BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

TEXT-TO-SPEECH PERSONALIZATION

PERSONALIZACE SYSTÉMŮ SYNTÉZY HLASU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MICHAL LUNER**
AUTOR PRÁCE

SUPERVISOR **Ing. JAN BRUKNER**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

145045

Institut:	Department of Computer Graphics and Multimedia (UPGM)
Student:	**Luner Michal**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**Text-to-Speech Personalization**
Category:	Speech and Natural Language Processing
Academic year:	2022/23

Assignment:

1. Get acquainted with current text-to-speech systems and available Czech datasets.
2. Train the selected model on the obtained dataset.
3. Analyze the results.
4. Adapt the system to a specific speaker and analyze the results.

Literature:

KIM, Jaehyeon; KONG, Jungil; SON, Juhee. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech. In: *International Conference on Machine Learning*. PMLR, 2021. p. 5530-5540.

KOPP, Matyáš, et al. ParCzech 3.0: A large Czech speech corpus with rich metadata. In: *International Conference on Text, Speech, and Dialogue*. Springer, Cham, 2021. p. 293-304.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Brukner Jan, Ing.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	4.5.2023

Abstract

This thesis aims to develop a model that can convert input text written in Czech into speech that closely resembles a target speaker. This work is based on the VITS text-to-speech neural network model.

The workflow is as follows: a Czech dataset is acquired, the neural network is trained, the trained model is then used to generate audio samples, which are evaluated using several objective metrics. A personalized dataset is developed and used to fine-tune the model, and the evaluation process is repeated.

As a result, two fine-tuned models were developed. The male model achieved a MOS of 4.12, and the female model achieved a score of 3.02. The scores prove that a base model fine-tuned using a personalized dataset can achieve results close to the original audio.

The contribution of this thesis is, apart from the personalized models, the pipeline for audio evaluation and dataset development, which can be easily adjusted for tasks on different data. In addition, a detailed analysis of best practices applied during the development of new datasets is provided.

Abstrakt

Tato práce si klade za cíl vytvořit model, který dokáže převést vstupní text na řeč cílového mluvčího. Základním stavebním kamenem je VITS model.

Postup byl následující: získal se obecný český dataset, na kterém se natrénoval model neuronové sítě, jenž se poté využil pro generování audio nahrávek, které se vyhodnocovaly pomocí objektivních metrik. Následně se vytvořil personalizovaný dataset, na kterém se provedl fine-tuning modelu získaného v předchozím kroku. Opět se provedlo vyhodnocení kvality nahrávek.

Výsledkem jsou dva personalizované modely. Model mužského mluvčího dosáhl v poslechových testech skóre 4.12/5 (MOS), model ženské mluvčí pak 3.02/5. Výsledky subjektivních i objektivních metrik ukázaly, že postupem zvoleným v této práci je možné vyvinout model, který se svou kvalitou generovaných nahrávek blíží skutečné řeči.

Přínosem této práce je, kromě personalizovaných modelů, i vytvoření vyhodnocovacího systému zpracování dat, které je možno uzpůsobit k evaluaci audio nahrávek z jiných modelů. Práce popisuje i způsob tvorby nového datasetu, který se může využít při tvorbě dalšího jiného datasetu v libovolném jazyce.

Keywords

speech processing, signals, neural networks, Czech text-to-speech, dataset development, audio evaluation metrics, fine-tuning

Klíčová slova

zpracování hlasu, signály, neuronové sítě, české text-to-speech systémy, tvorba datasetů, metriky vyhodnocení audia, ladění modelu

Reference

LUNER, Michal. *Text-to-Speech Personalization.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jan Brukner

Rozšířený abstrakt

Cílem práce je vytvořit TTS (**T**ext-**T**o-**S**peech) model, který dokáže převést vstupní text na řeč cílového mluvčího. Práce staví na modelu neuronové sítě VITS.

V prvním kroku bylo třeba získat dostatečně velký český dataset. Pravděpodobně nejoptimálnějším a především veřejně dostupným zdrojem se stal dataset sestávající z nahrávek ze sněmovny ČR. Používaný toolkit pro trénování se pak patřičně uzpůsobil struktuře datasetu a natrénoval se základní model. Za použití tohoto modelu se vygenerovaly nahrávky, které se vyhodnotily pomocí objektivních metrik jako je Word Error Rate, Character Error Rate či Mel-Cepstral Distortion. Kromě toho se kladl důraz na verifikaci, tj. podobnost generované nahrávky k cílovému mluvčímu. Následně se vytvořil personalizovaný dataset, který se využil pro fine-tuning základního modelu. Datasety vznikly dva, jeden pro mužského a jeden pro ženského řečníka. Výstupy obou personalizovaných modelů se opět vyhodnotily, tentokrát i subjektivními poslechovými testy.

Začátek práce čtenáře nejprve seznamuje s dostupnými TTS modely, kde se model VITS vybraný pro tuto práci rozebírá podrobněji.

Následuje analýza veřejně dostupných datasetů, které by připadaly v úvahu pro trénování TTS modelu. Jelikož TTS modely vyžadují specifické vlastnosti trénovacích dat, datasety jsou ohodnoceny na základě těchto požadavků. Datasety vytvořené v této práci pak tyto požadavky taktéž splňují.

V další části jsou popsány jednotlivé metriky využité pro vyhodnocení vygenerovaných nahrávek jednotlivých modelů. Kromě základních myšlenek fungování daných metrik jsou uvedeny i příslušné knihovny a toolkity, které se v této práci nasadily.

Natrénované modely se poté používají pro generování nahrávek. Kromě výsledků metrik jsou rozebrány i nedokonalosti nahrávek. Jsou představena a nasazena řešení, jejichž účinnost potvrzují zlepšené hodnoty metrik. Po dosažení přijatelných výsledků metrik pro základní model se přechází k personalizaci, tedy doladění modelu na konkrétního mluvčího. Modely se personalizovaly jak na celém vytvořeném datasetu, tak experimentálně i na jeho podčásti. Tím se demonstrovala schopnost vytvoření kvalitního modelu i z omezeného množství dat, příp. i trénovacích kroků. Závěrem se pak vygenerované nahrávky vyhodnotily subjektivními poslechovými testy, ve kterých především mužský model dosáhl výsledků přibližujícím se skutečné řeči.

Konec práce pak představuje webovou aplikaci, která zaobaluje modely vytvořené v této práci do uživatelského rozhraní.

Výstupem práce jsou dva personalizované modely, kde model mužského mluvčího dosáhl v poslechových testech skóre 4.12/5, model ženské mluvčí pak 3.02/5. Výsledky práce pak ukazují, že lze vytvořit model produkující kvalitní nahrávky i za použití omezeného množství trénovacích dat i trénovacích kroků. Z programového hlediska je pak k dispozici sada skriptů, které se dají využít pro vyhodnocení nahrávek z jiných modelů.

Text-to-Speech Personalization

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana inženýra Jana Bruknera. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Luner
May 7, 2023

Acknowledgements

I would like to thank my supervisor Jan Brukner, Ing. for replying to my countless questions, helping me with understanding the principles used in this thesis, and most importantly assisting me with numerous problems I have encountered.

Contents

1 Introduction **3**
1.1 Broader Impact . . . 4

2 Text-to-Speech Systems **5**
2.1 History of TTS Systems . . . 5
2.2 Two-Stage Models . . . 6
2.3 Text-to-spectrogram . . . 6
2.3.1 Tacotron . . . 6
2.3.2 Tacotron 2 . . . 8
2.3.3 FastSpeech . . . 9
2.3.4 GlowTTS . . . 11
2.4 Spectrogram-to-audio . . . 13
2.4.1 Griffin-Lim Algorithm . . . 13
2.4.2 WaveNet . . . 14
2.4.3 HiFi-GAN . . . 16
2.5 End-to-End Models . . . 19
2.5.1 VITS . . . 19

3 Datasets **27**
3.1 What Makes a Good TTS Dataset . . . 27
3.2 Current options of available Czech datasets . . . 28
3.3 Large Corpus of Czech Parliament Plenary Hearings . . . 28
3.4 ParCzech 3.0: A Large Czech Speech Corpus with Rich Metadata . . . 30
3.5 Personalized Datasets . . . 32
3.5.1 Male Dataset . . . 33
3.5.2 Female Dataset . . . 34

4 Evaluation Pipeline For New TTS Models **35**
4.1 Word Error Rate & Character Error Rate . . . 35
4.2 Mel-Cepstral Distortion . . . 36
4.3 Dynamic Time Warping . . . 37
4.4 Verification . . . 38
4.4.1 Cosine Distance . . . 38
4.5 MOS . . . 40

5 Implementation **41**
5.1 Training the ParCzech 3.0 Dataset . . . 41
5.1.1 Baseline Evaluation . . . 41

5.1.2 Trained Models . . . 42
5.1.3 Unseen Speakers . . . 45
5.2 Personalization . . . 46
5.2.1 Male Speaker . . . 46
5.2.2 Female Speaker . . . 49
5.2.3 MOS Evaluation . . . 50
5.3 User Application . . . 52

6 Conclusion **53**
6.1 Future Works . . . 54

Bibliography **55**

A Manual **61**

B Additional Model Architectures **63**

C Plots **64**
C.1 Base Model . . . 65
C.2 Male Model . . . 66
C.3 Female Model . . . 67

Chapter 1

Introduction

Text-to-speech systems can be deployed in numerous occasions, such as in assistive systems controlled by voice or other human-computer interaction. The most famous application of text-to-speech systems are probably voice announcements in shopping centres, public transport or in entertainment (deep fakes).

The goal of this work is to train a TTS (**T**ext-**T**o-**S**peech) model that converts an input text in the Czech language into a speech of a target speaker.

Although there can be found several publicly available TTS models, they typically rely on high-resource languages such as English. This serves as the leading reason for this work, which aims to explore the possibilities of training a model using samples in Czech, where options of publicly available datasets are rather limited.

First, a base model is created by training it on a large number of audio samples in the Czech language. Attention is given to a thorough analysis of a dataset's properties and their impact on the resulting trained model. Each trained model undergoes evaluation using a pipeline consisting of several metrics, analyzing it with respect to various audio properties. The ultimate goal is then to fine-tune the best-performing base model to match the target person's speech characteristics as closely as possible. This is accomplished using personalized datasets developed in this thesis. In addition to the metrics from the evaluation pipeline, the final personalized models are evaluated through subjective listening tests to conclude the final results. Once the personalized model is wrapped up in a user interface, its main purpose is to process a given text and synthesize it.

This work could have future applications in fine-tuning the base model for a different speaker. Additionally, the developed evaluation pipeline can be easily customized for any other TTS architecture as well as the dataset adjustments ideas described in this thesis can serve for optimizing or creating datasets.

The theory chapter describes the history and current options of text-to-speech systems, the dataset chapter deals with publicly available Czech datasets and custom personalized datasets, followed by the implementation chapter, which discusses the results of evaluation metrics on a given dataset. More detailed evaluation results can be found in the appendix. The last chapter demonstrates the uses and architecture of the developed user application.

1.1 Broader Impact

It should be taken into account that TTS systems have the potential to do harm. It is important to think about possible consequences when using such systems with bad intentions. On the other hand, these systems can also offer a number of advantages, such as helping people with speech or hearing disabilities.

Because the fine-tuned models developed in this thesis produce high-quality audio files, it is crucial to stress that they should be used responsibly.

Chapter 2

Text-to-Speech Systems

In this chapter, the history of TTS systems is briefly discussed.

Next, the contemporary two-stage systems are described in terms of their architecture and features. A two-stage system consists of a text-to-spectrogram and spectrogram-to-audio model. The description of such models and their deployment within more complex solutions are provided, and advantages and disadvantages are discussed too.

A detailed description of the model utilized in this thesis is presented in the end-to-end systems section.

2.1 History of TTS Systems

As written in [61], the first text-to-speech systems date back to the 17th century and were very different from the ones that are used nowadays. This brief overview focuses mainly on the second half of the 20th century.

In the late 70s, the very first reading machines for blind people were introduced. Due to their high cost, they were frequently located in libraries and not by individuals. Another step further was LPC (**L**inear **P**redictive **C**oding). The systems using this method offered more natural and less robotic-sounding results. Until this point, the speech synthesis systems had found their purpose in helping blind people.

In the 80s, speech synthesis entered the video game world and entertainment. In this era, a few systems that could sing were developed, though softer consonants still caused the synthesized speech to sound robotic.

This began to change in the 90s and in the early years of the 21st century. Although hardware possibilities advanced, researchers worldwide faced issues of how to pronounce phonemes properly, and how to work with intonation, tone, and other speech properties.

Before the rise of neural networks, the HMMs (**H**idden **M**arkov **M**odels) were predominantly used. This finite-state machine was used as a framework for various speech applications. In order to synthesize speech, the HMMs use statistical probabilities. Compared to the nowadays widely used neural networks, the HMMs, being a linear model, are not as flexible and their performance in recognizing different voices and phonemes at the beginning or at the end of the phrase is way worse. Hence, nowadays commonly used models are DNNs (**D**eep **N**eural **N**etworks).

2.2 Two-Stage Models

Two-stage models separate the neural network model into text-to-spectrogram and spectrogram-to-speech modules. The text-to-spectrogram module takes linguistic input, such as phonemes or words, and converts it into a spectrogram or other spectral representation. For example, Tacotron takes a sequence of character embeddings and outputs a linear spectrogram, whereas GlowTTS accepts phonemes and outputs a Mel-spectrogram. This is then followed by synthesizing the spectral representation into a speech waveform.

One of the primary advantages of the two-stage model approach is its modularity, which allows for easier model design and experimentation. However, this modularity also presents a disadvantage as each module needs to be trained independently. To improve performance, fine-tuning techniques can be deployed to optimize the two-stage model as a whole.

In the following sections, each of these modules will be discussed in detail, exploring the architectures and training procedures used for each. Each subsection builds on a source paper cited at the beginning.

2.3 Text-to-spectrogram

Text-to-spectrogram models convert text input into a spectrogram representation.

Typically, TTS systems receive text as their input, which can then be transformed into phonemes or other alternative representations, such as character embeddings, for further processing. A phoneme sequence can be obtained using analytical approaches. On the other hand, embeddings are outputs of a neural network.

There are several representations of spectrograms that can be used in text-to-spectrogram models. Linear spectrograms represent the frequency spectrum using a linear scale, while Mel-spectrograms and log-spectrograms use a logarithmic scale to better simulate the perceptual nature of sound. For each model, the selection of an intermediate representation is detailed.

2.3.1 Tacotron

Tacotron [60] accepts an input sequence of character embeddings and converts it into spectrogram frames, which are then turned into waveform. The architecture is shown in figure 2.1.

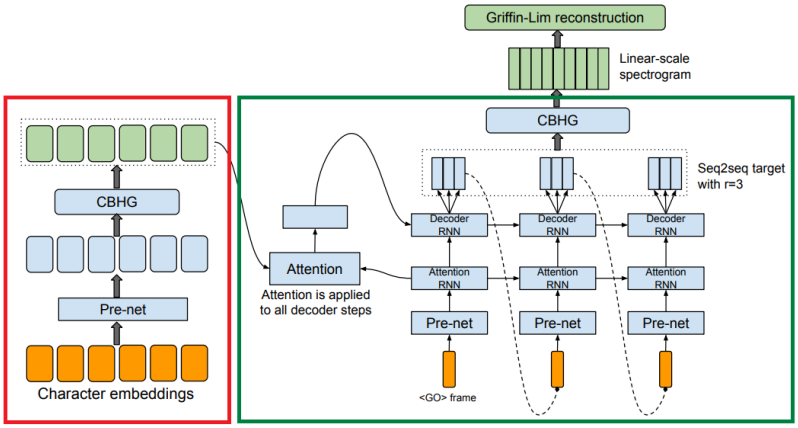

Figure 2.1: Tacotron model architecture [60] with highlighted encoder and decoder parts.

Tacotron is a sequence-to-sequence (encoder-decoder-based) model with attention mechanism. In general, the attention mechanism enables a neural network model to selectively focus on acoustic features of the input that are relevant for a given task, which then leads to improved performance [57].

The input text is represented as a sequence of character embeddings. Each embedding is passed through Pre-net with ReLU[1]. The output of the Pre-net is further transformed by a CBHD module (**C**onvolution **B**ank, **H**ighway network, Gate**d** recurrent unit) to generate the final encoder representation. The CBHG module is a neural network designed for extracting representations from sequences. Specifically, CBHG's input sequence is first convolved with sets of convolutional filters with residual connections. Then, the outputs are passed to a highway network, which extracts high-level features. Highway networks address the issue of gradient loss while training very deep networks by enabling information to flow across multiple layers [49], which is a similar idea to LSTM (**L**ong **S**hort **T**erm **M**emory). While both mechanisms use some sort of a gating function, LSTM regulates how much of the previous state is preserved or forgotten in RNNs. Highway networks, on the other hand, are usually deployed in feed-forward networks and control how much the input is transformed and passed to other layers. These then allow the network to better capture relevant information from the input sequence. Finally, a bi-directional GRU[2] RNN[3] (**R**ecurrent **N**eural **N**etwork) is used to extract sequential features from both forward and backward context. Ablation studies proved that using CBHD reduces the number of mispronunciations in the generated samples. The encoder's output, known as the context vector, is passed to the attention module for further processing.

The context vector together with the output of an RNN cell are used as input to the attention module. The output of this module then goes to the decoder RNNs. To learn the alignment between speech and text, a high-resolution linear-sale spectrogram is not necessary and a compressed version, such as a Mel-spectrogram, is sufficient as long as it contains enough information related to intelligibility and prosody. An all-zero <GO> frame defines the starting decoder step.

The CBHG module is used as a post-processing network to convert the output of the seq2seq module[4] into a representation that can be synthesized into a waveform. The CBHG module learns to predict spectral magnitudes, which are then passed to the Griffin-Lim algorithm (described in 2.4.1) to generate the final waveform. The concept of utilizing a post-processing net can be extended to other tasks, such as the estimation of vocoder parameters. The Griffin-Lim algorithm was here chosen for its simplicity.

Although Tacotron is not an end-to-end system, it can be easily trained only using text and audio pairs. This model generates speech way faster thanks to frame-level synthesis, whereas other models, such as WaveNet (2.4.2), use sample-level autoregressive methods. Frame-level defines an approach where each spectrogram frame is synthesized and then concatenated with others into the final audio sample.

As [35] states, the loss functions provide a measure of how well the model is able to predict the output for a given input. In general, losses guide the optimization process

[1] **Re**ctified **L**inear **U**nit is an activation function that helps to introduce non-linearity into the model and improve its ability to learn complex relationships between inputs and outputs [10].

[2] GRU is a special kind of LSTM [9].

[3] Unlike feed-forward networks described in section 2.3.3, RNNs can propagate information in both directions.

[4] Predicting r output frames at each decoder step was found to enhance performance and stability, likely due to the correlation between neighboring speech frames.

during training and help to improve the accuracy of the model. When training Tacotron, the L1 loss for the decoder RNN, which outputs Mel-spectrograms, is calculated as

$$L_{dec} = ||x_{mel} - \hat{x}_{mel}||_1, \tag{2.1}$$

where x_{mel} denotes the Mel-spectrogram of the original audio and $\hat{x}_{mel}$ is the generated Mel-spectrogram. Similarly, the L1 loss for the post-processing net (linear-scale spectrogram) is defined as

$$L_{post} = ||x_{lin} - \hat{x}_{lin}||_1, \tag{2.2}$$

where x_{lin} denotes spectrogram of the original audio and $\hat{x}_{lin}$ is the generated spectrogram [60].

