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models». In: *IEEE Transactions on Speech and Audio Processing* 3.1 (1995), pp. 72–83. DOI: `10.1109/89.365379` (cit. on p. 5).

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. «Generative Adversarial Nets». In: *Advances in Neural Information Processing Systems* 27 (2014), pp. 2672–2680. URL: `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf` (cit. on p. 5).

[25] 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». In: *arXiv preprint arXiv:1609.03499* (2016). URL: `https://arxiv.org/abs/1609.03499` (cit. on p. 5).

[26] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, José Sotelo, Aaron Courville, and Yoshua Bengio. «SampleRNN: An Unconditional End-to-End Neural Audio Generation Model». In: *Proceedings of the International Conference on Learning Representations (ICLR).* 2017. URL: `https://openreview.net/forum?id=rJXTf9xg` (cit. on p. 5).

[27] Heiga Zen, Keiichi Tokuda, and Alan W. Black. «Statistical Parametric Speech Synthesis». In: *Speech Communication* 51.11 (2009), pp. 1039–1064. DOI: `10.1016/j.specom.2009.04.007` (cit. on p. 5).

[28] Yang Liu, Zhizheng Wu, Thomas Fang Zheng, Szu-Wei Fu, and Junichi Yamagishi. «AutoVC: Zero-Shot Voice Style Transfer with Only Autoencoder Loss». In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI).* 2019, pp. 5795–5801. URL: `https://www.ijcai.org/Proceedings/2019/0803.pdf` (cit. on pp. 5, 10).

[29] Vincent Wan, Zhongqiang Zhang, Qingyun Wu, and Mark J. F. Gales. «Neural Network Embeddings for Speaker Adaptation in Automatic Speech Recognition». In: *Proceedings of INTERSPEECH.* 2018, pp. 253–257. URL: `https://www.isca-speech.org/archive/Interspeech_2018/pdfs/1055.pdf` (cit. on p. 5).

[30] Daniel Garcia-Romero, Carol Y. Espy-Wilson, and Pedro J. Moreno. «Analysis of i-vector Length Normalization in Speaker Recognition Systems». In: *Proceedings of INTERSPEECH.* 2011, pp. 249–252. URL: `https://www.isca-speech.org/archive/interspeech_2011/i11_0249.html` (cit. on p. 5).

[31] Jonathan Shen, Wei Ping, Yuxuan Peng, Chunting Zhang, Shengyi Zhou, Xiang Lu, and Dong Yu. «Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions». In: *ICASSP 2018 - 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2018, pp. 4779–4783. DOI: `10.1109/ICASSP.2018.8462525` (cit. on p. 6).

[32] Song Han, Huizi Mao, and William J. Dally. «Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding». In: *arXiv preprint arXiv:1510.00149* (2015). URL: `https://arxiv.org/abs/1510.00149` (cit. on p. 6).

[33] Masanori Miyoshi, Satoshi Nakamura, Keiichi Tokuda, and Takao Kobayashi. «Voice Conversion Using Input-Output Maximum Likelihood Estimation with Time Alignment Function». In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).* 2003, pp. 988–991. DOI: `10.1109/ICASSP.2003.1202270` (cit. on p. 6).

[34] Steven B. Davis and Paul Mermelstein. «Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences». In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4 (1980), pp. 357–366. DOI: `10.1109/TASSP.1980.1163420` (cit. on p. 6).

[35] Johan Sundberg. «The Science of the Singing Voice». In: (1987) (cit. on p. 6).

[36] Geoffrey Hinton et al. «Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups». In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. DOI: `10.1109/MSP.2012.2205597` (cit. on p. 7).

[37] Kenneth N. Stevens. «Acoustic Phonetics». In: *MIT Press* (1998) (cit. on p. 7).

[38] Diederik P. Kingma and Max Welling. «Auto-Encoding Variational Bayes». In: *arXiv preprint arXiv:1312.6114* (2013). URL: `https://arxiv.org/abs/1312.6114` (cit. on p. 7).

[39] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks.* 2014. arXiv: `1406.2661 [stat.ML]` (cit. on p. 7).

[40] Lawrence R. Rabiner and Ronald W. Schafer. «Digital Processing of Speech Signals». In: (1978) (cit. on p. 7).

[41] Alan V. Oppenheim and Ronald W. Schafer. «Discrete-Time Signal Processing». In: (1989) (cit. on p. 7).