The MOS results can be found in the table 2.1.

2.3.2 Tacotron 2

Similarly to the previous version of Tacotron, this is a sequence-to-sequence (encoder-decoder-based) feature prediction network with attention. Tacotron 2 [48] predicts a sequence of Mel-spectrogram frames from an input sequence of character embeddings. Then, instead of the Griffin-Lim algorithm, a modified version of WaveNet generates audio samples from the predicted Mel-spectrogram frames.

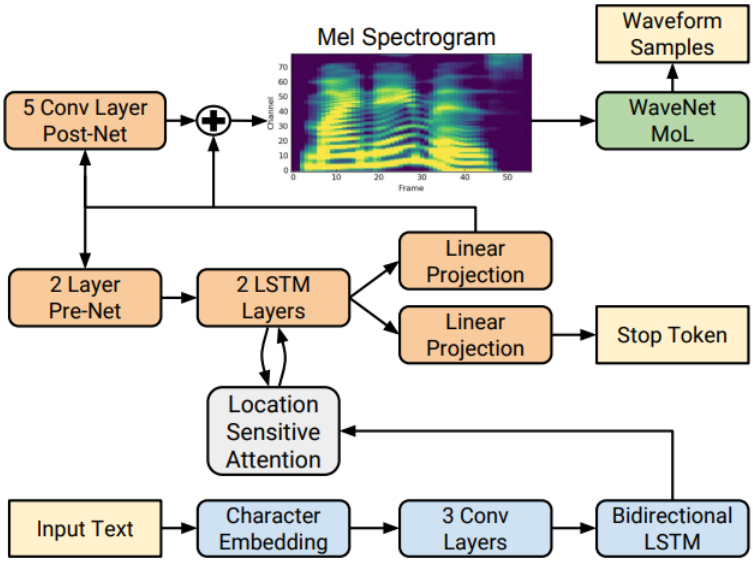

Figure 2.2: Tacotron 2 model architecture [48]. Modules at the bottom represent the encoder, which is connected via the attention module to decoder modules for the Mel-spectrogram generation.

As shown in figure 2.2, the input characters represented as character embeddings are processed through a stack of 3 convolutional layers. The output of the final convolutional layer is fed into a bi-directional LSTM layer, which generates the encoded features. As written in [51], LSTM is a type of RNN capable of analyzing previous steps to a deeper extent than a normal RNN.

The output of the encoder is passed to the location-sensitive attention module, which uses cumulative attention weights from previous decoder time steps. Unlike the original idea of the attention mechanism, this approach ensures that the model moves forward more consistently through the input and minimizes the error of ignoring or repeating words.

The autoregressive[5] decoder then predicts a Mel-spectrogram from the encoded input sequence. To aid in learning attention, the prediction from the previous time step is first passed through a pre-net containing linear layers followed by ReLUs. The pre-net's output and attention module's output are concatenated and passed through two LSTM layers to predict the target Mel-spectrogram frame. In parallel, this concatenation is also passed to an activation sigmoid function to predict a Stop Token, which is utilized only during inference to signal the end of generating the sample. This allows for diverse sample duration. Next, the predicted Mel-spectrogram is passed through post-net which adds a residual (spectrogram improvement) to refine the final reconstruction.

Finally, the WaveNet vocoder (2.4.2) is modified to transform Mel-spectrogram into a waveform. In comparison to the previous model, an improvement was achieved in terms of audio quality despite the fact that the model uses vanilla LSTM instead of a more complex CBHG module.

The training process consists of two parts. First, the feature prediction network is trained independently using the teacher-forcing method. Every predicted Mel-spectrogram frame is conditioned on both the encoded input sequence and the corresponding previous frame in the ground truth spectrogram. Using this approach, a precise alignment with the target waveform samples can be achieved. MSE[6] (**M**ean **S**quared **E**rror) is used to calculate the difference between the ground-truth and predicted Mel-spectrogram. Second, outputs of the feature prediction network (Mel-spectrograms) are then used to train a modified WaveNet.

The MOS results can be found in table 2.1.

2.3.3 FastSpeech

FastSpeech [38] is a Transformer-based feed-forward network that generates Mel-spectrograms in parallel from a phoneme sequence.

A feed-forward architecture describes a network where information flows only in one direction, from the input layer to the output layer, whereas RNNs can have feedback connections [45].

As stated in [57], the novel idea of Transformer-based neural networks is the use of a self-attention mechanism instead of RNNs to process sequential data. This approach minimizes the probability of losing long-range dependencies. In the original paper focused on language translation, the self-attention mechanism allows the model to decide the importance of different words in the input sequence when generating the output sequence. The Transformer architecture along with a brief description can be found in the appendix B.1.

As an autoregressive Mel-spectrograms generation can lead to a slow inference, lack of robustness and controllability, FastSpeech addresses these issues. The slow autoregressive inference is caused by conditioning on previously generated Mel-spectrograms, while parallel Mel-spectrogram generation can speed up the synthesis process. To minimize skipping and repeating words, often caused by incorrect attention alignments between speech and text,

[5] Autoregressive generation conditions new samples solely on previous outputs and current input.

[6] MSE measures the difference between two sets of n values using formula $MSE = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2$, where X_i is the original value and $\hat{X}_i$ is the predicted one [31].

a phoneme duration predictor ensures hard alignments between phoneme and its Mel-spectrogram. Lastly, the lack of controllability can be addressed using a length regulator that adjusts phoneme duration and can modify voice speed and prosody as well [38]. The FastSpeech architecture is shown in figure 2.3.

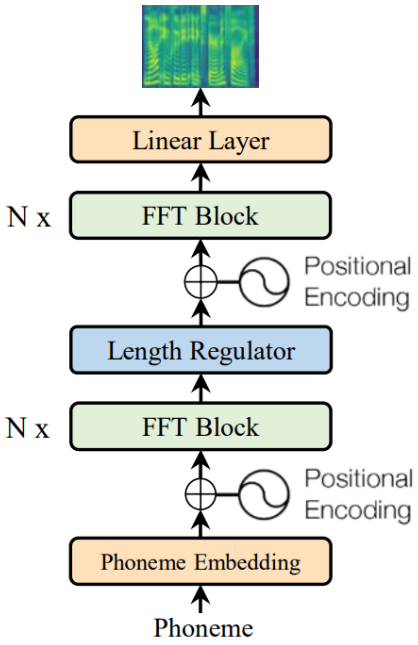

Figure 2.3: FastSpeech model architecture [38]. FFT block transforms an input phoneme sequence into a Mel-spectrogram. Length regulator enables speech duration adjustments.

The system utilizes an FFT (**F**eed-**F**orward **T**ransformer) block, which incorporates self-attention and 1D convolutional network to transform phoneme embeddings into Mel-spectrograms. Since the FFT Block uses self-attention, which attends to all positions in the sequence simultaneously, positional encoding is a technique that injects information about the position of each phoneme in the input sequence [43].

To address the issue of different lengths between the phoneme and Mel-spectrogram sequences (since one phoneme usually corresponds to several frames in a Mel-spectrogram), a length regulator is employed to up-sample the phoneme sequence based on the duration of each phoneme to match the Mel-spectrogram sequence length. This also enables control over voice speed and some aspects of prosody by adjusting the duration of space characters in the input sentence.

The phoneme duration predictor is a key component of the length regulator. First, an autoregressive teacher TTS model generates attention alignments for each training sequence pair (phoneme and corresponding number of Mel-spectrograms). These ground-truth phoneme durations are extracted from the teacher model and used for training the phoneme duration predictor.

The non-autoregressive approach improved inference performance[7] by generating the Mel-spectrogram in 0.025 seconds, which is faster than the autoregressive Transformer TTS

[7]Test conducted on a server with 12 Intel Xeon CPU, 256GB memory, 1x NVIDIA V100 GPU and batch size of 1 [38].

model's time of 6.735 seconds. The overall speed, including waveform synthesis, depends on the vocoder used.

During training, two MSE losses are deployed. First, duration loss denotes the difference between the predicted and extracted duration of the duration predictor. Second, Mel-spectrogram loss between the ground truth and generated Mel-spectrogram [38].

The MOS results can be found in table 2.1.

2.3.4 GlowTTS

GlowTTS [20] is a parallel flow-based model that does not require external aligners like FastSpeech to obtain aligned attention maps between text and speech. Instead, it internally learns its own alignment by efficiently searching for the most probable monotonic alignment between text representation (phonemes in this case) and the latent representation of speech.

One of the advantages of using flow-based models is that they can efficiently generate new samples by simply inverting the flow. Specifically, GlowTTS builds on normalizing flows, which provide the transformation of a simple distribution into a more complex one by applying sequences of invertible transformations until the needed complexity is achieved. This is accomplished using the change of variables formula [52]. In general, the model using flows is able to capture more complex data, which then leads to generating high-quality samples [39].

During training, the flow-based decoder takes ground-truth Mel-spectrogram as input and creates a latent representation. The representation together with the projected text is then aligned using MAS (**M**onotonic **A**lignment **S**earch). Since there is no ground-truth Mel-spectrogram available during synthesis, the training process involves training a duration predictor, which can predict the best alignment only from input text [52]. Figure 2.4 shows the training pipeline of the model.

As mentioned in [52], GlowTTS proposes a robust method of finding the most likely alignment between speech and text. Approaches used in models like Tacotron 2 (2.3.2) tackle the alignment using autoregressive attention mechanisms, which might lead to errors, such as word-skipping or repeating. In MAS, on the other hand, the approach of enforcing hard monotonic alignments in combination with normalizing flows helps with the robustness of TTS, especially when longer utterances are synthesized.

To illustrate the idea behind the MAS, the following example from [52] will be considered. During the training, the decoder outputs 100 Mel-spectrogram frames (vectors $z_1, z_2, \ldots, z_{100}$), and the text encoder outputs mean vectors $(\mu_1, \mu_2, \ldots, \mu_5)$ and standard deviation vectors $(\sigma_1, \sigma_2, \ldots, \sigma_5)$ for each character (phoneme) in input text "hello". In the next step, a likelihood score matrix $P \in \mathbb{R}^{5\times100}$ is constructed where each entry is computed using $P_{ij} = \log(\mathcal{N}(z_j; \mu_i, \sigma_i)$, where $\mathcal{N}$ denotes normal probability density function. This matrix defines the likelihood of the i-th character aligning with the j-th Mel-spectrogram frame. Since MAS shares similar properties with DTW, further information can be found in section 4.3.

During synthesis, the statistics of the prior distribution (variance and mean) are first obtained from the input text using the text encoder. Next, the duration predictor estimates the duration of each phoneme in the input text. In the following step, latent representations are sampled (with respect to phoneme durations) from the prior distribution, concatenated, and the decoder inverts its flow to generate the final Mel-spectrogram [52].

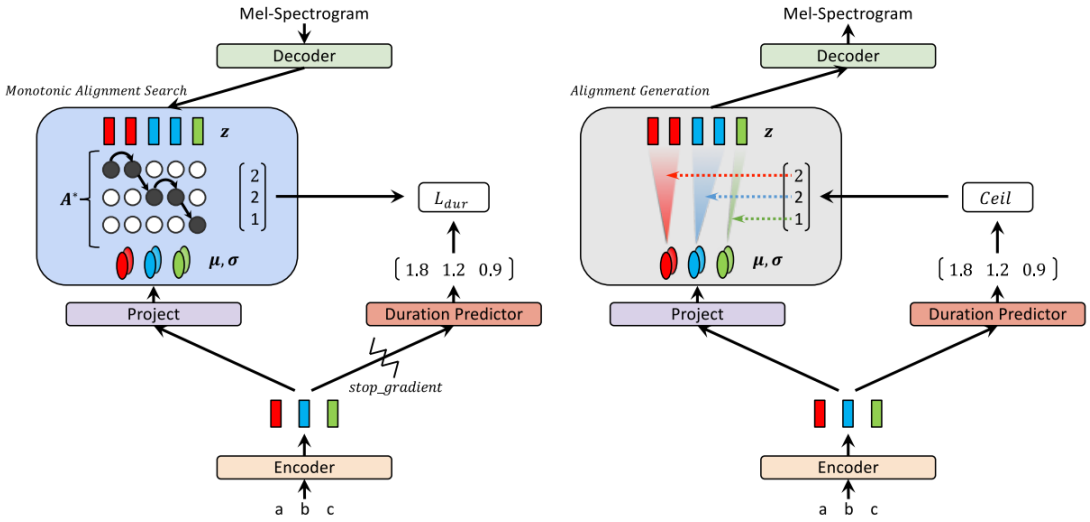

Figure 2.4: GlowTTS training and inference procedures [20]. After the training procedure, the decoder's flow is inverted to produce Mel-spectrograms conditioned on the input text and duration predictor.

GlowTTS utilizes the duration predictor loss, which can be calculated as

$$L_{dur} = MSE\big(f_{dur}(sg[f_{enc}(c)]), d\big), \tag{2.3}$$

where f_{dur} denotes the duration predictor, sg is the stop gradient operator, f_{enc} is the text encoder, c is the input Mel-spectrogram, and d denotes the duration label calculated from the alignment A (calculated using MAS).

GlowTTS provides 15x faster Mel-spectrogram synthesis compared to Tacotron 2 while preserving the audio quality [20]. The novel concept of MAS is also implemented in other models, such as in VITS (2.5.1).

The MOS results can be found in table 2.1.

Table 2.1: Results of MOS for Tacotron [60], Tacotron 2 [48], FastSpeech [38], and GlowTTS [20]. MOS is provided for both ground truth and generated samples for each model.

Model + Vocoder	**Ground-truth**	**Generated**
Tacotron + Griffin-Lim	N/A	3.82 (±0.09)
Tacotron 2 + WaveNet	4.58 (±0.05)	4.53 (±0.07)
FasSpeech + WaveGlow	4.41 (±0.08)	3.84 (±0.08)
GlowTTS + WaveGlow	4.54 (±0.06)	4.01 (±0.08)

2.4 Spectrogram-to-audio

Vocoders, the spectrogram-to-audio models, can convert various types of spectrograms (linear, Mel, log) into waveform audio. This process involves reconstructing the audio signal from its frequency content by using algorithms that can reproduce the audio waveforms. Other representations, such as MFCCs (**M**el-**F**requency **C**epstral **C**oefficients), can also be used as input to some of these models. This section describes the evolution of vocoders and their properties.

2.4.1 Griffin-Lim Algorithm

An algorithm from 1986 is able to estimate a signal from a modified STFT (**S**hort-**T**ime **F**ourier **T**ransform). More specifically, a phase is estimated from a magnitude spectrogram passed as input. In theory, estimating new signal $x(n)$ from the magnitude spectrogram of $y(n)$ can be achieved by minimizing the MSE (**M**ean **S**quared **E**rror) between the STFT of the estimated signal and the MSTFT (Modified STFT) [15] calculated as follows:

$$D\Big[x(n), Y_w(mS,\omega)\Big] = \sum_{m=-\infty}^{\infty} \frac{1}{2\pi} \int_{\omega=-\pi}^{\pi} |X_w(mS,\omega) - Y_w(mS,\omega)|^2 \, d\omega, \tag{2.4}$$

where $D[\cdot]$ denotes the distance between the STFT of an estimated signal $x(n)$ and the STFT of a signal $y(n)$, m is a frame index, S is a hop size to the successive window, and ω is frequency. While $X_w(mS,\omega)$ is an STFT of an existing signal $x(n)$, there may not be $y(n)$, whose STFT is represented by $Y_w(mS,\omega)$. The aim of this computation is to estimate $X_w(mS,\omega)$ to be as similar as possible to $Y_w(mS,\omega)$.

This idea introduces a few issues since a signal that consists of complex numbers cannot be reconstructed with a magnitude spectrogram only, as written in [6]. However, this reconstruction can be achieved using correlated values. These values can be found in the magnitude spectrogram since frames computed using STFT overlap in time, therefore, they partly share the same information[8] that have both magnitude and phase components. The Griffin-Lim uses this leakage between time-frequency components, which allows for estimating the phase components from magnitude-only values.

To estimate the phase of a signal, an iterative approach is deployed. First, an input magnitude spectrogram receives an initial phase estimate. Next, ISTFT and STFT are iteratively applied to the signal, followed by reapplying the original magnitude at each step, i.e., only the phase estimate is used for the next iteration. Using this approach, information leaks between time-frequency bins, which gradually improves the phase estimate. As soon as applying ISTFT and STFT do not change either phase or magnitude, the estimate can be found as a resulting signal. Other examples can be found in [6].

The described ideas are illustrated in algorithm 1.

[8] This „shared part“ is given by STFT's hop-size or window-size parameter.

Algorithm 1 Griffin-Lim algorithm [6]

```
function GriffinLim(magnitude_spectrogram)
    #apply initial random phase
    spectrogram = apply_random_phase(magnitude_spectrogram)
    #set number of iterations
    n ← 30
    for i ← 0 to n do
        #apply ISTFT
        waveform = ISTFT(spectrogram)
        #apply STFT
        spectrogram = STFT(waveform)
        #use new phase values & magnitude information remains unchanged
        spectrogram = magnitude_spectrogram × spectrogram.phase_values
    end for
end function
```

This algorithm does not rely on any neural networks and its implementation compared to other modern solutions is fairly simple. However, this algorithm delivers lower-quality results compared to modern solutions.

2.4.2 WaveNet

WaveNet [56] is a convolutional neural network capable of generating high-quality speech waveforms using causal convolutions as shown in figure 2.5.

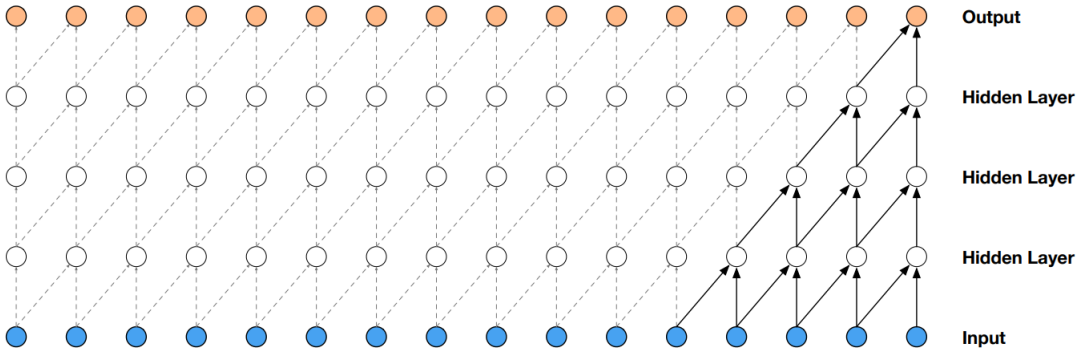

Figure 2.5: A stack of causal convolutional layers in WaveNet [56].

The sample generation is autoregressive. When a sample is predicted, it is fed back into the network to repeat this process for a new sample. The following architectures, such as WaveGlow [33], attempt to solve the issue of autoregressive processing by parallelizing audio generation, which leads to faster synthesis speed.

Unlike RNNs with recurrent connections, convolutional networks do not have any, therefore, they are faster to train. The downside of this, however, is that CNN (**C**onvolutional **N**eural **N**etwork) has many hidden layers and generating new samples from all of them would increase the computational cost. To reduce the number of causal convolutions layers, dilated convolutions are deployed. In other words, previous values are skipped with certain steps, as shown in figure 2.6.

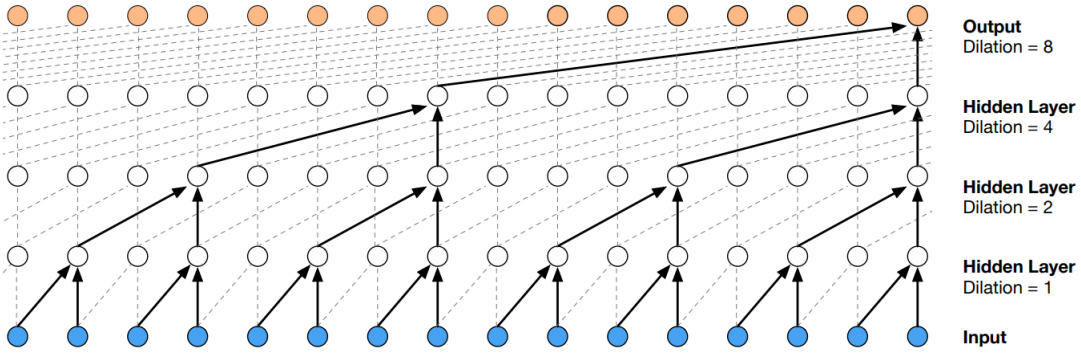

Figure 2.6: A stack of dilated causal convolutional layers in WaveNet [56].

Since raw audio is a 16-bit sequence (65,536 probabilities per time step), it is transformed and quantized into 256 possible values without losing quality. Each WaveNet's output is a 256-digit long vector, which is normalized using Softmax function [14] to produce categorical distribution[9]. More details can be found in paper [56].

Similarly to RNN sequence-to-sequence models using an attention mechanism allowing to focus only on important parts of the input sequence, feed-forward networks, such as CNNs, use gated activation units to filter important information.

WaveNet also utilizes skip connections. In general, they help with propagating signals more effectively through the network, which leads to improved performance, and the loss of important information is prevented [1].

To closer analyze the architecture of WaveNet, figure 2.7 shows a stack of residual blocks, the key element of this model.

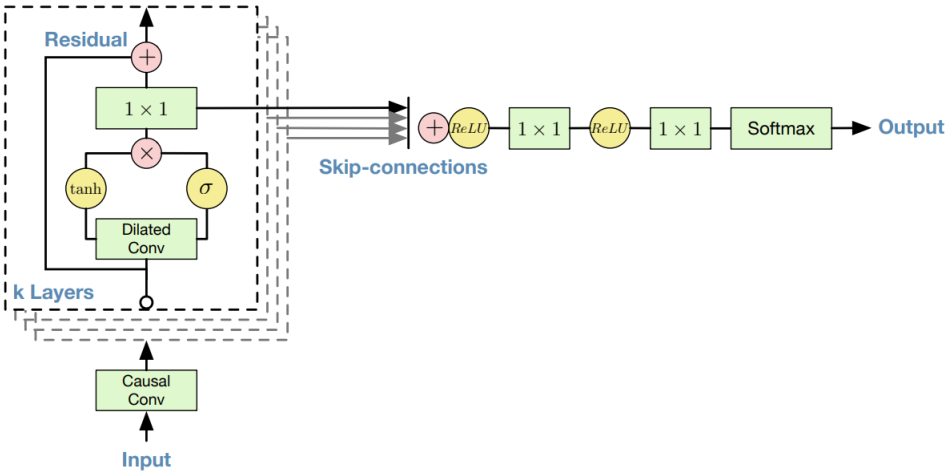

Figure 2.7: Overview of the residual block architecture [56].

First, input goes through a convolution layer. Once the convoluted input is fed to the residual block, it goes through a dilated convolution layer and the output is passed through the non-linearity unit. After that, it is further processed through a 1x1 convolution layer.

[9]By applying Softmax on the input vector, the values are normalized and their sum adds up to 1. This approach allows for viewing the values as probabilities.

The output is added to the original input and this is further passed to the next residual block. The whole process is repeated for all blocks in the stack. Simultaneously, outputs of each block are summed and a chain of ReLUs and 1x1 convolution layers are applied to output the final, quantized sample.

WaveNet can be conditioned with additional input to produce audio with the required acoustic characteristics. The audio generation can be conditioned either globally, i.e., passing a speaker embedding[10] to generate such an output distribution that reflects the speaker's characteristics, or locally, which means passing a spectrogram. In speech synthesis, WaveNet is used as a vocoder and the input condition is usually a spectrogram.

During the training phase, the ground-truth samples are used as inputs. At each time step, WaveNet outputs a quantized distribution, which is compared to the ground truth sample using cross-entropy loss.

Later, Fast WaveNet [29] was released. Thanks to using cache memory, no redundant convolutions needed to be calculated and that improved the previous model in terms of time complexity.

The MOS results can be found in table 2.2.

2.4.3 HiFi-GAN

As stated in [41], GANs (**G**enerative **A**dversarial **N**etworks) consist of a generator and discriminator. Both parts are trainable neural networks. The generator can be viewed as a transform function, which accepts a simple random variable and returns a random variable that follows the targeted distribution. On the other hand, the discriminator receives a point (or vector, ...) as input and calculates the probability of it being a real point. Both networks are trained jointly while:

- The generator's goal is to fool the discriminator, i.e., to maximize the classification error between real and generated data.
- The task of a discriminator is to decide whether the generated data is real or generated. Contrary to the generator, the aim is to minimize the classification error between real and generated data.
- In this iterative process, both the discriminator and generator improve their performance through competition, therefore, the training process is called adversarial.

Compared to autoregressive (WaveNet) and flow-based models (WaveGlow), HiFi-GAN [22] is computationally more efficient and produces higher-quality samples.

This model is composed of a generator and two discriminators: multi-scale and multi-period. For improving the training stability and overall model performance, additional losses are used.

[10] A speaker embedding is a vector (for example 512-dimensional) representing speaker's acoustic characteristics.

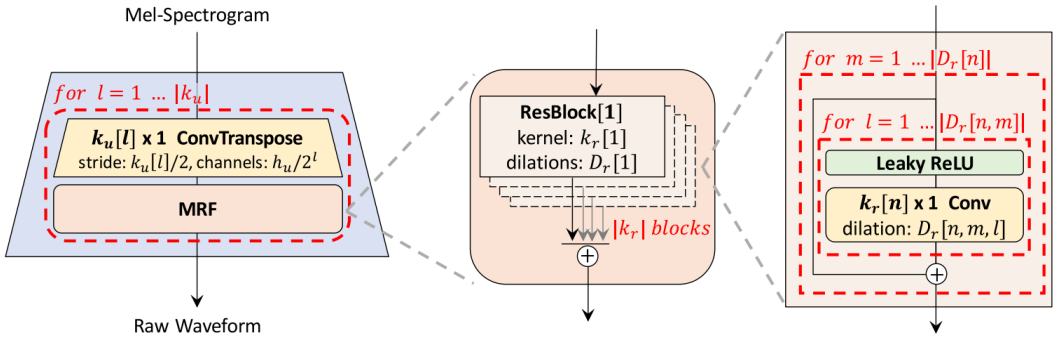

Figure 2.8: Architecture of a generator [22]. An input Mel-spectrogram is upsampled and the MRF module provides the generator with a wide range of contextual information from the input signal.

The generator shown in figure 2.8 is a CNN. An input Mel-spectrogram is upsampled to be as long as the temporal resolution of raw waveforms. After that, MRF (**M**ulti-**R**eceptive **F**ield fusion) is utilized.

MRF provides the generator with a wide range of contextual information from the input signal. More specifically, it analyzes different receptive fields (regions in the input sequence), which leads to enhancing the generator's abilities. That consequently results in more accurate and realistic audio synthesis. MRF module consists of multiple residual blocks of various kernel sizes and dilation rates. Parameters of MRF can be adjusted, either in favor of synthesis efficiency or sample quality.

Accurately identifying real or fake samples requires the correct learning of long-term dependencies. Thus, two types of discriminators are deployed.

Each speech audio contains signals with various periods. To generate a realistic speech sample, it is important to capture these periodic patterns. To achieve that, the chosen method is to use an MPD (**M**ulti-**P**eriod **D**iscriminator) that consists of multiple sub-discriminators, each of which analyzes only a specific periodic portion of the raw waveform. It should be noted that a minimum number of overlaps is desired.

Since MPD worked with non-overlapping samples, MSD (**M**ulti-**S**cale **D**iscriminator) analyzes the audio sequence as a whole. Specifically, MSD consists of 3 sub-discriminators (the second one shown in figure 2.9), each of which analyzes input at different scales: raw audio, 2x average-pooled audio, and 4x average-pooled audio.

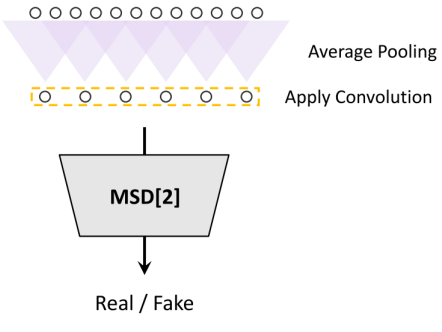

Figure 2.9: The second sub-discriminator of MSD [22]. Pooling describes a process of sliding a window over an audio signal where the values within the window are averaged to produce a single value.

Multiple losses are employed during the training process.

The discriminator[11] is trained to distinguish real samples as 1 and synthetic samples as 0, while the generator is trained to deceive the discriminator by improving the quality of its samples to be classified as close to 1 as possible. The results of GAN loss are calculated using least squares [27] loss functions. The discriminator (D) and the generator (G) losses are defined as:

$$L_{Adv}(G) = \mathbb{E}_s\Big[(D(G(s)) - 1)^2\Big], \tag{2.5}$$

$$L_{Adv}(D) = \mathbb{E}_{(x,s)}\Big[(D(x) - 1)^2 + (D(G(s)))^2\Big], \tag{2.6}$$

where x denotes the ground truth audio and s denotes the Mel-spectrogram of the ground truth audio (input condition).

Next, Mel-spectrogram loss measures the L1 distance[12] between the Mel-spectrograms of the synthesized waveform and the ground truth waveform. Ablation studies confirmed that Mel-spectrogram loss improves the generator's ability to produce realistic waveforms (see figure 2.2) and the loss can be calculated as

$$L_{Mel}(G) = \mathbb{E}_{(x,s)}\Big[||\phi(x) - \phi(G(s))||_1\Big], \tag{2.7}$$

where ϕ transforms a waveform into the corresponding Mel-spectrogram.

Finally, feature matching loss is measuring the L1 distance between intermediate features extracted by the discriminator for a real sample and a generated sample. This loss further encourages the generator to produce samples that have similar features to the real samples. It can be computed using the formula

$$L_{FM}(G; D) = \mathbb{E}_{(x,s)}\Big[\sum_{i=1}^{T}\frac{1}{N_i}\|D^i(x) - D^i(G(s))\|_1\Big], \tag{2.8}$$

where T denotes the number of layers in the discriminator, N_i is the number of features, and D^i denotes the features in the i-th layer of the discriminator.

[11] In spite of having multiple discriminators, they are generalized in one loss for simplicity.

[12] The difference between two vectors in space, denoted as $||vector_1 - vector_2||_1$.

Table 2.2: Ablation study results and comparison with WaveNet. Assessing the impact of each component on the quality of the synthesis [22] where the baseline value is derived from HiFi-GAN V3, which has the lowest parameter count among the versions considered, thus, it was used for the fastest results. HiFi-GAN outperforms WaveNet [56] and it is deployed in multiple TTS systems (such as in section 2.5.1).

Model	MOS
Ground Truth	4.57 (±0.04)
HiFi-GAN V1	4.36 (±0.07)
Baseline (HiFi-GAN V3)	4.10 (±0.05)
without MPD	2.28 (±0.09)
without MSD	3.74 (±0.05)
without MRF	3.92 (±0.05)
without Mel-spectrogram loss	3.25 (±0.05)
WaveNet	4.21 (±0.08)

HiFi-GAN generalizes well to unseen speakers and outperforms WaveNet and WaveGlow in terms of MOS. While fine-tuning WaveGlow did not bring samples quality improvement, fine-tuned HiFi-GAN up to 100 000 steps with Mel-spectrograms generated from Tacotron 2 (2.3.2) improved MOS scores.

This model is deployed in several end-to-end systems, such as VITS (2.5.1).

2.5 End-to-End Models

This section deals with end-to-end models – a type of system that does not use explicit intermediate representations, such as spectrograms or Mel-spectrograms. They are more robust to errors from each component since all components are trained jointly. Additionally, it does not require as much work to design as in the case of two-stage models, which involve separately training and optimizing multiple components, such as phoneme generation, acoustic features extraction, and a vocoder.

2.5.1 VITS

First, a brief overview of the VITS model is presented. Next, the underlying theory of variational autoencoders as a main building block is described. This is followed by an architecture description. Finally, a summary of ablation studies and MOS results are provided.

General overview

VITS [21] is a non-autoregressive end-to-end model producing audio samples from a phoneme input sequence. To convey efficient end-to-end training, a VAE (**V**ariational **A**uto-**e**ncoder) is utilized. The use of latent variables and VAE facilitated a connection between text-to-spectrogram (section 2.3) and spectrogram-to-audio (section 2.4) modules. Normalizing flows applied on prior distribution further help to enhance the synthesis process in order to generate high-quality audio. To synthesize speech with various rhythms, durations, or pitches, a stochastic duration predictor is proposed.

As written in [21], VITS works as a „conditional VAE aiming to maximize the variational lower bound (ELBO) of the intractable marginal log-likelihood of data $\log p_\theta(x|c)$“. An explanation of concepts like conditional VAE and ELBO is provided in the following two subsections.

Basics of Variational Autoencoders

As stated in [42], a VAE is an autoencoder that applies regularisation to the encoding distribution during training, to guarantee that the latent space provides desirable properties enabling the generation of new data. To describe the basic ideas behind autoencoders, several concepts must be explained.

When encoder-decoder architecture is considered, the encoder takes data from the initial space and compresses them to the encoded (latent) space. The decoder decompresses them while preserving as much information as possible. This process is called dimensionality reduction.

However, there is nothing that forces the autoencoder to organize points in the latent space. That is where variational encoders help to regularise this undesired behavior. In addition to a standard autoencoder, a regularisation of the latent space during the training process is proposed. Input is not encoded as a single point, but rather as a distribution across the latent space. These distributions are then constrained to be close to a standard normal distribution. This enforcement is achieved using a regularisation loss, usually KL-divergence. A regular latent space guarantees that nearby points in the latent space produce similar outputs, and a point sampled from this latent space yields relevant content when decoded.

Once the model is trained, a point from a distribution in the latent space is sampled to produce new content.

Varitional Inference and Evidence Lower Bound

In this section, the underlying mathematical concepts of VAEs are described.

As [42] states, in order to generate a new value of x, the following process is executed: Firstly, latent representation z is sampled from the prior distribution $p(z)$. Secondly, data x is sampled from the conditional likelihood distribution $p(x|z)$. This idea can be described using Bayes theorem as

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}. \tag{2.9}$$

However, the denominator $p(x)$ is usually intractable. Thus, the computation of $p(z|x)$ requires approximation methods such as variational inference. The variational inference provides an approximation of complex distributions. This approach consists of selecting a parametrized family of distributions Q (for example the family of Gaussian distributions) and searching for the most suitable approximation of the target distribution within this family [40].

To find posterior probability $p(z|x)$, a surrogate $q(z)$, sharing similar properties, is employed. To calculate $q(z)$, a function $q \in Q$ defining this distribution has to be found. This can be written as

$$q^*(z) = argmin_{q(z)\in Q}\big(KL(q(z) \mid\mid p(z|x)\big), \tag{2.10}$$

where KL-divergence computation of $q(z)$ and $p(z|x)$ is as follows

$$KL\big(q(z) \;||\; p(z|x)\big) = \mathbb{E}_{z\in q(z)}\left(\log \frac{q(z)}{p(z|x)}\right), \tag{2.11}$$

where $\mathbb{E}_{z\in q(z)}$ denotes expectation over z distributed according to $q(z)$. However, the posterior $p(z|x)$ is not available at this point. After a series of modifications, intractable computation in equation 2.11 can be interpreted as

$$\begin{aligned} KL\big(q(z) \;||\; p(z|x)\big) &= -\mathbb{E}_{z\in q(z)} \log\left(\frac{q(z,x)}{q(z)}\right) + \log\big(p(x)\big) \\ &= -\mathcal{L}(q) + \log\big(p(x)\big). \end{aligned} \tag{2.12}$$

Data point x is a fixed value[13]. Then, the $p(x)$'s range of values lies in $\langle 0, 1\rangle$ and *log* of a number smaller than one is negative, therefore, $\log p(x)$ is a negative number called evidence. Since KL is a non-negative number, the value of $\mathcal{L}(q)$ must be negative and smaller than the evidence. In other words, $\mathcal{L}(q)$ is called ELBO (**E**vidence **L**ower **Bo**und). By increasing the ELBO, the result of KL-divergence decreases and vice versa.

By converting an intractable computational problem into an optimizing problem using variational inference, the posterior probability $p(z|x)$ can be obtained.

Application of Variational Inference and Evidence Lower Bound

Having explained VAE and ELBO, VITS can be characterized using the formula

$$\log p_\theta(x|c) \geq \mathrm{E}_{q_\phi(z|x)}\left[\log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p_\theta(z|c)}\right], \tag{2.13}$$

where

$p_\theta(x|c)$ an intractable marginal likelihood of data point x given condition c,
$p_\theta(z|c)$ a prior distribution of the latent variables z given condition c,
$p_\theta(x|z)$ a likelihood function of a data point x,
$q_\phi(z|x)$ an approximate posterior distribution.

Colored labels correspond to the boxes in the architecture in figure 2.10, where x denotes speech, c is text, and z is a latent representation of a spectrogram.

[13] In the case of VITS, x is an audio input.

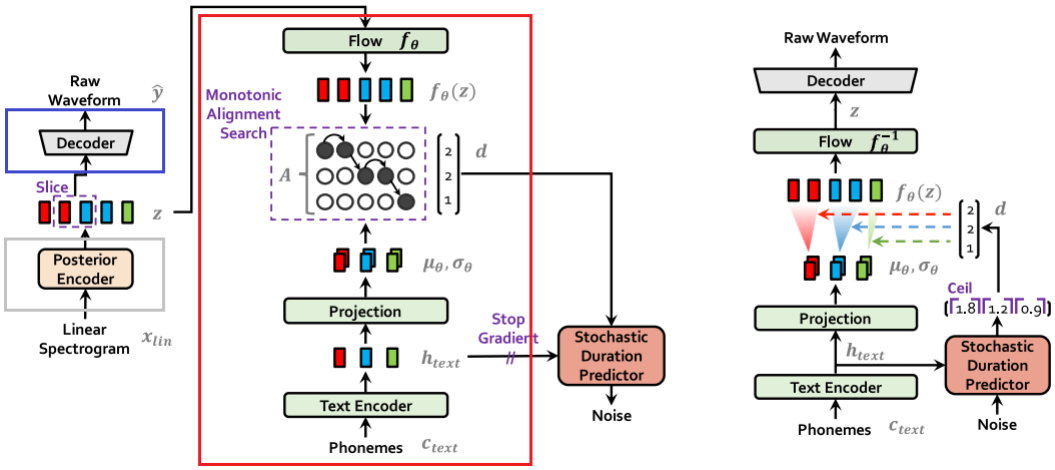

Figure 2.10: Training (left) and inference (right) procedure of VITS [21]. The architecture of the model consists of a posterior encoder (only during training), prior encoder, decoder, discriminator (only during training), and stochastic duration predictor.