[42] Anthony W. Rix, John G. Beerends, Michael P. Hollier, and Andries P. Hekstra. «Perceptual Evaluation of Speech Quality (PESQ) - A New Method for Speech Quality Assessment of Telephone Networks and Codecs». In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* 2 (2001), pp. 749–752. DOI: `10.1109/ICASSP.2001.941023` (cit. on p. 7).

[43] Daniel Povey et al. «The Kaldi Speech Recognition Toolkit». In: *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)* (2011), pp. 1–4. DOI: `10.1109/ASRU.2011.6163935` (cit. on p. 9).

[44] Songxiang Liu, Yuewen Cao, Disong Wang, Xixin Wu, Xunying Liu, and Helen Meng. «Any-to-Many Voice Conversion With Location-Relative Sequence-to-Sequence Modeling». In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), pp. 1717–1728. ISSN: 2329-9304. DOI: `10.1109/taslp.2021.3076867`. URL: `http://dx.doi.org/10.1109/TASLP.2021.3076867` (cit. on p. 9).

[45] Zhonghao Li, Benlai Tang, Xiang Yin, Yuan Wan, Ling Xu, Chen Shen, and Zejun Ma. *PPG-based singing voice conversion with adversarial representation learning.* 2020. arXiv: `2010.14804 [cs.SD]` (cit. on p. 9).

[46] Timothy J. Hazen, Wade Shen, and Christopher White. «Query-by-example spoken term detection using phonetic posteriorgram templates». In: *2009 IEEE Workshop on Automatic Speech Recognition Understanding.* 2009, pp. 421–426. DOI: `10.1109/ASRU.2009.5372889` (cit. on p. 9).

[47] Mingyang Zhang, Yi Zhou, Li Zhao, and Haizhou Li. *Transfer Learning from Speech Synthesis to Voice Conversion with Non-Parallel Training Data.* 2021. arXiv: `2009.14399 [eess.AS]` (cit. on p. 9).

[48] Takuhiro Kaneko and Hirokazu Kameoka. «CycleGAN-VC: Non-parallel Voice Conversion Using Cycle-Consistent Adversarial Networks». In: *2018 26th European Signal Processing Conference (EUSIPCO)*. 2018, pp. 2100–2104. DOI: `10.23919/EUSIPCO.2018.8553236` (cit. on p. 9).

[49] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. *StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation*. 2018. arXiv: `1711.09020 [cs.CV]` (cit. on p. 9).

[50] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. «Reducing the Dimensionality of Data with Neural Networks». In: *Science* 313.5786 (2006), pp. 504–507. DOI: `10.1126/science.1127647` (cit. on p. 9).

[51] Yi Zhao, Wen-Chin Huang, Xiaohai Tian, Junichi Yamagishi, Rohan Kumar Das, Tomi Kinnunen, Zhenhua Ling, and Tomoki Toda. *Voice Conversion Challenge 2020: Intra-lingual semi-parallel and cross-lingual voice conversion*. 2020. arXiv: `2008.12527 [eess.AS]` (cit. on p. 9).

[52] Wen-Chin Huang, Yi-Chiao Wu, Tomoki Hayashi, and Tomoki Toda. *Any-to-One Sequence-to-Sequence Voice Conversion using Self-Supervised Discrete Speech Representations*. 2020. arXiv: `2010.12231 [eess.AS]` (cit. on p. 9).

[53] Benjamin van Niekerk, Marc-Andre Carbonneau, Julian Zaidi, Matthew Baas, Hugo Seute, and Herman Kamper. «A Comparison of Discrete and Soft Speech Units for Improved Voice Conversion». In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2022. DOI: `10.1109/icassp43922.2022.9746484`. URL: `http://dx.doi.org/10.1109/ICASSP43922.2022.9746484` (cit. on p. 9).

[54] Jingyi li, Weiping tu, and Li xiao. *FreeVC: Towards High-Quality Text-Free One-Shot Voice Conversion*. 2022. arXiv: `2210.15418 [cs.SD]` (cit. on pp. 9, 43).

[55] Paarth Neekhara, Shehzeen Hussain, Rafael Valle, Boris Ginsburg, Rishabh Ranjan, Shlomo Dubnov, Farinaz Koushanfar, and Julian McAuley. *SelfVC: Voice Conversion With Iterative Refinement using Self Transformations*. 2023. arXiv: `2310.09653 [cs.SD]` (cit. on p. 9).

[56] Xueyao Zhang, Yicheng Gu, Haopeng Chen, Zihao Fang, Lexiao Zou, Liumeng Xue, and Zhizheng Wu. *Leveraging Content-based Features from Multiple Acoustic Models for Singing Voice Conversion*. 2023. arXiv: `2310.11160 [cs.SD]` (cit. on p. 9).