In addition to a standard VAE described earlier, VITS uses conditional VAE, which only expands the original computation by conditions. The ultimate goal is to maximize the lower bound (the right side of the inequality), by which the true log-likelihood of the data (the left side) increases as well.

Specifically, x represents audio input and c input text. Then, $p_\theta(x|c)$ describes how well the model fits the speech on a given text input. The bigger this value is, the better.

Term $p_\theta(x|z)$ measures how well the decoder can reconstruct the original data x from the latent variable z. Again, the goal is to reconstruct as much information from the latent space as possible, thus, the aim is to maximize this term.

Next, the KL-divergence term $\left[\log \frac{q_\phi(z|x)}{p_\theta(z|c)}\right]$ provides information about how well the approximate posterior distribution matches the prior distribution. By minimizing this fraction, the model is encouraged to learn a posterior distribution that is similar to the prior distribution.

Model Architecture

VITS is built on GlowTTS (2.3.4). The overall architecture is depicted in figure 2.10.

The posterior encoder uses non-causal WaveNet residual blocks, consisting of dilated convolutions, gated activation units, and skip connections (described in section 2.4.2).

The prior encoder consists of two parts. First, the main part is a Transformer-based (described in section 2.3.3) text encoder, allowing the extraction of hidden representation h_{text} from input text c_{text}. Using the linear projection layer above the text encoder, the mean and variance of the prior distribution are obtained. Second, a normalization flow (described in section 2.3.4) follows similar architecture with residual blocks as the posterior encoder. It helps with the expressiveness of the prior distribution, which then enhances the quality of the generated samples.

The decoder (generator) and discriminator are taken from HiFi-GAN (V1) described in section 2.4.3. The adversarial training method described in section 2.4.3 is used here as well.

Aligning the latent speech variables with input text utilizes MAS. A description of this method can be found in section 2.3.4.

Instead of training a deterministic duration predictor, an SDP (**S**tochastic **D**uration **P**redictor) is designed. SDP is able to predict a phoneme duration based on the hidden representation h_{text} of input text c_{text}. The deterministic approach fails to generate a diverse speaking rate for a given speaker, whereas the SDP estimates the durations of given phonemes. SDP reflects the fact that in the real world, a given word is never uttered with the same duration. SDP consists of residual blocks with dilated convolutional layers.

Losses

During the training phase, several losses are utilized.

First, the reconstruction loss measures how well the model is capable of reconstructing the input data. Reconstruction loss is calculated as

$$L_{recon} = ||x_{mel} - \hat{x}_{mel}||_1, \tag{2.14}$$

where x_{mel} denotes Mel-spectrogram of the original audio and $\hat{x}_{mel}$ is the generated Mel-spectrogram.

Next, the KL-divergence measures the difference between the prior encoder and the posterior encoder probability distributions as follows:

$$L_{kl} = \log q_\phi(z|x_{lin}) - \log p_\theta(z|c_{text}, A), \tag{2.15}$$

where the input of the prior encoder consists of phonemes (c_{text}) extracted from the input text and alignment (A) between phonemes and latent variables (z). The posterior encoder accepts a linear spectrogram (x_{lin}) as a representation of target speech instead of Mel-spectrograms with the aim of maximizing the resolution of information. In other words, this loss measures how well the prior encoder can generate latent variables z solely from input text c_{text} and alignment A compared to the latent variables of the ground-truth linear spectrogram x_{lin}.

VITS further uses losses $L_{adv}(G), L_{adv}(D)$, and $L_{fm}(G)$ that are detailed in equations 2.5, 2.6, and 2.8, respectively.

Speech Synthesis Quality and Ablation Studies

To evaluate speech quality, MOS (described in section 4.5) tests were carried out on a model trained for 300 000 steps using the LJSpeech dataset. The results confirm that using the stochastic duration predictor leads to generating better-sounding samples.

Table 2.3: MOS results of the models [21]. DPP signals using the deterministic duration predictor instead of the stochastic version, F-T indicates fine-tuning. Ablation studies indicate that not using normalizing flow considerably influences results. Furthermore, using Mel-spectrograms instead of the linear-scale spectrogram as input for the posterior encoder during the training stage shows a decrease in quality as well – VITS can benefit from high-resolution data.

Model	MOS
Ground Truth	4.46 (±0.06)
Tacotron 2+HiFi-GAN	3.77 (±0.08)
Tacotron 2+HiFi-GAN (F-T)	4.25 (±0.07)
GlowTTS+HiFi-GAN	4.14 (±0.07)
GlowTTS+HiFi-GAN (F-T)	4.32 (±0.07)
VITS (DPP)	4.39 (±0.06)
VITS	**4.43 (±0.06)**
without Normalizing flow	2.98 (±0.08)
with Mel-spectrogram	4.31 (±0.08)

Table 2.4 provides information about the speed of synthesizing 1 second of speech using Google Colab both on CPU and GPU[14].

Table 2.4: Comparison of synthesis speed in Google Colab based on sentence „Když jsem jim tenkrát sebral těch dvacet procent, tak jim to také přišlo moc“. Using GPU for synthesis speeds up the generation process as expected.

	Speed
CPU	0.39 s
GPU	1.93 s

Due to the non-autoregressive[15] nature of sample generation, it cannot be used for real-time text-to-speech conversion.

Multi-speaker Text-to-Speech

For the prior and posterior encoder, a speaker embedding is added in residual blocks (so-called „global conditioning“ described in section 2.4.2). For the decoder and SDP, a new linear layer is added. The speaker embedding is then appended to the latent variables z (h_{text} in case of SDP).

Using the multi-speaker VCTK dataset [58] to train several models, VITS (MOS 4.38) outperforms models Tacotron 2 (MOS 3.19) and Glow-TTS (MOS 3.82).

Using the sentence „How much variation is there?“, several samples were generated to show how many different lengths of speech and different speech characteristics can the model generate. The results are shown in figures 2.11, 2.12, and 2.13.

[14] Google Colab offers NVIDIA Tesla T4, Intel(R) Xeon(R) CPU @ 2.00GHz, and 12 GB of RAM.
[15] The whole input text sequence must be available prior to synthesis.

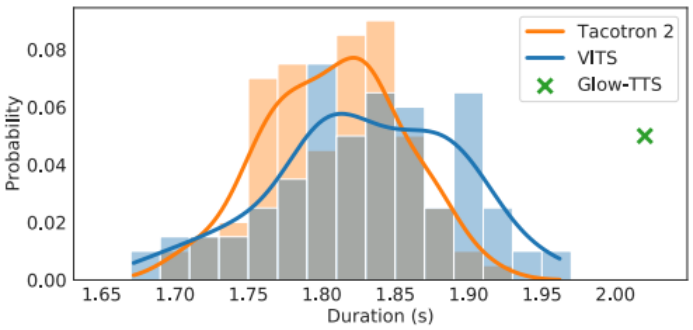

Figure 2.11: Comparison of audio duration [21]. The plot shows that the distribution of VITS sentence lengths closely resembles that of Tacotron 2, indicating its ability to produce varying audio lengths. However, Glow-TTS employs a fixed duration predictor, resulting in a constant sample length.

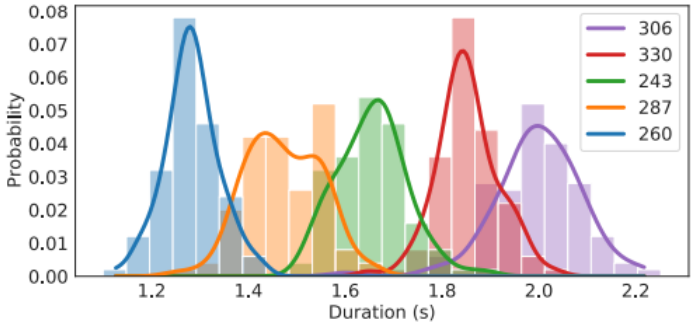

Figure 2.12: Comparison of audio duration [21]. This figure shows 100 utterances of each of the 5 selected speakers, confirming that the model can learn speaker-dependent phonemes durations.

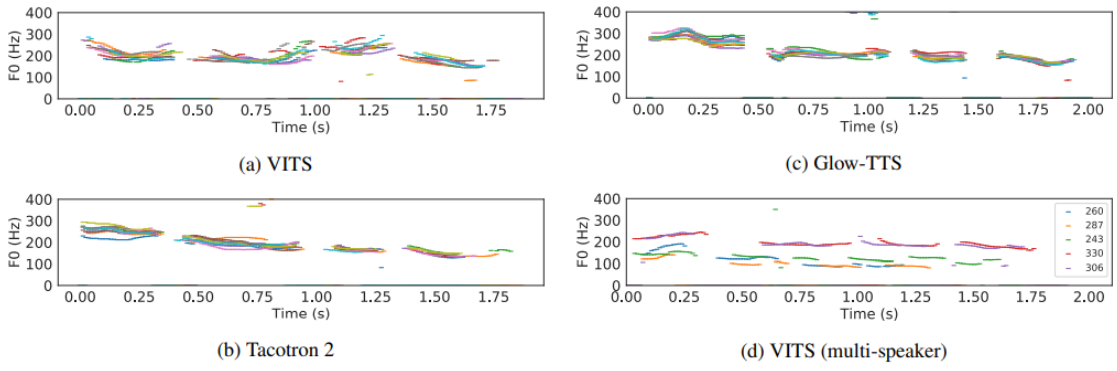

Figure 2.13: The figure demonstrates how 5 samples from 5 different speakers can be diverse in pitch [21].

YourTTS

YourTTS [7], a model derived from VITS, incorporates a speaker encoder model to compute speaker embedding vectors (d-vectors) that carry speaker-specific information. The speaker encoder is trained independently. These d-vectors are then used to condition the generated speech samples on external speaker embeddings, allowing the model to produce personalized speech outputs. The extended architecture can be found in figure 2.14.

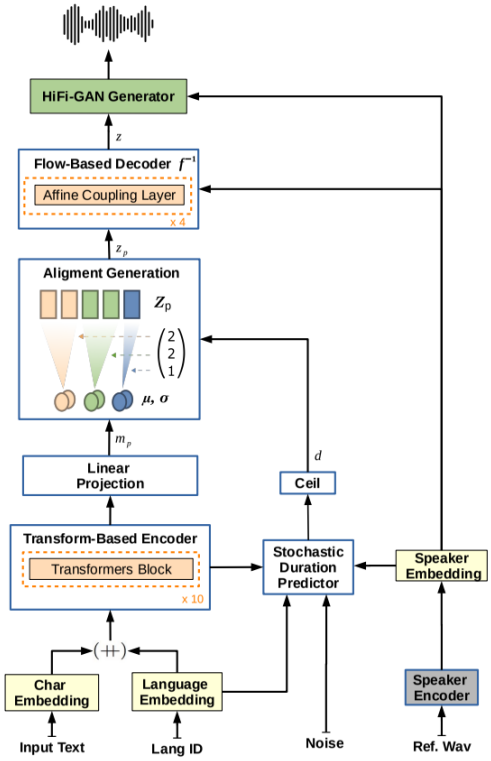

Figure 2.14: Inference procedure of YourTTS [7] version of VITS. The speaker encoder provides additional information to the SDP, decoder, and generator. This enables the generation of speech conditioned on input d-vector. Other minor architecture changes can be found in paper [7].

YourTTS version of VITS was chosen as a model used in this thesis as it offers end-to-end training and produces high-quality audio samples conditioned on input speaker embedding. The models were trained using the Coqui-TTS toolkit[16].

[16] https://github.com/coqui-ai/TTS

Chapter 3

Datasets

This chapter provides an overview of the available Czech datasets and examines their strengths and weaknesses. First, an overview of desired TTS dataset properties is proposed. Next, two corpora are analyzed. The last subsection introduces the developed male and female datasets for personalization.

3.1 What Makes a Good TTS Dataset

LibriTTS [62] provides a comprehensive overview of the topic of what makes a dataset suitable for TTS training purposes. LibriTTS is a filtered version of the LibriSpeech corpus. The original LibriSpeech corpus consists of read English speech and it was specifically made for training and evaluating ASR systems. It contains more than 1000 hours of audio with a sampling rate of 16 kHz [30]. The issues of using LibriSpeech for TTS systems were addressed in LibriTTS and can serve as a great source of information for preprocessing other datasets.

Low Sampling Rate

Having source audio with a high sampling rate is crucial to train a high-quality TTS system. Usually, a 16 kHz sampling rate is enough for training an ASR, while for TTS, audio files with a sampling rate of 24 kHz (or higher) are preferred.

Inappropriate Audio Segmenting

The TTS systems cannot be trained on hours-long audio files. Instead, these source files are divided into shorter ones and accompanied by their transcriptions. This segmenting can be done on an audio-signal level, which can lead to splitting sentences based on a silence longer than a threshold, completely ignoring the sentence breaks. This approach is not ideal for TTS training, instead, the audio files must be split at sentence breaks in order to preserve sentence-level prosody.

Upper-cased Text Without Punctuation, Non-standard Words

Capitalized letters and punctuation marks are useful for learning the emphasis of words, as well as the length of pauses. Special expressions like abbreviations, numbers, and currency symbols should be text-normalized.

Background Noise & Surrounding Silence

Noisy utterances should be filtered out. After computing the SNR, a threshold should be set, by which noisy audio files are eliminated from the dataset. Silence at the beginning and/or at the end of the audio file serves no purpose and should be cut off as well.

Downsides of Filtering

After filtering the dataset, fewer data will be available. If the corpus is large enough, this consequence causes no issues. On the other hand, it may interfere with the gender distribution of the speakers, which could lead to worse results later on.

3.2 Current options of available Czech datasets

The Czech dataset options available for training TTS systems are relatively limited. Considering the need for a large corpus of data, transcribed with minimal errors, and clean audio files with high bit-rate, finding an open-source dataset presents a challenge.

However, there are a few crowd-sourced options available, one of which is the Common Voice dataset [3]. Although the Czech language is included, the audio files are stored in a compressed .mp3 format and suffer from significant noise. Furthermore, the dataset contains a large number of speakers without proper labeling, making it difficult to determine the gender distribution. Given these limitations, the dataset is not ideal for training a TTS system.

Another source of speech samples could be telephone conversations, such as Vystadial 2013 – Czech data [24]. However, it must be taken into account that such a dataset might provide poor audio quality, making it unsuitable for TTS training purposes.

The plenary sessions of the Czech parliament provide a better source for training a TTS system, as they are publicly available, recorded daily, and semi-manually transcribed. Additionally, the quality of the microphones is sufficient and the sampling rate is consistently 48 kHz. Parliamentary hearings have been a valuable resource for speech processing applications for some time now, as noted in [23].

3.3 Large Corpus of Czech Parliament Plenary Hearings

The Large corpus[1] [25] consists of 444 hours of speech and its corresponding transcriptions. The corpus was built on data from November 2017 till November 2019 and they are available on the website of the Chamber of Deputies[2].

Using political utterances as a dataset is not uncommon in other countries, and there exist several datasets in foreign languages that use the same source of data, as in the case of The multilingual corpus for speech translation of parliamentary debate [19].

The audio files were segmented on an audio-signal level, with no regard to sentence punctuation, using the Kaldi speech recognition toolkit [32]. For ASR training purposes, this form of segmentation might not be such an issue. However, for training a TTS model, it is crucial to establish a link between a certain symbol in a sentence (., ?, !) and the change of speech melody.

[1] This corpus will be further addressed as Large Corpus.

[2] psp.cz/eknih/2002ps/audio/2006/01/17/index.html

While the transcriptions with speaker identifications for the original (long) audio files were available, segmented audio files did not offer any. Mapping the speaker information on the segmented audio files would be a very time-consuming process. One of the approaches could be identifying the speakers using a clustering algorithm. First, speaker embeddings of all the samples were computed. This was achieved using `SpeechBrain` [36] toolkit with a pretrained `ECAPA-TDNN` model [11]. It can be assumed that opting for 190 clusters, each representing a unique speaker, would be an adequate choice. To visualize embeddings with centroids, t-SNE[3] was deployed as shown in figure 3.1.

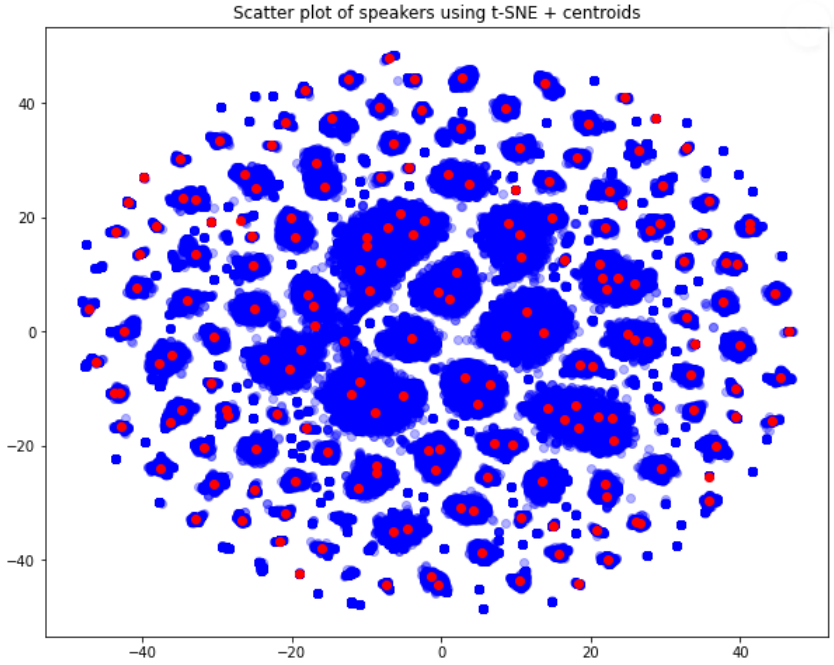

Figure 3.1: Some speakers in the dataset were evidently more active than others. By employing this naive categorization method with the k-means algorithm, there were inevitably some instances of overlapping with other speakers. It should be noted that identifying speakers using k-means obviously does not offer accurate results.

Table 3.2 provides an overview of the corpus statistics along with a comparison to another dataset. However, as table 3.3 shows, this corpus does not offer desirable properties for training a TTS model.

[3] https://medium.com/analytics-vidhya/visualising-embeddings-with-t-sne-b54bf6b635f

3.4 ParCzech 3.0: A Large Czech Speech Corpus with Rich Metadata

The ParCzech corpus [23] consists of hearings from the Chamber of Deputies from November 2013 to April 2021. Altogether, this provides more than 250 GB of well-organized training data. Each version of ParCzech offers improved cleaning processes and additional metadata, with version 3 being currently the latest.

A great challenge in creating a dataset of public sessions is the audio-transcription alignment. The recordings were segmented into sentences, mostly not longer than 20 seconds, which serve the needs for TTS training purposes.

Compared to the Large Corpus described in section 3.3, this dataset advanced in processing the transcriptions done by stenographers. Not only information about the person speaking was gained, but also personal information about the speaker, which further helped with enlarging the metadata.

When aligning speech with the appropriate transcriptions, the recordings were first segmented on a sentence-level. For each word in the transcription, its time span was calculated. Then, word time spans were extracted from the audio itself using an ASR. The next step was to find a match between the transcription and the words present in the audio file. Each word was treated as a unit – if the words did not match, a penalty was given, otherwise, a reward was given. The dataset was filtered based on these results.

Each audio file folder includes an ASR transcription, original transcription (with punctuation), word information, statistics for data analysis, and a speaker's name. Not only the ID of a speaker is provided, but also a text file for the whole dataset with speakers' information is available. It is possible to derive statistics about the gender and age distribution, which might be useful for experiments.

The audio files were separated into three sets: the training set, the development set, and the test set. Furthermore, the development and test sets were each divided into three subsets:

The Segments subset was created by random sampling from the filtered data.

The Context subset consists of speakers present in the training data. This subset served as a suitable resource for calculating objective metrics such as WER, CER, and speaker similarity in section 5.1 on seen speakers.

The Speakers subset is suited for experiments on unseen speakers. Data in these subsets were extracted from cleaned data, considering a balanced representation of men and women. Speakers subset was used in section 5.1.3 to assess the quality of generated samples on unseen speakers.

The dataset's subsections statistics are presented in table 3.1.

Table 3.2 compares the statistics of both datasets.

Although both datasets comprise enough data, table 3.3 compares both corpora in terms of the features that are important for a TTS dataset, described in section 3.1.

Table 3.1: Distribution of recordings in the ParCzech subsets [23].

	Files	**Hours**	**Audio Duration**	**Unique Speakers**
Train	579 169	1 271	7.9 s	417
Speakers Dev	4 596	10.7	8.4 s	30
Speakers Test	4 261	10.6	9.0 s	30
Context Dev	4 556	10.0	7.9 s	186
Context Test	4 868	10.0	7.4 s	186
Segment Dev	4 575	10.0	7.9 s	301
Segment Test	4 515	10.0	8.0 s	291

Table 3.2: Large Corpus of Czech Parliament Plenary Hearings [25] and ParCzech corpus statistics [23]. Both datasets provide more than a sufficient amount of data for training a TTS system.

	Large Corpus	**ParCzech**
Audio [h]	444	1 332
Audio files	191 455	606 540
Duration range [s]	0.2–44	0.82–53.99
Unique speakers	212	474
Total words	3 029 646	10 146 591
Time period	11/2017-11/2019	11/2013-04/2021

Table 3.3: This table provides a comparison of the features of discussed datasets. The first dataset obviously does not meet the requirements for a good TTS dataset.

	Large Corpus	**ParCzech**
Sampling rate [kHz]	16	48
Audio segmenting	audio-level	sentence-level
Original & ASR transcriptions	both	both
Background noise, silence	present	partially filtered

The ParCzech dataset was chosen as the main corpus for training a base TTS model. Due to HW limitations, only a certain subset of training data was selected. Table 3.4 summarizes the statistics of the used data.

Table 3.4: As the corpus lacked statistics based on years, these data were manually calculated. The number of hours was estimated based on the audio average duration.

	ParCzech
Audio [h]	~545
Audio files	248 437
Unique speakers	255
Male speakers	207
Female speakers	48
Time period	01/2018-04/2021

3.5 Personalized Datasets

To create a personalized dataset, several interviews were collected. The interviews were separated into chunks of speech where the targeted speaker speaks. This process is called diarization and `Pyannote` [5, 4] with a pretrained model[4] was used for this task. To classify the segments of an audio file by speaker, speaker embeddings are repeatedly computed within small time windows. The final output is a text file that lists the individual time frames, in seconds, along with the corresponding speaker ID. Using these time frames, the original long recordings were split into many shorter ones.

After creating new audio files, transcriptions needed to be created for each of them, following the ParCzech data structure. For both tasks, a modified version of `Whisper` [26, 13] was utilized using its `large-v2` pretrained model. Compared to the original version of `Whisper` [34], there are several additional features available. Firstly, time spans are defined for each word in the analyzed audio file. The time spans are especially important for proper audio segmenting as shown in algorithm 2. Secondly, the so-called confidence (probability that the word is correctly guessed by `Whisper`) can be used to identify the transcriptions that might need further tweaking.

Algorithm 2 Sentence-level segmenting

```
function SentenceLevelSegmenting(words, audio)
    start ← words[0].start
    end ← words[0].end
    for word in words do
        if . in word then
            end ← word.end
            new_clip ← audio.clip(from = start, to = end)
            new_clip.save()
            start ← word.end
        end if
    end for
end function
```

This simplified pseudocode 2 demonstrates the main idea behind sentence-level segmenting. While the diarization process returned the speaker's clips, some of them were several minutes long and they needed to be further segmented on a sentence-level. The algorithm illustrates the processing of one of these audio files.

To obtain these time spans, word alignment based on DTW (described in section 4.3) is deployed on `Whisper`'s cross-attention weights after each decoding of a speech segment.

In general, it is necessary to choose interviews, where the quality of the microphones is similar to ensure consistent speech characteristics.

After creating a dataset using the automated pipeline, the data were manually filtered. This consisted of reviewing each audio file with its transcription and correcting possible typos that might have occurred during the transcription process. The data were then split into train, dev, and test subsets.

To obtain baseline values of metrics, audio samples generated using the original ParCzech model prior to fine-tuning were evaluated. Next, the original model was fine-tuned,

[4]The model can be accessed at https://huggingface.co/pyannote/speaker-diarization.

audio files were generated, evaluated, and the results were compared to assess the impact of fine-tuning on the model's performance (sections 5.2.1 and 5.2.2).

In light of recent political events, Petr Pavel was chosen as a male speaker. To cover both genders, the female dataset was developed using interviews with Danuše Nerudová, another candidate in the presidential elections. The reason behind using presidential candidates lies in the fact that they were frequently in the media spotlight and made numerous public appearances, which provided suitable sources for building a TTS dataset.

3.5.1 Male Dataset

Tables 3.5 and 3.6 provide an overview of the male dataset statistics.

Table 3.5: Male dataset statistics. Manual filtering involved deleting files with speech overlaps, incomprehensible words, bloopers, or numerous words repetition. The sampling rate is 44.1 kHz.

	Unfiltered	**Filtered**
Total duration [min]	112.84	87.78
Average text length [chars]	124.17	137.61
Average audio duration [s]	8.48	9.22
Number of words	99 086	78 573
Number of audio files	798	571

Table 3.6: Male dataset detailed statistics. The test subset represents around 15 % of the data of the whole dataset.

	Train	**Test**	**Dev**
Total duration [min]	73.14	12.59	2.05
Average audio duration [s]	9.6	7.4	10.27
Number of audio files	457	102	12

This dataset consists of 3 interviews:

1. Prostor X – Pavel: Situace na Hradě je horší, než jsem čekal, možná padne trestní oznámení [37]
2. STANDASHOW – Generál PETR PAVEL: Občas je třeba ukázat zuby a bránit demokracii... [50]
3. Insider #129 – Petr Pavel [17]

3.5.2 Female Dataset

Similarly to the male speaker, tables 3.7 and 3.8 show information about the female dataset.

Table 3.7: Female dataset statistics. Audio files with incomprehensible speech and speech overlaps were deleted. The sampling rate is 44.1 kHz.

	Unfiltered	**Filtered**
Total duration [min]	84.67	59.13
Average text length [chars]	110.03	129.31
Average audio duration [s]	7.36	8.55
Number of words	75 923	53 663
Number of audio files	690	415

Table 3.8: Female dataset detailed statistics. The test subset represents around 12 % of the data of the whole dataset.

	Train	**Test**	**Dev**
Total duration [min]	52.26	7.34	1.47
Average audio duration [s]	8.83	7.34	8.01
Number of audio files	355	60	11

This dataset consists of 2 interviews:

1. U Kulatého stolu – Danuše Nerudová: Babiš prezidentem by byl signál jít do politiky. Další kandidaturu nevylučuji [55]
2. Insider #130 – Danuše Nerudová [18]

Chapter 4

Evaluation Pipeline For New TTS Models

Many metrics can be employed to evaluate the quality of the model from various perspectives. In this work, a pipeline was created for evaluation, where each step tests a different aspect of the generated samples. By applying this pipeline to each model, it becomes easy to monitor and evaluate any progress.

It should be noted that the test data must be a subset of the dataset that was not used for training.

4.1 Word Error Rate & Character Error Rate

As written in [12], WER measures the percentage of words that are incorrect in a given sentence in comparison to the ground truth sentence. WER value is computed using formula 4.1.

$$WER = \frac{S + D + I}{S + D + H}, \tag{4.1}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, and H is the number of hits (correct words).

However, WER lacks an upper bound and cannot provide information about the quality of the system, it is only capable of indicating, which system performs better than the other. Additionally, WER is not symmetric with respect to insertions and deletions and gives more weight to insertions, which can potentially lead to larger errors in noisy environments.

To calculate WER, a ground truth (i.e., the actual speech transcript) and a hypothesis generated by an STT model are required. It is essential to choose a well-performing model trained in the target language to minimize WER errors. In this study, `Whisper` [34] and its `large-v2` pretrained model was utilized. `Whisper` reported a WER performance of approximately 13 % in the Czech language.

To facilitate the comparison of results across the trained models, WER is first calculated on the original test subset audio files. This provides a baseline WER that can be used to evaluate the WER results of the generated audio files. Next, the tested model is used to generate audio files with the same content and spoken by the same person as in the test subset.

CER goes hand in hand with WER. Its result indicates how many characters were incorrectly predicted in the given word. CER can be calculated using the same formula as in WER but at the character level.

The `jiwer`[1] library was used for both WER and CER. To ensure a fair comparison between the ground truth and hypothesis and avoid increasing the error rate, the text was cleaned by lowercasing, removing punctuations, and white spaces using the `jiwer`'s text preprocessing methods.

4.2 Mel-Cepstral Distortion

The aim of MCD (**M**el-**C**epstral **D**istortion) is to measure the similarity between two audio signals in their Mel-cepstra.

The formula for MCD computation was introduced in paper [54] and is defined as

$$MCD = \frac{10}{\ln(10)} \sqrt{2 \sum_{d=1}^{D} \left(mc_d^{(y)} - \hat{mc}_d^{(y)}) \right)^2}, \tag{4.2}$$

where term $mc_d^{(y)} - \hat{mc}_d^{(y)}$ compares the dth coefficient of the original and generated Mel-cepstra, respectively.

A spectral envelope, representing the overall shape of the frequency spectrum of the signal [46], must be obtained. MCEPs (**M**el-**C**epstral **C**oefficients) are then computed from the spectral envelope. For both tasks, SPTK (**S**peech **P**rocessing **T**oolkit) was used. Further information can be found in the SPTK documentation[2].

Library Function

To compute MCD, `pymcd`[3] library described in [8] was used. The original and generated audio files were amplitude normalized[4] and down-sampled to 16 kHz. The library offers 3 modes for MCD computation: MCD-plain, MCD-DTW, and MCD-DTW-SL. MCD-DTW allows for the comparison of spectrograms of speech samples that may not be time-aligned, which is often the case. MCD-DTW-SL[5] additionally penalizes different durations of speech. The FastDTW [44] implementation used in `pymcd` was compared to DTW from the `Librosa` library, and their results were found to be similar.

To better understand the results, a ground truth baseline is set in figure 4.1. From two[6] speeches, an identical sentence from various years was extracted and adjusted to be as similar as possible in terms of duration and amplitude.

[1] https://github.com/jitsi/jiwer
[2] https://sp-tk.sourceforge.net/
[3] https://github.com/chenqi008/pymcd
[4] https://github.com/jiaaro/pydub/issues/90
[5] MCD-DTW-SL will be further referenced as MCD.
[6] https://youtu.be/KSRFKVQ8elE, https://youtu.be/WvvO08FmP_g

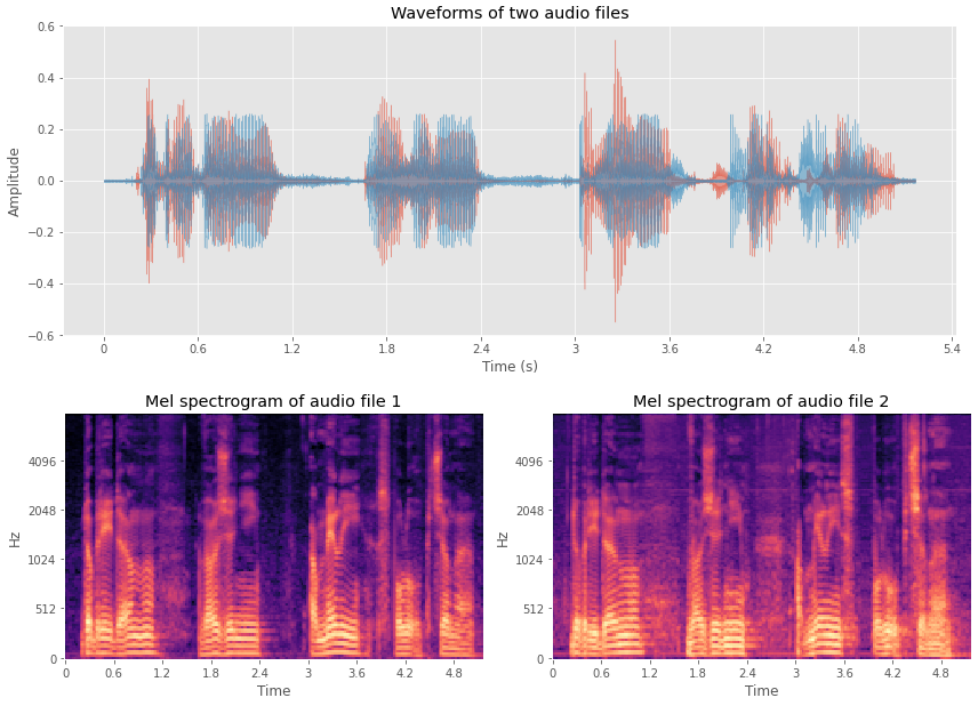

Figure 4.1: Mel-spectrograms and waveforms of the reference samples containing the sentence „Dobrý den, vážení a milí spoluobčané.“. It can be observed that the second sample exhibits a higher level of noise. While comparing the reference sample to itself resulted in the value of 0 as expected, both MCD-DTW and MCD-DTW-SL for these two samples were equal to 9.05.

4.3 Dynamic Time Warping

DTW determines the optimal alignment between the points of a reference and a test sequence. Ideally, an aligner should be able to linearly adjust a tested word to one from the dictionary of reference vectors representing all possible words. However, words are never uttered in the same way, and thus, they cannot be represented by a single vector. Since DTW aligns vectors non-linearly, it provides a robust solution for utterances of the same words yet diverse in speed [64].

First, the distance between each pair of vectors of the reference and target speaker is calculated. This value is added to the minimum cumulative distance from the previous neighboring cells. The resulting matrix consists of values that denote a distance from the start. To find the optimal alignment path, DTW backtracks through the matrix, choosing the path with the lowest cumulative cost. Algorithm 3 could be used to compute a matrix for finding a minimal warping path.

Algorithm 3 DTW algorithm [47]

```
1: function DTWDistance(ref_array [1..n], test_array [1..m])
2:     DTW ← array [0..n, 0..m]                          ▷ set DTW as 2D matrix
3:     for i ← 0 to n do
4:         for j ← 0 to m do
5:             DTW[i, j] ← ∞          ▷ set bottom row and left column to infinity
6:         end for
7:     end for
8:     DTW[0, 0] ← 0                                          ▷ set [0, 0] to 0
9:     for i ← 1 to n do
10:        for j ← 1 to m do                ▷ loop through matrix and compute cost
11:            cost ← d(ref_array[i], test_array[j])
12:            DTW[i, j] ← cost + min(DTW[i − 1, j], DTW[i, j − 1], DTW[i − 1, j − 1])
                                                    ▷ insertion, deletion, or match
13:        end for
14:    end for
15:    return DTW[n, m]          ▷ the final cost is the value in the top right corner
16: end function
```

Function $d(x, y)$ calculates a distance between the points of sequences as $|x-y|$. Function $DTW[i, j]$ denotes a distance between $r[1:i]$ and $t[1:j]$ with the best alignment.

DTW is utilized during the MCD computation when comparing two time-unaligned Mel-spectrograms, i.e., MCEPs. The minimal warping path mean divided by the number of frames provides the final MCD result. Another use-case of DTW is in the dataset development pipeline (described in section 3.5).

4.4 Verification

In general, the task of speaker verification determines whether two audio samples were uttered by the same or a different speaker. In this work, speaker verification serves to verify the identity of the speaker in the audio sample generated by the TTS system. One of the possible approaches is to compute a speaker embedding[7] of the original and generated audio samples and compare them using a cosine distance metric. Speaker embeddings were extracted using the `SpeechBrain` [36] toolkit with a pretrained `ECAPA-TDNN` model [11]. To decide, whether the generated audio sample sounds close enough to the target speaker, a threshold must be defined as shown in figure 4.2.

4.4.1 Cosine Distance

Cosine distance is derived from cosine similarity [16], which measures how similar two vectors are. In terms of mathematical representations, it defines an angle between two vectors. The formula is defined as follows:

$$cosine\_similarity(A, B) = \cos\theta = \frac{A \cdot B}{||A||\,||B||}, \tag{4.3}$$

[7] Speaker embedding is a vector uniquely identifying speaker's acoustic characteristics.

where A and B are speaker embeddings and $||A||$ is the Euclidean norm of the vector[8]. A result of -1 signalizes vectors with no match. The smaller the angle between vectors, the closer the value is to 1.

Cosine distance can be then calculated as

$$cosine\_distance(A, B) = 1 - \cos\theta = 1 - \frac{A \cdot B}{||A||\,||B||}. \tag{4.4}$$

The range of possible results is from 0 to 2, where lower values indicate that the audio sample has acoustic properties similar to those of the target speaker.

Figure 4.2 shows an example of target and non-target trials based on the ParCzech development subset.

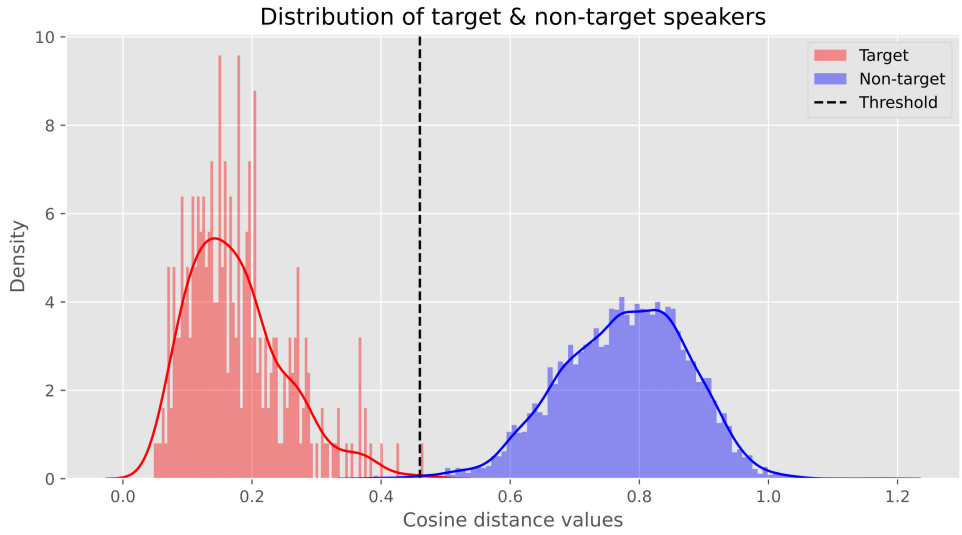

Figure 4.2: Distribution of target and non-target trials. The point, where the target and non-target lines intersect, is the optimal threshold (highlighted with a black dashed line). Results below this value increase the acceptance rate.