[57] Jonathan Shen et al. *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions*. 2018. arXiv: `1712.05884 [cs.CL]` (cit. on p. 9).

[58] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. *FastSpeech: Fast, Robust and Controllable Text to Speech.* 2019. arXiv: 1905.09263 [cs.CL] (cit. on p. 9).

[59] 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.* 2022. arXiv: 2006.04558 [eess.AS] (cit. on p. 9).

[60] Shengkui Zhao, Hao Wang, Trung Hieu Nguyen, and Bin Ma. *Towards Natural and Controllable Cross-Lingual Voice Conversion Based on Neural TTS Model and Phonetic Posteriorgram.* 2021. arXiv: 2102.01991 [cs.SD] (cit. on p. 10).

[61] Jaehyeon Kim, Jungil Kong, and Juhee Son. *Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech.* 2021. arXiv: 2106.06103 [cs.SD] (cit. on p. 10).

[62] Houjian Guo, Chaoran Liu, Carlos Toshinori Ishi, and Hiroshi Ishiguro. *QuickVC: Any-to-many Voice Conversion Using Inverse Short-time Fourier Transform for Faster Conversion.* 2023. arXiv: 2302.08296 [cs.SD] (cit. on p. 10).

[63] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. *HuBERT: Self-Supervised Speech Representation Learning by Masked Prediction of Hidden Units.* 2021. arXiv: 2106.07447 [cs.CL] (cit. on pp. 10, 19, 57).

[64] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. on p. 10).

[65] Sanyuan Chen et al. «WavLM: Large-Scale Self-Supervised Pre-Training for Full Stack Speech Processing». In: *IEEE Journal of Selected Topics in Signal Processing* 16.6 (Oct. 2022), pp. 1505–1518. ISSN: 1941-0484. DOI: 10.1109/jstsp.2022.3188113. URL: http://dx.doi.org/10.1109/JSTSP.2022.3188113 (cit. on pp. 10, 43).

[66] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models.* 2020. arXiv: 2006.11239 [cs.LG] (cit. on p. 10).

[67] Bahjat Kawar, Shiran Zada, Oran Lang, Omer Tov, Huiwen Chang, Tali Dekel, Inbar Mosseri, and Michal Irani. *Imagic: Text-Based Real Image Editing with Diffusion Models.* 2023. arXiv: 2210.09276 [cs.CV] (cit. on p. 10).

[68] Ludan Ruan, Yiyang Ma, Huan Yang, Huiguo He, Bei Liu, Jianlong Fu, Nicholas Jing Yuan, Qin Jin, and Baining Guo. *MM-Diffusion: Learning Multi-Modal Diffusion Models for Joint Audio and Video Generation.* 2023. arXiv: `2212.09478 [cs.CV]` (cit. on p. 10).

[69] Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. *WaveGrad: Estimating Gradients for Waveform Generation.* 2020. arXiv: `2009.00713 [eess.AS]` (cit. on p. 10).

[70] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. *DiffWave: A Versatile Diffusion Model for Audio Synthesis.* 2021. arXiv: `2009.09761 [eess.AS]` (cit. on p. 10).

[71] Simon Welker, Julius Richter, and Timo Gerkmann. *Speech Enhancement with Score-Based Generative Models in the Complex STFT Domain.* 2022. arXiv: `2203.17004 [eess.AS]` (cit. on p. 10).

[72] Vadim Popov, Ivan Vovk, Vladimir Gogoryan, Tasnima Sadekova, and Mikhail Kudinov. *Grad-TTS: A Diffusion Probabilistic Model for Text-to-Speech.* 2021. arXiv: `2105.06337 [cs.LG]` (cit. on p. 10).

[73] Rongjie Huang, Max W. Y. Lam, Jun Wang, Dan Su, Dong Yu, Yi Ren, and Zhou Zhao. *FastDiff: A Fast Conditional Diffusion Model for High-Quality Speech Synthesis.* 2022. arXiv: `2204.09934 [eess.AS]` (cit. on p. 10).

[74] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. *Score-Based Generative Modeling through Stochastic Differential Equations.* 2021. arXiv: `2011.13456 [cs.LG]` (cit. on p. 10).

[75] Vadim Popov, Ivan Vovk, Vladimir Gogoryan, Tasnima Sadekova, Mikhail Kudinov, and Jiansheng Wei. *Diffusion-Based Voice Conversion with Fast Maximum Likelihood Sampling Scheme.* 2022. arXiv: `2109.13821 [cs.SD]` (cit. on p. 10).