An optimal threshold needs to be determined to decide, whether an audio file sounds similar enough to the target speaker. The threshold corresponds to the value of cosine distance, where EER (**E**qual **E**rror **R**ate) occurs – at this point, the rate of the wrongly classified target (same speaker) and non-target (different speaker) trials is the same. The threshold value in figure 4.2 is represented as a black dashed line and it is the point. Once the threshold is established, the aim of this metric can be applied – each generated audio sample's embedding is computed and compared to the mean embedding of the speaker; if the result of the cosine distance is smaller than the threshold, the audio sample has desirable properties and increases the overall acceptance rate[9]. Data used for the threshold calculation can be found in section 5.1.

[8] For vector $x = (x_1, x_2, \ldots, x_n)$ it is defined as $\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.

[9] AR will be used as an abbreviation for acceptance rate throughout this document.

4.5 MOS

As [59] states, MOS (**M**ean **O**pinion **S**cale) is a subjective evaluation metric for measuring the quality of speech systems. The MOS standard consists of several questions that can be answered on a scale from 1 to 5. The averaged results assess system quality in terms of naturalness and intelligibility. Even though [59] provides interesting conclusions based on psychometric findings, using such a complicated questionnaire (9 questions per sample) is quite time-consuming for the respondents. In addition, other TTS models use simpler methods to gain MOS results, such as in Voice Conversion Challenge 2020 [63].

Following recommendations from both [59] and [63], the test samples were randomly sampled from the dataset's test subset. This was further mixed with original samples and samples generated prior to fine-tuning. All samples were amplitude normalized and their sample rate matched to start with equal conditions. The resulting number of samples was 15, using 5 of each category. This way the results could be directly compared. As in [63], each sample was evaluated on a scale from 1 to 5 using the question „Listen to the following audio and rate it for quality, both in terms of naturalness and intelligibility. Some of the audio samples you will hear are of high quality, but some of them may sound artificial due to deterioration caused by computer processing. Please evaluate the voice quality on a scale of 1 to 5 from Bad to Excellent.“ Whereas the original paper proposed only to evaluate naturalness and ignore possible mispronunciations, the evaluation in this work is based on both factors.

More specific information about the number of respondents, their age, and sample statistics can be found in section 5.2.3.

Chapter 5

Implementation

The first part of this chapter describes the training of the base models, along with the issues that arose during this process. Following this, the models were fine-tuned to personalize them for the target speakers. In addition, the base models fine-tuned on a limited amount of data are presented as well. All models were evaluated using objective metrics, and the personalized models were also evaluated using subjective listening tests. Finally, the final user application is described.

5.1 Training the ParCzech 3.0 Dataset

This section describes training a model using data from the ParCzech dataset (3.4).

Firstly, a baseline for WER & CER was set on the original data. A few models were then trained on various subsets of the training data. Their performance was evaluated using the metrics pipeline. Lastly, the ability to adapt to unseen speakers was analyzed.

5.1.1 Baseline Evaluation

As stated in section 3.4, several subsets for development and testing are available.

For testing purposes, `parczech-3.0-asr-context.test` was used. First, speakers with the largest amount of audio files during training were selected. Extremely short audio files (usually repetitive greetings) were excluded. There were 20 speakers selected (17 men, 3 women), each with 20 audio files. Since the models were trained using d-vectors[1] (speaker embeddings), a mean speaker embedding was calculated for each speaker. Therefore, 5 out of 20 selected audio files were averaged out, resulting in one mean speaker embedding. These mean vectors were used for generating new audio samples.

While this would be sufficient for WER, CER, and MCD computation, verification compares the embedding of the generated audio file with the mean speaker embedding of the speaker, therefore, the same embedding cannot be utilized both for generating and comparing. To address this issue, 20 audio files from `parczech-3.0-asr-context.dev` were selected. Following the same processing pipeline as above, audio files from the same speakers were obtained, and mean embeddings were calculated. 15 unused embeddings along with the mean embeddings were then used for threshold calculation, which can be found as an example in section 4.4. Speaker embedding extraction was achieved using `SpeechBrain` [36] toolkit with a pretrained `ECAPA-TDNN` model [11].

[1] To generate speech from a particular speaker, a d-vector is used as an additional input. Models trained with d-vectors are also more controllable during fine-tuning.

To obtain a universal mean embedding, vectors were taken from various years.

To set a baseline for WER and CER, the results of the original data can be found in table 5.4.

5.1.2 Trained Models

For a better orientation in the following text, table 5.1 summarizes all trained models. The first column defines model abbreviations used in the text later.

Table 5.1: List of trained models and their brief description.

Model	Brief description
1M	Base model trained on short samples from ParCzech (3.4) for 1 million steps, data from 2018 to 2021.
200k	Base model trained on all samples from ParCzech (3.4) for 200 000 steps, data from 2018 to 2021.
1M(P)	Base 1M model fine-tuned on the male dataset (3.5.1) for additional 70 000 steps.
200k(P)	Base 200k model fine-tuned on the male dataset (3.5.1) for additional 70 000 steps.
200k(P20)	Base 200k model fine-tuned on a subset of the male dataset (3.5.1) for additional 20 000 steps.
200k(F)	Base 200k model fine-tuned only on the female samples from ParCzech (3.4) for additional 150 000 steps.
200k(N)	Base 200k model fine-tuned on the female dataset (3.5.2) for additional 70 000 steps.
200k(N+F)	Fine-tuned 200k model further fine-tuned on the female dataset (3.5.2) for additional 70 000 steps.

Additionally, the results of all the models are listed in table 5.2 to provide a good overview. The results are further discussed in the following text.

Table 5.2: This table summarizes the results of all trained models, which are further discussed in the following text. Highlighted models are the resulting personalized models.

Model	WER	CER	MCD	AR	MOS
Seen					
Original	13.38 %	5.01 %			
1M	28.64 %	11.29 %	13.01 dB	99.67 %	
200k	56.96 %	37.52 %	17.93 dB	81 %	
Unseen					
Original	10.03 %	3.46 %			
1M	28.11 %	10.78 %	12.65 dB	92.31 %	
200k	58.6 %	40.54 %	18.83 dB	39.6 %	
Unseen – Male					
1M	22.43 %	10.25 %	14.19 dB	98.04 %	1.68
200k	35.98 %	23.24 %	17.27 dB	79.41 %	
Fine-tuned – Male					
1M(P)	13.39 %	5.78 %	12.22 dB	100 %	4.12
200k(P)	12.47 %	5.35 %	12.07 dB	100 %	
200k(P20)	14.29 %	6.04 %	12.15 dB	100 %	
Unseen – Female					
1M	19.84 %	8.10 %	15.97 dB	21.67 %	
200k	34.09 %	22.18 %	18.36 dB	10 %	
200k(F)	9.76 %	3.81 %	15.63 dB	28.33 %	3.42
Fine-tuned – Female					
200k(N)	13.93 %	5.93 %	12.30 dB	100 %	
200k(N+F)	11.61 %	4.6 %	12.67 dB	100 %	3.02

The 1M model was the first trained model. By manually evaluating the results, some samples performed very well. On the other hand, a lot of them were incomprehensible. The model did not perform well with longer sentences. When analyzing the origin of this issue, the training script from the Coqui-AI[2] API contained an upper limit for text length, which restricted the model from using longer audio files for training.

Table 5.3 provides WER mean values based on input length.

Table 5.3: The mean WER values decrease as the number of characters in the input increases, consequently leading to a decrease in speech intelligibility.

Input length	**<100 chars**	**<200 chars**	**>300 chars**
Mean WER value	52.18 %	85.17 %	105.5 %

To resolve the issue of garbled speech when a longer text was passed as input, a new approach was taken. Based on the commas in the input sentence, the text was divided into several shorter text segments and each part was generated separately. The individual segments of the sentence were then combined into the resulting audio file.

As table 5.4 shows, a significant improvement can be seen in WER, CER, and speaker similarity results. The resulting audio files provided by no means perfect samples since

[2]The repository can be found at https://github.com/coqui-ai/TTS. It was used for training all models.

the duration of each part of the sentence differed depending on the number of characters. To enhance the naturalness of speech, more sophisticated heuristics would need to be developed.

Figure 5.1 presents the speaker similarity results for further analysis.

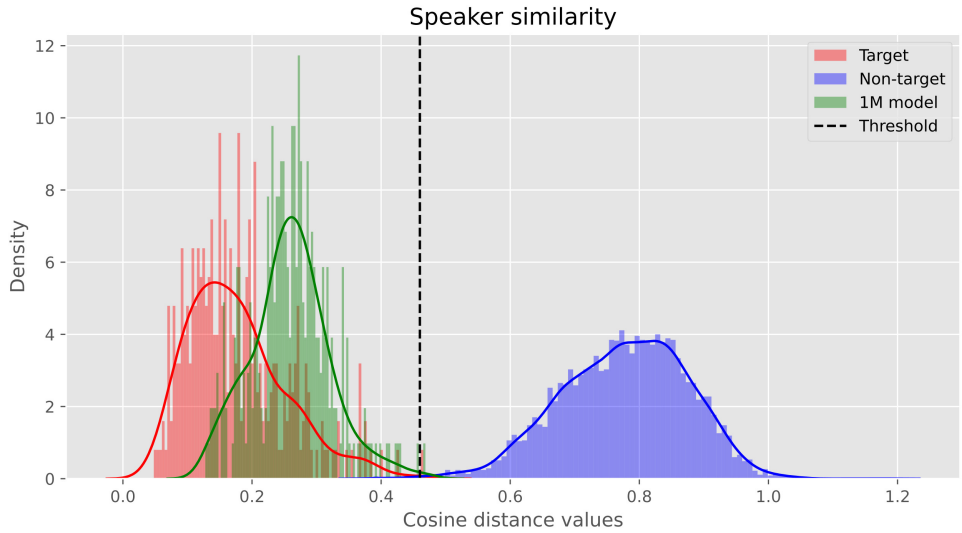

Figure 5.1: Speaker similarity results based on 20 speakers × 15 wavs = 300 samples. The mean value is 0.26 and the acceptance rate is 99.67 %. The test subset comprises 17 men and 3 women. The cosine distance matrix can be found as an example in section 4.4.

To improve multi-speaker performance, a possible solution could be training a new model on all corpus data, i.e., including the longer audio files. For almost the same amount of time (approximately 2 weeks) as the 1M model spent training, a new model trained for 200 000 steps using all data from 2018 to 2021 was created. The issue with intelligibility was not solved, as can be seen in table 5.4, and more training steps would probably be required. The most time-consuming part was resampling the data to the target sampling rate on the fly. Resampling the dataset prior to training could speed up the whole process of training, however, the main focus was on fine-tuning the models, not on creating the best possible multi-speaker system.

Table 5.4: The metrics of the 1M model, with and without segmentation method deployed, compared to the baseline WER & CER values and to the 200k model. A significant improvement can be observed in terms of speech intelligibility (WER, CER) for the 1M model. Even though the same segmentation approach was employed on the 200k model, it performed worse.

	Original	**1M (segmented)**	**1M**	**200k (segmented)**
WER	13.38 %	28.64 %	63.19 %	56.96 %
CER	5.01 %	11.29 %	37.64 %	37.52 %
MCD		13.01 dB	24.12 dB	17.93 dB
AR		99.67 %	97 %	81 %

5.1.3 Unseen Speakers

Tracking the metrics results for unseen speakers provides insight into the improvements achieved after fine-tuning. It is anticipated that the acceptance rate of audio generated using an unknown speaker embedding will be low. Data from `parczech-3.0-asr-speakers.test` and `parczech-3.0-asr-speakers.dev` were used in the same fashion as in section 5.1.1. These subsets are gender-balanced.

Table 5.5 provides the metrics results of the 1M and 200k models.

Table 5.5: Metrics of the 1M and 200k models calculated using subset with speakers unseen during training. Similarly to the seen speakers from the previous section, the 200k model performed poorly. Therefore, only the 1M model is further analyzed in this section.

	Original	**1M**	**200k**
WER	10.03 %	28.11 %	58.6 %
CER	3.46 %	10.78 %	40.54 %
MCD		12.65 dB	18.83 dB
AR		92.31 %	39.6 %

Figure 5.2 shows a comparison between the speaker similarity results of seen and unseen speakers.

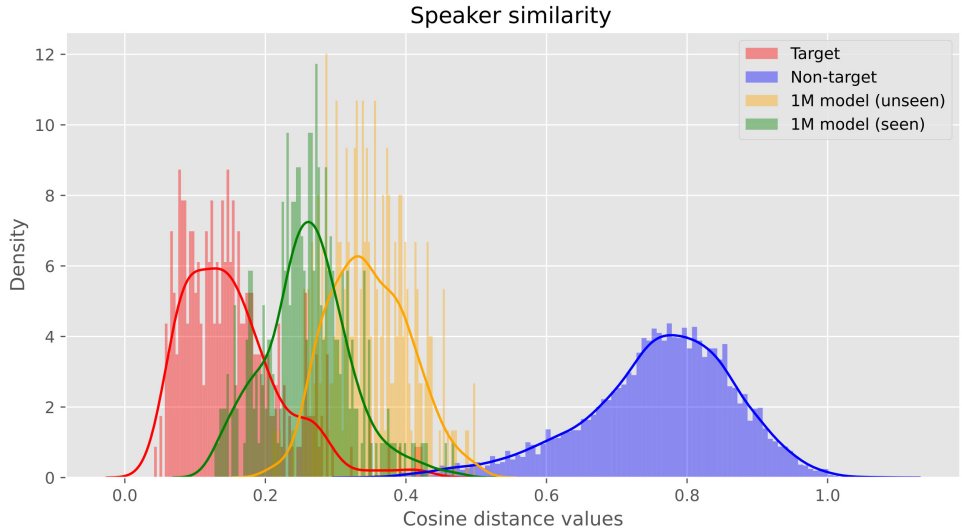

Figure 5.2: Unseen speaker subset is gender-balanced. The plot compares seen and unseen speakers in terms of speaker similarity distribution using 20 speakers not present during training. As anticipated, the acceptance rate is better with seen speakers. Target speaker overliers were manually analyzed, revealing that the speaker slightly elevated the vocal pitch, however, this does not affect the validity of the data. The orange curve of unseen speakers is shifted to the right, which can be expected since the model performs better on speakers seen during training.

Since the dataset used for training was not gender-balanced and women were not present as much as men, it could be expected that values for women would be slightly worse than for men. However, the results do not fully confirm this hypothesis as figure 5.3 shows.

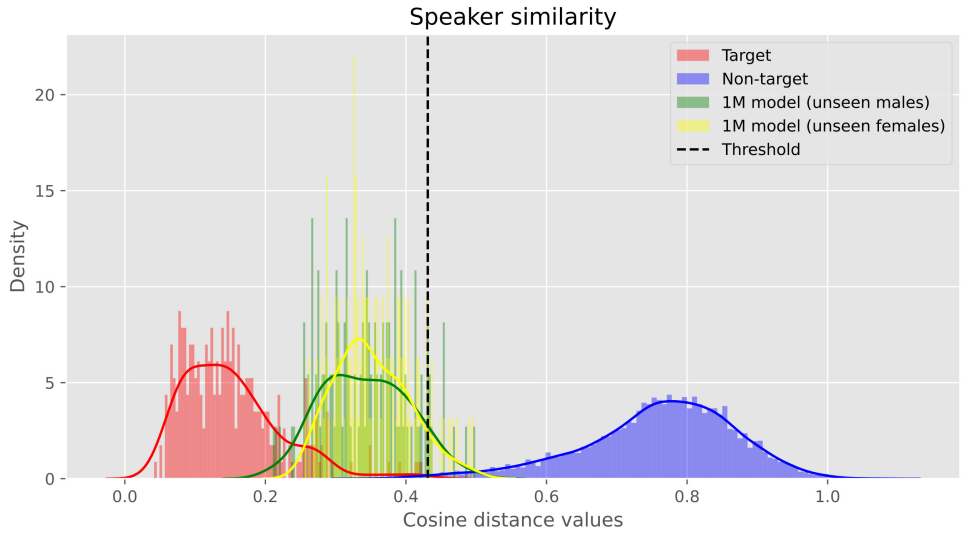

Figure 5.3: Speaker similarity of males and females. **Males**: The mean value and the acceptance rate were 0.34 and 92.19 %, respectively. **Females**: Despite the gender imbalance of the training data, the system is capable of generating women's voices. The mean value and the acceptance rate were 0.35 and 92.42 %, respectively.

Analyzing the results based on age did not uncover any other pattern.

In summary, based on the acceptance rate values, the 1M model is capable of generating both male and female voices with similar quality.

5.2 Personalization

This section deals with fine-tuning the base models 1M and 200k. The models were fine-tuned using the male dataset described in section 3.5.1. The same process was then applied using the female dataset described in section 3.5.2. The results were evaluated using both objective and subjective metrics.

5.2.1 Male Speaker

The generated samples prior to fine-tuning were evaluated as well as defining a baseline for WER and CER metrics. The results of the fine-tuned model are then discussed.

Prior to fine-tuning

Table 5.6 compares the unseen speaker adaptation of the 1M and 200k models.

Table 5.6: The 1M model adapts better to the unseen speaker than the 200k model. A comparison of the speaker similarity results between the 1M and 200k models can be found in the appendix (C.2).

	Original	**1M**	**200k**
WER	18.31 %	22.43 %	35.98 %
CER	11.78 %	10.25 %	23.24 %
MCD		14.19 dB	17.27 dB
AR		98.04 %	79.41 %

The segmentation method described in section 5.1.2 was used here for generating samples as well. While the WER results of the generated samples were slightly worse than the original values, the CER results outperformed the original data.

One of the problems was that `Whisper` [34] ignored filler words and transcribes informal words as formal[3]. The filler words and informal endings were more apparent in the generated data, therefore, `Whisper` could no longer ignore them. The speaker occasionally repeated some words of the sentence more than once, which could confuse `Whisper`. This phenomenon did not occur in the generated data. Similarly, the original speech was sometimes too fast, especially when the sentences were short as shown in table 5.7, whereas the generated samples did not suffer from the increased speech speed and the error was constant.

Table 5.7: Mean WER values of the 1M model conditioned on an unseen speaker embedding.

Input length	**<100 chars**	**<200 chars**	**>300 chars**
Original WER	23.43 %	11.72 %	13.91 %
Inferred WER	20.94 %	21.90 %	23.11 %

Fine-tuned model

Both 1M and 200k models were fine-tuned for another 70 000 steps using the male dataset (3.5.1). Table 5.8 provides a comparison of the metrics.

Table 5.8: Comparison of the 1M and the 200k fine-tuned models. The almost identical AR results of both models can be found in the appendix (C.3). Fine-tuning resolved the issues with garbled speech when a longer input text was passed. Therefore, the segmentation method was not utilized.

	1M(P)	**200k(P)**
WER	13.39 %	12.47 %
CER	5.78 %	5.35 %
MCD	12.22 dB	12.07 dB
AR	100 %	100 %

Based on table 5.8, both fine-tuned models perform similarly in terms of all metrics.