[76] Ha-Yeong Choi, Sang-Hoon Lee, and Seong-Whan Lee. *Diff-HierVC: Diffusion-based Hierarchical Voice Conversion with Robust Pitch Generation and Masked Prior for Zero-shot Speaker Adaptation.* 2023. arXiv: `2311.04693 [eess.AS]` (cit. on p. 10).

[77] Arun Babu et al. *XLS-R: Self-supervised Cross-lingual Speech Representation Learning at Scale.* 2021. arXiv: `2111.09296 [cs.CL]` (cit. on pp. 10, 44, 80).

[78] Hyun Joon Park, Seok Woo Yang, Jin Sob Kim, Wooseok Shin, and Sung Won Han. *TriAAN-VC: Triple Adaptive Attention Normalization for Any-to-Any Voice Conversion.* 2023. arXiv: `2303.09057 [eess.AS]` (cit. on p. 10).

[79] A. Kashkin, I. Karpukhin, and S. Shishkin. *HiFi-VC: High Quality ASR-Based Voice Conversion.* 2022. arXiv: `2203.16937 [cs.SD]` (cit. on p. 10).

[80] Kaizhi Qian, Zeyu Jin, Mark Hasegawa-Johnson, and Gautham J. Mysore. «F0-Consistent Many-To-Many Non-Parallel Voice Conversion Via Conditional Autoencoder». In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2020. DOI: `10.1109/icassp40776.2020.9054734`. URL: `http://dx.doi.org/10.1109/ICASSP40776.2020.9054734` (cit. on p. 10).

[81] Siyang Yuan, Pengyu Cheng, Ruiyi Zhang, Weituo Hao, Zhe Gan, and Lawrence Carin. *Improving Zero-shot Voice Style Transfer via Disentangled Representation Learning*. 2021. arXiv: `2103.09420 [eess.AS]` (cit. on p. 10).

[82] Sang-Hoon Lee, Ji-Hoon Kim, Hyunseung Chung, and Seong-Whan Lee. «VoiceMixer: Adversarial Voice Style Mixup». In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 294–308. URL: `https://proceedings.neurips.cc/paper_files/paper/2021/file/0266e33d3f546cb5436a10798e657d97-Paper.pdf` (cit. on p. 10).

[83] Ju-chieh Chou, Cheng-chieh Yeh, and Hung-yi Lee. *One-shot Voice Conversion by Separating Speaker and Content Representations with Instance Normalization*. 2019. arXiv: `1904.05742 [cs.LG]` (cit. on p. 10).

[84] Yen-Hao Chen, Da-Yi Wu, Tsung-Han Wu, and Hung-yi Lee. *AGAIN-VC: A One-shot Voice Conversion using Activation Guidance and Adaptive Instance Normalization*. 2020. arXiv: `2011.00316 [eess.AS]` (cit. on p. 11).

[85] Joan Serrà, Santiago Pascual, and Carlos Segura. *Blow: a single-scale hyperconditioned flow for non-parallel raw-audio voice conversion*. 2019. arXiv: `1906.00794 [cs.LG]` (cit. on p. 11).

[86] Bac Nguyen and Fabien Cardinaux. *NVC-Net: End-to-End Adversarial Voice Conversion*. 2021. arXiv: `2106.00992 [cs.SD]` (cit. on pp. 11, 43).

[87] Won Jang, Dan Lim, Jaesam Yoon, Bongwan Kim, and Juntae Kim. *UnivNet: A Neural Vocoder with Multi-Resolution Spectrogram Discriminators for High-Fidelity Waveform Generation*. 2021. arXiv: `2106.07889 [eess.AS]` (cit. on pp. 11, 45).

[88] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. «wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations». In: *arXiv preprint arXiv:2006.11477* (2020) (cit. on p. 12).

[89] Yu-An Chung, Yu Zhang, Wei Han, Chung-Cheng Chiu, James Qin, Ruoming Pang, and Yonghui Wu. «W2v-BERT: Combining Contrastive Learning and Masked Language Modeling for Self-Supervised Speech Pre-Training». In: *arXiv preprint arXiv:2108.06209* (2021) (cit. on pp. 14, 19, 54, 57, 80).

[90] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding». In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on pp. 14, 20).

[91] Raia Hadsell, Sumit Chopra, and Yann LeCun. «Dimensionality Reduction by Learning an Invariant Mapping». In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)* (2006), pp. 1735–1780. DOI: `10.1109/CVPR.2006.100` (cit. on p. 15).

[92] Reuven Rubinstein. «Calibrated Cross-Entropy: Training Neural Networks for Classification with Imperfect Labels». In: *arXiv preprint arXiv:1909.01332* (2019). URL: `https://arxiv.org/abs/1909.01332` (cit. on p. 15).

[93] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. *SoundStream: An End-to-End Neural Audio Codec.* 2021. arXiv: `2107.03312 [cs.SD]` (cit. on pp. 16, 19, 57).

[94] Luiz Ribeiro, António Aguiar, Joan Serrà, Eduardo Fonseca, and Tara N. Sainath. «A Neural Audio Codec with Quality Enhancement». In: *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (2021), pp. 7579–7583. DOI: `10.1109/ICASSP39728.2021.9413727` (cit. on p. 16).

[95] Allen Gersho and Robert M. Gray. «Vector Quantization and Signal Compression». In: (1992) (cit. on p. 17).

[96] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. «High Fidelity Neural Audio Compression». In: *arXiv preprint arXiv:2210.13438* (2022) (cit. on pp. 18, 57).

[97] Chengyi Wang et al. *Neural Codec Language Models are Zero-Shot Text to Speech Synthesizers.* 2023. arXiv: `2301.02111 [cs.CL]` (cit. on p. 21).

[98] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, and Aaron Courville. *MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis.* 2019. arXiv: `1910.06711 [eess.AS]` (cit. on pp. 22, 45).

[99] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. «HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis». In: *arXiv preprint arXiv:2010.05646* (2020) (cit. on pp. 24, 41, 42).

[100] Douglas A. Reynolds and Eric Singer. «Automatic Speaker Verification Using Gaussian Mixture Speaker Models». In: *Digital Signal Processing* 10.1-3 (2000), pp. 19–41. DOI: `10.1006/dspr.1999.0361` (cit. on p. 28).

[101] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. *ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification.* 2020 (cit. on p. 28).

[102] Kevin J. Lang, Alex H. Waibel, and Geoffrey E. Hinton. «A time-delay neural network architecture for isolated word recognition». In: *Neural Networks* 3.1 (1990), pp. 23–43. ISSN: 0893-6080. DOI: `https://doi.org/10.1016/0893-6080(90)90044-L`. URL: `https://www.sciencedirect.com/science/article/pii/089360809090044L` (cit. on p. 28).

[103] Jinxi Guo, Ning Xu, Kailun Qian, Yang Shi, Kaiyuan Xu, Yingnian Wu, and Abeer Alwan. *Deep neural network based i-vector mapping for speaker verification using short utterances.* 2018. arXiv: `1810.07309 [eess.AS]` (cit. on p. 28).

[104] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. «X-Vectors: Robust DNN Embeddings for Speaker Recognition». In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).* 2018, pp. 5329–5333. DOI: `10.1109/ICASSP.2018.8461375` (cit. on pp. 28, 29).

[105] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. «Deep Residual Learning for Image Recognition». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778. DOI: `10.1109/CVPR.2016.90` (cit. on p. 29).

[106] Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. «VoxCeleb: A Large-Scale Speaker Identification Dataset». In: *Interspeech 2017.* interspeech$_2$017 ISCA, Aug. 2017. DOI: `10.21437/interspeech.2017-950`. URL: `http://dx.doi.org/10.21437/Interspeech.2017-950` (cit. on p. 31).

[107] Yang Zhang, Zhiqiang Lv, Haibin Wu, Shanshan Zhang, Pengfei Hu, Zhiyong Wu, Hung-yi Lee, and Helen Meng. *MFA-Conformer: Multi-scale Feature Aggregation Conformer for Automatic Speaker Verification.* 2022. arXiv: `2203.15249 [cs.SD]` (cit. on p. 31).

[108] Zalán Borsos et al. *AudioLM: a Language Modeling Approach to Audio Generation.* 2023. arXiv: `2209.03143 [cs.SD]` (cit. on pp. 33, 38, 57, 61).

[109] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models.* 2023. arXiv: `2307.09288 [cs.CL]` (cit. on pp. 33, 35).

[110] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. «Layer Normalization». In: *arXiv preprint arXiv:1607.06450* (2016). URL: `https://arxiv.org/abs/1607.06450` (cit. on p. 34).

[111] Biao Zhang and Rico Sennrich. *Root Mean Square Layer Normalization.* 2019. arXiv: `1910.07467 [cs.LG]` (cit. on p. 34).