[3]For example „hezkej“ → „hezký“

Constrained training

A dataset of more than one hour of data may not always be available. To determine the amount of data required to create a well-performing model, an additional model was trained using approximately 11 minutes of data obtained from the original dataset. Table 5.9 compares this model with the best model as a reference. Figure 5.4 then compares the models in terms of speaker similarity.

Table 5.9: Comparison of the fine-tuned models of the male speaker. It can be seen that speech intelligibility increases with the number of training steps as expected. However, the results do not differ much from each other.

	200k(P20)	**200k(P)**
WER	14.29 %	12.47 %
CER	6.04 %	5.35 %
MCD	12.15 dB	12.07 dB
AR	100 %	100 %

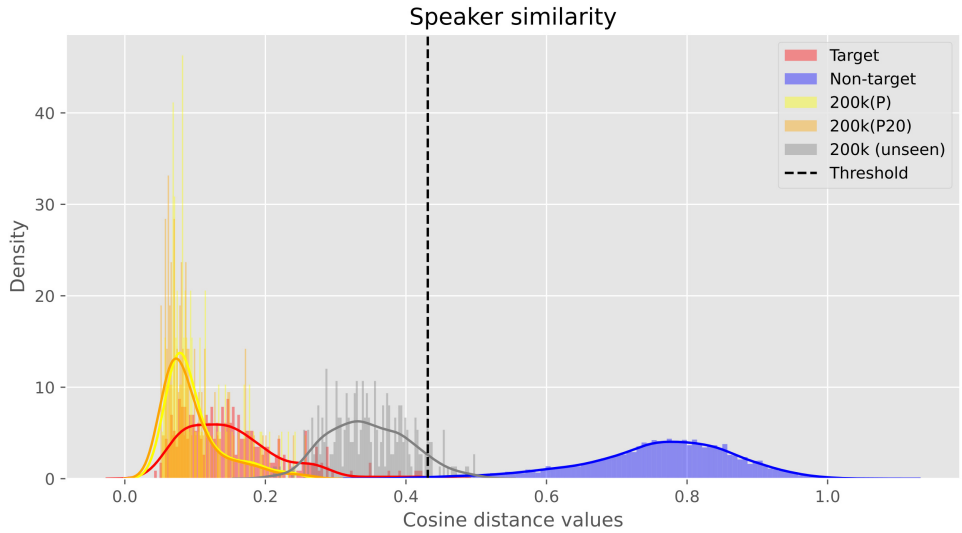

Figure 5.4: The fine-tuned models performed very similarly in terms of speaker similarity with a mean value of 0.1.

Based on the results in table 5.9, it is possible to fine-tune a personalized TTS model producing high-quality audio samples and get by with a limited amount of training data. It should be noted that the training data were manually picked to best reflect the speech properties of the target speaker and to preserve the average audio duration of the original dataset.

5.2.2 Female Speaker

This section deals with fine-tuning the 200k model using the personalized female dataset (3.5.2). In addition to the evaluation steps taken in section 5.2.1, the base model was fine-tuned on female samples prior to fine-tuning it to the target speaker.

Prior to fine-tuning

The pipeline for generating samples and the issues, which were encountered, were the same as described in section 3.5.1. Both the 1M and 200k models underperformed in terms of verification. This may be caused by the gender imbalance of the training data, despite the fact that this hypothesis was partly disproved in section 5.1.3.

Fine-tuning the base model using female data from ParCzech could not only improve the base model but also help with the subsequent fine-tuning for the target speaker. To prove this hypothesis, the 200k model was further trained for 150 000 steps on female speakers samples. Table 5.10 provides the statistics of the 1M model, 200k model, and fine-tuned 200k(F)[4].

Table 5.10: The 200k(F) model outperformed other models in all metrics. While in the case of the 1M model and 200k model, the segmentation method was deployed, the 200k(F) model was capable of generating correct durations for all samples. It should be noted that male voices were almost completely discarded. A comparison of the speaker similarity results can be found in the appendix (C.4).

	Original	**1M**	**200k**	**200k(F)**
WER	11.26 %	19.84 %	34.09 %	9.76 %
CER	6.25 %	8.10 %	22.18 %	3.81 %
MCD		15.97 dB	18.36 dB	15.63 dB
AR		21.67 %	10 %	28.33 %

Fine-tuned model

Since both 1M and 200k models in section 3.5.1 performed similarly once fine-tuned, only the 200k model for the female speaker was fine-tuned. Table 5.11 describes two fine-tuned models.

Table 5.11: Metrics of the fine-tuned female models. The speaker similarity results of both fine-tuned models can be found in the appendix (C.5).

	200k(N)	**200k(N+F)**
WER	13.93 %	11.61 %
CER	5.93 %	4.6 %
MCD	12.3 dB	12.67 dB
AR	100 %	100 %

Even though fine-tuning helped with performance on unseen female speakers, the results show that additional fine-tuning had little to almost no influence on the resulting model once fine-tuned on the same personalized dataset.

[4]Figure C.1 compares the enhanced performance of the fine-tuned model on unseen female speakers.

5.2.3 MOS Evaluation

As described in section 4.5, 15 samples were randomly selected for each male and female speaker. According to [59], samples used in the evaluation should not be too long[5] or too short. Therefore, the average audio duration for both speakers was 10 seconds, and all samples had different content.

The MOS evaluation was conducted using an online form using 26 respondents. The results are presented in table 5.12. To ensure the validity of the results, only respondents who were native Czech speakers were selected. Only one respondent stated their nationality as Slovak; however, given the similarity between Czech and Slovak languages and the fact that this respondent has lived in the Czech Republic for almost 10 years, their response was also considered valid.

Table 5.12: Table describing the MOS results for individual models. The samples were generated using 1M and 1M(P) models for the male speaker and 200k(F) and 200k(N+F) models for the female speaker.

	MOS		
Dataset	**Original**	**Fine-tuned**	**Non-fine-tuned**
Male	4.59	4.12	1.68
Female	4.55	3.02	3.42

The male fine-tuned model performed very well. However, the non-fine-tuned model (1M) was unable to produce intelligible samples with appropriate durations, and the segmentation method that was deployed may have had a negative impact on the results, as the samples did not sound natural.

The performance of the female fine-tuned model was worse than that of the male model. In the case of the fine-tuned model (200k(N+F)), the samples produced had clearer artifacts, which could have led to a less favorable evaluation by the listeners. On the other hand, the fine-tuned model (200k(F)) generated relatively good samples with natural-sounding durations.

The reason behind using the 1M model for the male speaker and the 200k model for the female speaker is as follows: the 1M model performed on the unseen male speaker better than the 200k model. The 200k(F) was fine-tuned on the female samples, thus, it was only fair to compare it to a better-performing model for the male speaker. Additionally, as can be seen in table 5.8, using either the 1M(P) or the 200k(P) does not make any difference in terms of the quality of the generated audio samples.

Since the research was conducted online, it is always challenging to validate the results. Thus, two of the original samples in the male subset were intentionally identical. The final mean MOS values of both samples were equal, which in some sense helped with validating the results.

It should be reminded that only naturalness and intelligibility were to be evaluated. Interesting results could also be obtained using a questionnaire extended with a question about the similarity to the target speaker, where the female personalized model could get better results.

Trying to compare these results to other TTS models trained in the Czech language presents quite a difficult task. Other Czech models are usually trained using proprietary

[5] Listeners may evaluate long samples based on the final few seconds, having forgotten the earlier parts.

datasets and deploy different subjective evaluation methods, such as [28]. The VITS model was also utilized and trained for up to 1.3 million steps, with the use of 14 hours of high-quality training data obtained from a single male speaker [28]. However, the results cannot be compared, as a different subjective metric was used.

5.3 User Application

The user application, called `Voiceify`, was developed beyond the scope of the assignment. It provides audio generation with a specified speaker, along with other features that are described further. The front-end was built using React, and the back-end was developed using Python with a Flask server, which communicates asynchronously using the POST method. Firebase[6] Authentication was utilized for authentication and the Firebase Firestore database was used to store data.

The application allows users to generate audio with either a male or female voice by utilizing the two best-performing personalized models for each gender. The target speaker is selected using a switch button next to the text input. Once an audio sample is generated, it is automatically played, and it is also added to the collection of user audio files, which can be viewed under the My Audio tab.

Due to the potential for misuse, user authentication was implemented using Firebase Authentication library functions to maintain information on the currently logged-in user. If the user is not logged in, the user is not permitted to access any other pages.

To generate audio, the client sends a POST request to the server. The server then runs a Python script that utilizes the appropriate model to generate the audio data. Once the data is generated, it is saved locally and a link to the audio is sent back to the client.

Unlike a mobile application that is specific to an operating system, this web application provides a responsive user interface that can be easily turned into a PWA (**P**rogressive **W**eb **A**pplication). This enables the application to behave like a regular mobile application. That implies downloading it to the target device, a full-screen mode, push notifications, and offline mode. Offline mode, for example, would allow for local storing of generated recordings.

The current limitations are higher HW and space requirements. The hosting server must offer enough performance and space for the models, which are around 2 GB in size. Another performance bottleneck is that the models are re-initialized when models are switched, which slows down the entire generation process. These problems could be solved with higher-quality domain and server parallelization to allow both models to be initialized.

[6] https://console.firebase.google.com

Chapter 6

Conclusion

The goal of this thesis was training a TTS (**T**ext-**T**o-**S**peech) model that generates the speech of a target speaker from input text. Specifically, the work focused on training a model using data in the Czech language, where high-quality training data options are limited. The VITS model was deployed for speech synthesis. The trained models were evaluated using objective metrics such as Word Error Rate, Character Error Rate, Mel-Cepstral Distortion, and acceptance rate for speaker verification. The generation of samples was conditioned on d-vectors, which uniquely characterize a given speaker. One problem that was frequently encountered was the inaccurate estimation of speech speed for longer input texts. As the cause could not be clearly identified, the input text was segmented, and audio parts were then joined to improve speech intelligibility. This problem was later resolved by fine-tuning on data of the target speaker.

As the aim of the work was a personalization for a specific speaker, datasets were created for a male and a female speaker, on which the base models were fine-tuned. The results of the work show that by using a large sample of data to train a base model and fine-tuning this model on a smaller personalized dataset, it is possible to achieve results of quality similar to the original recordings. In addition, the results of training using a limited amount of data demonstrated that it is indeed possible to train a high-quality TTS model with limited resources. The gender imbalance of the dataset used in the base model was attempted to be resolved by fine-tuning only on female data. Although this improved the ability to generate speech for female unseen speakers, it had no effect on the final personalized model. In addition to the objective metrics, audio samples generated by the personalized models were evaluated using the listening tests.

Moreover, the work's contribution includes a pipeline for evaluating recordings based on a set of objective metrics, which can be used for any other generated data after adjusting the paths. Furthermore, a pipeline was implemented for dataset creation that can be used to develop any other dataset, regardless of the language used.

Beyond the scope of this work, a responsive web application was created, which wraps the created personalized models into a user interface.

Examples of the generated samples can be found at the repository website[1].

[1] https://mike327327.github.io/TTS-0.8.0/

6.1 Future Works

One possible extension of the work could be fine-tuning the base model on data from a different speaker. Moreover, the datasets could be used for training newer models, such as Naturalspeech [53], outperforming the VITS model in MOS evaluation (4.56). Lastly, it would be worth exploring other female speakers and analyzing whether poor results would still be present compared to male speakers.

Bibliography

[1] Adaloglou, N. *Intuitive Explanation of Skip Connections in Deep Learning* [online]. The AI summer, 2020 [cit. 2023-02-02]. Available at: https://theaisummer.com/skip-connections/.

[2] Alammar, J. *The Illustrated Transformer* [online]. 2018 [cit. 2023-03-29]. Available at: http://jalammar.github.io/illustrated-transformer/.

[3] Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M. et al. Common voice: A massively-multilingual speech corpus. *ArXiv preprint arXiv:1912.06670*. 2019.

[4] Bredin, H. and Laurent, A. End-to-end speaker segmentation for overlap-aware resegmentation. In: *Proc. Interspeech 2021*. Brno, Czech Republic: [b.n.], August 2021.

[5] Bredin, H., Yin, R., Coria, J. M., Gelly, G., Korshunov, P. et al. pyannote.audio: neural building blocks for speaker diarization. In: *ICASSP 2020, IEEE International Conference on Acoustics, Speech, and Signal Processing*. Barcelona, Spain: [b.n.], 2020.

[6] Bäckström, T., Räsänen, O., Zewoudie, A., Zarazaga, P. P., Koivusalo, L. et al. *Introduction to Speech Processing*. 2nd ed. 2022. Available at: https://speechprocessingbook.aalto.fi.

[7] Casanova, E., Weber, J., Shulby, C. D., Junior, A. C., Gölge, E. et al. Yourtts: Towards zero-shot multi-speaker TTS and zero-shot voice conversion for everyone. In: PMLR. *International Conference on Machine Learning*. 2022, p. 2709–2720.

[8] Chen, Q., Tan, M., Qi, Y., Zhou, J., Li, Y. et al. V2C: Visual Voice Cloning. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, p. 21242–21251.

[9] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *ArXiv preprint arXiv:1406.1078*. 2014.

[10] Choubey, V. *Activation Functions in Neural Network: Steps and Implementation* [online]. 2023 [cit. 2023-03-27]. Available at: https://medium.com/codex/activation-functions-in-neural-network-steps-and-implementation-df2e4c858c21.

[11] Desplanques, B., Thienpondt, J. and Demuynck, K. ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based

Speaker Verification. In: Meng, H., Xu, B. and Zheng, T. F., ed. *Interspeech 2020*. ISCA, 2020, p. 3830–3834.

[12] Errattahi, R., El Hannani, A. and Ouahmane, H. Automatic Speech Recognition Errors Detection and Correction: A Review. *Procedia Computer Science*. 2018, vol. 128, p. 32–37. DOI: https://doi.org/10.1016/j.procs.2018.03.005. ISSN 1877-0509. 1st International Conference on Natural Language and Speech Processing. Available at: https://www.sciencedirect.com/science/article/pii/S1877050918302187.

[13] Giorgino, T. Computing and Visualizing Dynamic Time Warping Alignments in R: The DTW Package. *Journal of Statistical Software*. 2009, vol. 31, no. 7. DOI: 10.18637/jss.v031.i07.

[14] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016. 180-183 p. ISBN 978-0-26203561-3. www.deeplearningbook.org.

[15] Griffin, D. and Lim, J. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on acoustics, speech, and signal processing*. IEEE. 1984, vol. 32, no. 2, p. 236–243.

[16] Han, J., Kamber, M. and Pei, J. Getting to Know Your Data. In: Han, J., Kamber, M. and Pei, J., ed. *Data Mining*. 3rd ed. Boston: Morgan Kaufmann, 2012, p. 39–82. The Morgan Kaufmann Series in Data Management Systems. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00002-2. ISBN 978-0-12-381479-1. Available at: https://www.sciencedirect.com/science/article/pii/B9780123814791000022.

[17] Insider. *Insider 129 – Petr Pavel* [online]. 2023 [cit. 2023-02-25]. Available at: https://youtu.be/XSf_YkxzCBE.

[18] Insider. *Insider 130 – Danuše Nerudová* [online]. 2023 [cit. 2023-02-25]. Available at: https://youtu.be/8BWlSpp8Dfk.

[19] Iranzo Sánchez, J., Silvestre Cerda, J. A., Jorge, J., Roselló, N., Giménez, A. et al. Europarl-st: A multilingual corpus for speech translation of parliamentary debates. In: IEEE. *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, p. 8229–8233.

[20] Kim, J., Kim, S., Kong, J. and Yoon, S. Glow-TTS: A generative flow for text-to-speech via monotonic alignment search. *Advances in Neural Information Processing Systems*. 2020, vol. 33, p. 8067–8077.

[21] Kim, J., Kong, J. and Son, J. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech. In: PMLR. *International Conference on Machine Learning*. 2021, p. 5530–5540.

[22] Kong, J., Kim, J. and Bae, J. HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems*. 2020, vol. 33, p. 17022–17033.

[23] Kopp, M., Stankov, V., Kruza, J. O., Straňák, P. and Bojar, O. ParCzech 3.0: A large Czech speech corpus with rich metadata. In: Springer. *International Conference on Text, Speech, and Dialogue*. 2021, p. 293–304.

[24] Korvas, M., Plátek, O., Dušek, O., Žilka, L. and Jurčíček, F. *Vystadial 2013 – Czech data*. 2014. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Available at: http://hdl.handle.net/11858/00-097C-0000-0023-4670-6.

[25] Kratochvíl, J., Polák, P. and Bojar, O. Large corpus of Czech parliament plenary hearings. In: *Proceedings of The 12th Language Resources and Evaluation Conference*. 2020, p. 6363–6367.

[26] Louradour, J. *Whisper-timestamped* [https://github.com/linto-ai/whisper-timestamped]. GitHub, 2023.

[27] Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z. et al. Least squares generative adversarial networks. In: *Proceedings of the IEEE international conference on computer vision*. 2017, p. 2794–2802.

[28] Matoušek, J. and Tihelka, D. On Comparison of Phonetic Representations for Czech Neural Speech Synthesis. In: Sojka, P., Horák, A., Kopeček, I. and Pala, K., ed. *Text, Speech, and Dialogue*. Cham: Springer International Publishing, 2022, p. 410–422. ISBN 978-3-031-16270-1.

[29] Paine, T. L., Khorrami, P., Chang, S., Zhang, Y., Ramachandran, P. et al. Fast WaveNet generation algorithm. *ArXiv preprint arXiv:1611.09482*. 2016.

[30] Panayotov, V., Chen, G., Povey, D. and Khudanpur, S. Librispeech: an ASR corpus based on public domain audio books. In: IEEE. *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. 2015, p. 5206–5210.

[31] Pishro Nik, H. *Introduction to probability, statistics, and random processes* [online]. Kappa Research LLC, 2014 [cit. 2022-03-23]. Available at: www.probabilitycourse.com.

[32] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O. et al. The Kaldi speech recognition toolkit. *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. 2011.

[33] Prenger, R., Valle, R. and Catanzaro, B. Waveglow: A flow-based generative network for speech synthesis. In: IEEE. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, p. 3617–3621.

[34] Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C. et al. Robust speech recognition via large-scale weak supervision. *ArXiv preprint arXiv:2212.04356*. 2022.

[35] Rakhecha, A. *Importance of loss function in machine learning*. Towards Data Science, 2019 [cit. 2022-12-30]. Available at: https://towardsdatascience.com/importance-of-loss-function-in-machine-learning-eddaaec69519.

[36] Ravanelli, M., Parcollet, T., Plantinga, P., Rouhe, A., Cornell, S. et al. SpeechBrain: A General-Purpose Speech Toolkit. *ArXiv preprint arXiv:2106.04624*. 2021.

[37] reflexcz. *Pavel: Situace na Hradě je horší, než jsem čekal, možná padne trestní oznámení. Babiš mi napsal SMS* [online]. 2023 [cit. 2023-02-25]. Available at: https://youtu.be/EvehtO5neBk.

[38] Ren, Y., Ruan, Y., Tan, X., Qin, T., Zhao, S. et al. Fastspeech: Fast, robust and controllable text to speech. *Advances in Neural Information Processing Systems*. 2019, vol. 32.