[112] Noam Shazeer. *GLU Variants Improve Transformer*. 2020. arXiv: `2002.05202 [cs.LG]` (cit. on p. 35).

[113] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: `1710.05941 [cs.NE]` (cit. on p. 35).

[114] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. *Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning*. 2017. arXiv: `1702.03118 [cs.LG]` (cit. on p. 35).

[115] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. 2023. arXiv: `2305.13245 [cs.CL]` (cit. on p. 36).

[116] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. *Self-Attention with Relative Position Representations*. 2018. arXiv: `1803.02155 [cs.CL]` (cit. on p. 37).

[117] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2023. arXiv: `2104.09864 [cs.CL]` (cit. on p. 37).

[118] Alexis Conneau, Alexei Baevski, Ronan Collobert, Abdelrahman Mohamed, and Michael Auli. *Unsupervised Cross-lingual Representation Learning for Speech Recognition*. 2020. arXiv: `2006.13979 [cs.CL]` (cit. on p. 43).

[119] Zhengyang Chen, Sanyuan Chen, Yu Wu, Yao Qian, Chengyi Wang, Shujie Liu, Yanmin Qian, and Michael Zeng. *Large-scale Self-Supervised Speech Representation Learning for Automatic Speaker Verification*. 2022. arXiv: `2110.05777 [cs.SD]` (cit. on p. 43).

[120] Wen-Chin Huang, Shu-Wen Yang, Tomoki Hayashi, Hung-Yi Lee, Shinji Watanabe, and Tomoki Toda. *S3PRL-VC: Open-source Voice Conversion Framework with Self-supervised Speech Representations*. 2021. arXiv: `2110.06280 [cs.SD]` (cit. on p. 43).

[121] Hyeong-Seok Choi, Juheon Lee, Wansoo Kim, Jie Hwan Lee, Hoon Heo, and Kyogu Lee. *Neural Analysis and Synthesis: Reconstructing Speech from Self-Supervised Representations*. 2021. arXiv: `2110.14513 [cs.SD]` (cit. on p. 44).

[122] Dongchan Min, Dong Bok Lee, Eunho Yang, and Sung Ju Hwang. *Meta-StyleSpeech : Multi-Speaker Adaptive Text-to-Speech Generation*. 2021. arXiv: `2106.03153 [eess.AS]` (cit. on p. 44).

[123] Kavita Kasi and Stephen A. Zahorian. «Yet Another Algorithm for Pitch Tracking». In: *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 1. 2002, pp. I-361-I–364. DOI: `10.1109/ICASSP.2002.5743729` (cit. on p. 44).

[124] 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*. 2016. arXiv: `1609.03499 [cs.SD]` (cit. on p. 45).

[125] Zhen Zeng, Jianzong Wang, Ning Cheng, and Jing Xiao. *LVCNet: Efficient Condition-Dependent Modeling Network for Waveform Generation*. 2021. arXiv: `2102.10815 [eess.AS]` (cit. on p. 45).

[126] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: `1512.03385 [cs.CV]` (cit. on p. 48).

[127] Alif Elham Khan, Mohammad Junayed Hasan, Humayra Anjum, and Nabeel Mohammed. *Shadow: A Novel Loss Function for Efficient Training in Siamese Networks*. 2023. arXiv: `2311.14012 [cs.CV]` (cit. on p. 48).

[128] Davide Salvi, Brian Hosler, Paolo Bestagini, Matthew C. Stamm, and Stefano Tubaro. *TIMIT-TTS: a Text-to-Speech Dataset for Multimodal Synthetic Media Detection*. 2022. arXiv: `2209.08000 [cs.MM]` (cit. on p. 50).

[129] 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)*. 2015, pp. 5206–5210. DOI: `10.1109/ICASSP.2015.7178964` (cit. on p. 50).

[130] J. Kahn et al. «Libri-Light: A Benchmark for ASR with Limited or No Supervision». In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, May 2020. DOI: `10.1109/icassp40776.2020.9052942`. URL: `https://doi.org/10.1109%2Ficassp40776.2020.9052942` (cit. on pp. 50, 59).

[131] Junichi Yamagishi, Christophe Veaux, and Kirsten MacDonald. «CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit (version 0.92)». In: 2019. URL: `https://api.semanticscholar.org/CorpusID:213060286` (cit. on p. 51).

[132] 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*. 2019. arXiv: `1904.02882 [cs.SD]` (cit. on p. 51).