[39] Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In: PMLR. *International conference on machine learning*. 2015, p. 1530–1538.

[40] Rocca, J. *Bayesian inference problem, MCMC and variational inference* [online]. Towards Data Science, 2019 [cit. 2023-02-25]. Available at: https://towardsdatascience.com/bayesian-inference-problem-mcmc-and-variational-inference-25a8aa9bce29.

[41] Rocca, J. *Understanding Generative Adversarial Networks (GANs)* [online]. Towards Data Science, 2019 [cit. 2023-03-28]. Available at: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29.

[42] Rocca, J. *Understanding variational autoencoders (VAES)* [online]. Towards Data Science, 2021 [cit. 2023-02-25]. Available at: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

[43] Saeed, M. *A Gentle Introduction to Positional Encoding in Transformer Models, Part 1* [online]. 2023 [cit. 2023-02-27]. Available at: https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/.

[44] Salvador, S. and Chan, P. FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. In:. January 2004, vol. 11, p. 70–80.

[45] Sazli, M. A brief review of feed-forward neural networks. *Communications Faculty Of Science University of Ankara*. 2006, vol. 50, p. 11–17. DOI: 10.1501/commua1-2_0000000026.

[46] Schwarz, D. and Rodet, X. Spectral envelope estimation, representation, and morphing for sound analysis, transformation, and synthesis. In: *ICMC: International Computer Music Conference*. 1999, p. 1–1.

[47] Senin, P. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*. 2008, vol. 855, 1-23, p. 40.

[48] Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N. et al. Natural TTS synthesis by conditioning Wavenet on Mel-spectrogram predictions. In: IEEE. *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. 2018, p. 4779–4783.

[49] Srivastava, R. K., Greff, K. and Schmidhuber, J. Highway networks. *ArXiv preprint arXiv:1505.00387*. 2015.

[50] STANDASHOW. *Generál PETR PAVEL: Občas je třeba ukázat zuby a bránit demokracii...* [online]. 2023 [cit. 2023-02-25]. Available at: https://youtu.be/SZX_HZDzLVM.

[51] Staudemeyer, R. C. and Morris, E. R. Understanding LSTM–a tutorial into long short-term memory recurrent neural networks. *ArXiv preprint arXiv:1909.09586*. 2019.

[52] Tae, J. *Glow-TTS* [online]. 2022 [cit. 2023-02-27]. Available at: https://jaketae.github.io/study/glowtts/.

[53] Tan, X., Chen, J., Liu, H., Cong, J., Zhang, C. et al. Naturalspeech: End-to-end text to speech synthesis with human-level quality. *ArXiv preprint arXiv:2205.04421*. 2022.

[54] Toda, T., Black, A. W. and Tokuda, K. Voice Conversion Based on Maximum-Likelihood Estimation of Spectral Parameter Trajectory. *IEEE Transactions on Audio, Speech, and Language Processing*. 2007, vol. 15, no. 8, p. 2222–2235. DOI: 10.1109/TASL.2007.907344.

[55] U Kulatého stolu. *Danuše Nerudová: Babiš prezidentem by byl signál jít do politiky. Další kandidaturu nevylučuji* [online]. 2023 [cit. 2023-02-25]. Available at: https://youtu.be/JRAZReZXO8o.

[56] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O. et al. Wavenet: A generative model for raw audio. *ArXiv preprint arXiv:1609.03499*. 2016.

[57] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention is all you need. *Advances in neural information processing systems*. 2017, vol. 30.

[58] Veaux, C., Yamagishi, J., MacDonald, K. et al. CSTR VCTK corpus: English multi-speaker corpus for CSTR voice cloning toolkit. *University of Edinburgh. The Centre for Speech Technology Research (CSTR)*. 2017.

[59] Viswanathan, M. and Viswanathan, M. Measuring speech quality for text-to-speech systems: development and assessment of a modified mean opinion score (MOS) scale. *Computer Speech & Language*. 2005, vol. 19, no. 1, p. 55–83. DOI: https://doi.org/10.1016/j.csl.2003.12.001. ISSN 0885-2308. Available at: https://www.sciencedirect.com/science/article/pii/S0885230803000676.

[60] Wang, Y., Skerry Ryan, R., Stanton, D., Wu, Y., Weiss, R. J. et al. Tacotron: Towards end-to-end speech synthesis. *ArXiv preprint arXiv:1703.10135*. 2017.

[61] Weitzman, T. *History of text-to-speech* [online]. 2022 [cit. 2022-12-29]. Available at: speechify.com/blog/history-of-text-to-speech/.

[62] Zen, H., Dang, V., Clark, R., Zhang, Y., Weiss, R. J. et al. LibriTTS: A corpus derived from LibriSpeech for text-to-speech. *ArXiv preprint arXiv:1904.02882*. 2019.

[63] Zhao, Y., Huang, W.-C., Tian, X., Yamagishi, J., Das, R. K. et al. Voice conversion challenge 2020: Intra-lingual semi-parallel and cross-lingual voice conversion. *ArXiv preprint arXiv:2008.12527*. 2020.

[64] Černocký, J. *Zpracování řečových signálů — studijní opora* [online]. Brno: VUT FIT, 2006 [cit. 2022-12-30]. Available at: https://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf.

Appendix A

Manual

This chapter describes the files provided on the attached SD card. More detailed steps on how to install and use required toolkits to run speech generation locally are provided in the `README.txt` file.

Scripts

Each script defines the required libraries and dependencies at the beginning of the file. Tested on Python 3.7. Files that can be found in the `scripts` folder are:

- `dataset.ipynb` – dataset creation pipeline, accepts a long .wav audio file and processes it as described in section 3.5,
- `metrics.ipynb` and `metrics_unseen.ipynb` – pipeline for generating samples with a given model on seen/unseen speakers and evaluating them using metrics, described in section 5,
- `mcd.ipynb` – calculating MCD of the generated samples,
- `mos_data_selection.ipynb` – used for random selection of the evaluation samples for MOS,
- `create_metadata_map_wavs.py` – generates mapping files of the training audio samples (`metadata.txt` and `spk_mapping.txt`).

Training

Version 0.8.0 of the Coqui-AI[1] API with modifications[2] is provided in folder `user_app/TTS-0.8.0`.

To reproduce the training procedure, the API can be installed using `pip install -e TTS-0.8.0/`. The training samples must follow[3] the same data structure as the ParCzech dataset (see 3.4). To train using d-vectors, `compute_embeddings.py` script must be run with the pretrained speaker encoder. Next, two additional files are required: `metadata.txt` and `spk_mapping.txt`. File `metadata.txt` pairs the audio path with its transcription, `spk_mapping.txt` maps the audio path with the speaker identifier. Once all three files and the training samples are obtained, `train_tts.py` is run with a configuration file[4] passed to it.

Synthesis

It is possible to generate speech without the user application. The synthesis section is provided in `YourTTS_metrics.ipynb`. The model path and configuration file must be selected at the beginning of the file and function `inference_d_vecs(embedding, txt, wav_name, out_path, scaler)` can be used to synthesize speech.

User Application

The user application files are stored in the `user_app` folder. It consists of two trained models[5], a pretrained speaker encoder, React project source files (`public` and `src` folders), and Flask server `serverV2.py`[6]

Detailed instructions for installing the user application to generate speech can be found in `README.txt`.

Others

Other models are stored in the `models` folder, and source files of this thesis are stored in the `bp` folder.

[1] https://github.com/coqui-ai/TTS

[2] Such as customized data formatters and code modification for sampling rate in the VITS model.

[3] Or a customized formatter must be defined.

[4] A sample configuration file is provided – relative paths to d-vector file, speaker encoder, mapping files, and other output directories must be redefined to match your environment.

[5] 200k(N+F) and 200k(P)

[6] Note that static paths in the script must be redefined to match your environment.

Appendix B

Additional Model Architectures

This additional section describes the Transformer model architecture (figure B.1), which was used in FastSpeech (2.3.3).

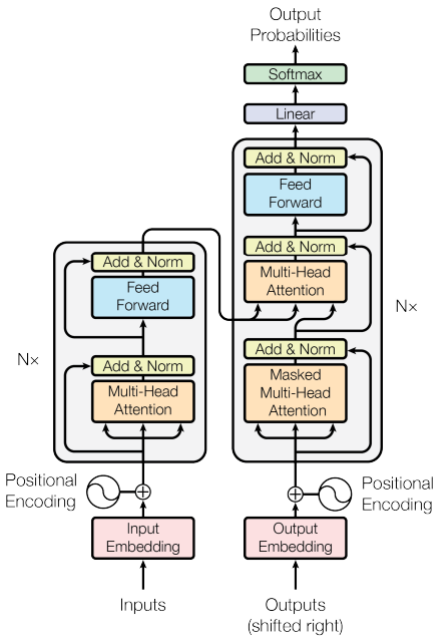

Figure B.1: Transformer model architecture [57]. First, input sentences are converted into word embeddings. Considering the language translation example, it is expected that a word could weigh a relation (self-attention) with itself higher. To overcome this issue, the attention vector for each word is computed multiple times and these vectors are then averaged and passed to the feed-forward network. Thus, the attention module is called MHA (**M**ulti-**H**ead-**A**ttention), as it consists of several attention units. During the training, the first MHA is masked to aid learning. If the whole input of the sentence in the target language could be seen, the network would not learn. The following MHA then determines how related each word vectors are to each other in both languages. After being trained, self-attention is utilized to identify which parts of the input and previously generated output are significant in creating a new word [2].

Appendix C

Plots

To ensure consistency of plot captions, each plot is defined (if possible) by

- `dataset` representing the name of a dataset used (ParCzech, Male, Female),
- `metric` describing the name of a metric shown in figure (WER&CER, verification),
- `model` denoting the model used (list of models can be found in 5.1),
- `seen/unseen/fine-tuned` defining if the results are based on a test subset of speakers seen/unseen during the training phase,
- `segmented/not segmented` providing information on whether the segmenting method was or was not used,
- `type of data` that can be either original or inferred,
- `additional comment` which further comments on the results if they are not self-explanatory.

C.1 Base Model

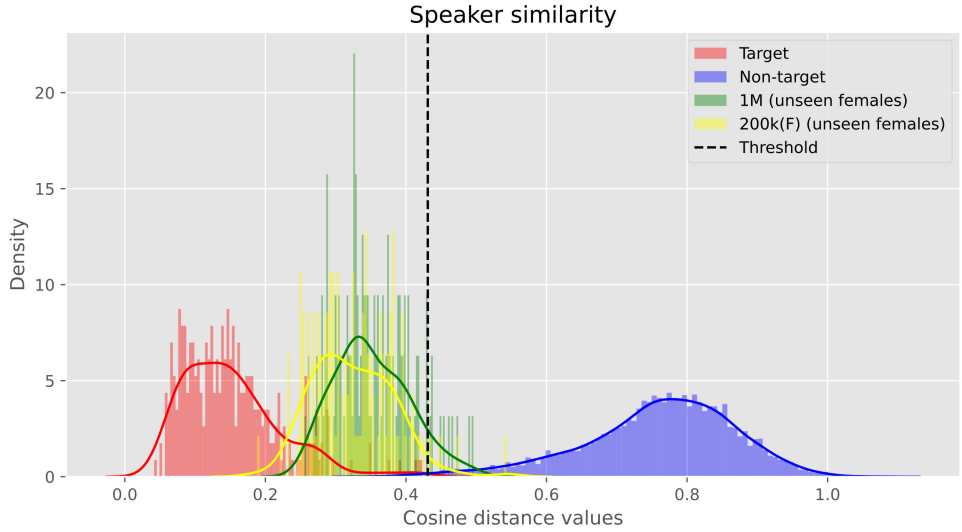

Figure C.1: ParCzech, verification, 1M and 200k(F) models, unseen, segmented and not segmented (respectively), inferred female data. After fine-tuning, the model adapts better to unseen speakers. The fine-tuned model is compared to the 1M model as it performed better than the original 200k model. The 1M model's mean value and acceptance rate were 0.35 and 92.42 %, respectively. For the 200k model, the results were 0.32 and 96.67 %.

C.2 Male Model

Unseen

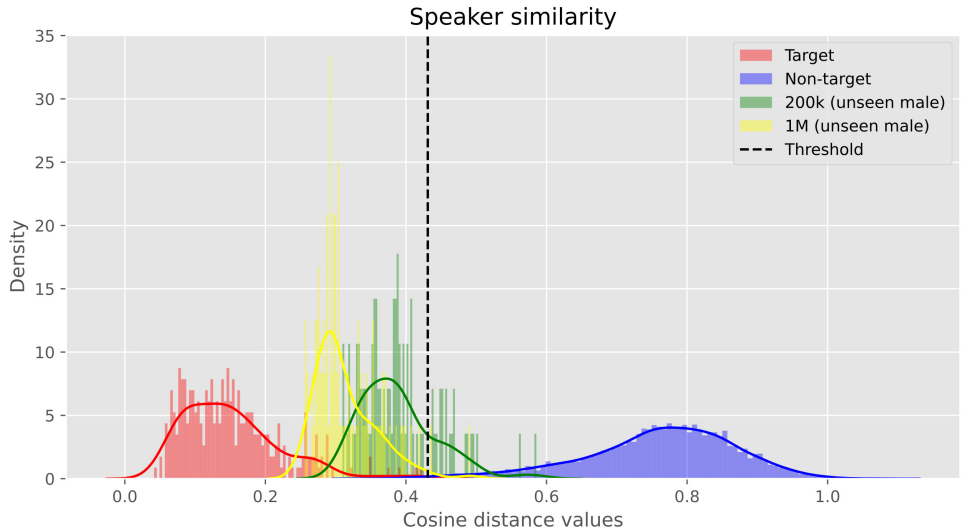

Figure C.2: Male, verification, 1M model and 200k model, unseen, segmented, inferred data. The 1M model adapts to the unseen speaker better. The 1M model's mean value and acceptance rate were 0.31 and 98.04 %, respectively. For the 200k model, the results were 0.38 and 79.41 %.

Fine-tuned

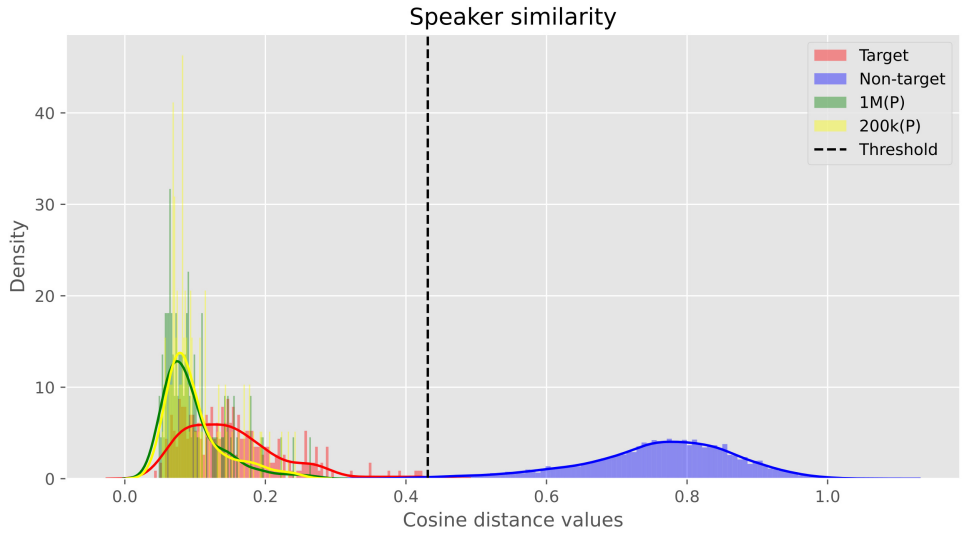

Figure C.3: Male, verification, 1M(P) and 200k(P) models, fine-tuned, not segmented, inferred data.

C.3 Female Model

Unseen

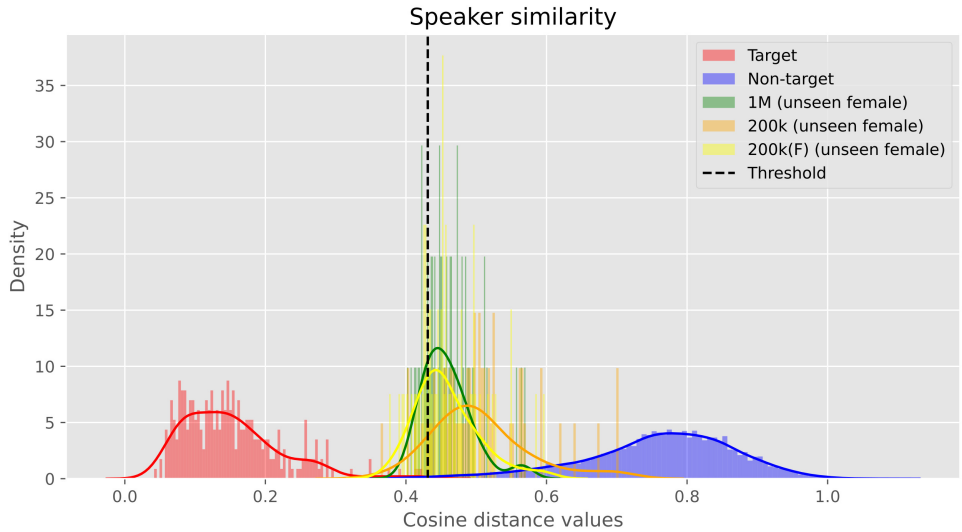

Figure C.4: Female, verification, 1M and 200k and 200k(F) models, unseen, segmented and segmented and not segmented, inferred data. The mean values of samples generated by the 1M, 200k, and 200k(F) were 0.46, 0.5, and 0.46, respectively.

Fine-tuned

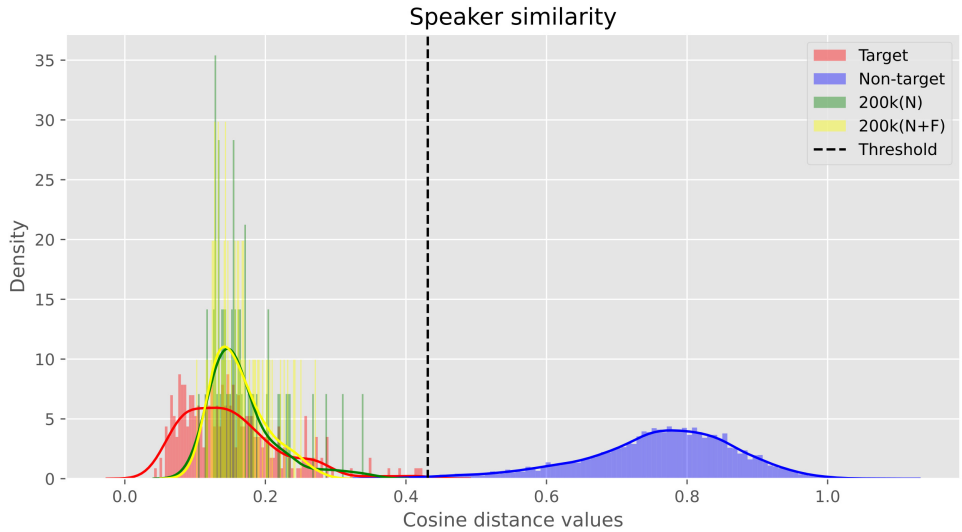

Figure C.5: Female, verification, 200k(N) and 200k(N+F) models, fine-tuned, both not segmented, inferred data.