Personalized Natural Language Interface

Valdemar Bång
Johannes Eriksson
David Grahn
Philip Gustafsson
Max Jonsson
Lukas Lundberg
Daniel Tufvesson

January 31, 2026

Contents

1 Introduction

We present a user interface application for finetuning open-source text-to-speech (TTS) and speech-to-text (STT) models to specific language domains. The application also provides easy methods for collecting speech data and evaluating the finetuned models.

1.1 Motivation

Recent advancements in natural language processing has given rise to highly sophisticated TTS [1], [2] and STT models [3]. Such models has the potential to be used in voice-driven user interfaces. However, their effectiveness often depend on how well they can adapt to an individual user. Existing models are trained on large generic dataset. Thus these models can struggle with characteristics specific to different users, such as speaking style, accent and preferred voice. Furthermore, some models, although trained in multi-lingual settings [2], perform poorly in many low resource languages. Additionally, many domains contain words that do not exist outside of that domain. Therefore, many of these domain specific words do not exist in the original training data, and hence the models fail to pronounce or recognize these. This lack of user personalization reduces accuracy and naturalness.

Many of these issues can be fixed via finetuning of these models. However, despite many of these models being open-source, their original vendors often only provide finetuning as a paid service. This means the user has to provide and trust the vendor with their data. This can be especially problematic when the user in question is a government agency (or comparable entity) where much of the data may be classified. In such cases, the user must implement their own finetuning solution. Although, possible, this requires extensive technical expertise. What is lacking is an easy-to-use tool that allows the user to finetune these models on a local machine without any extensive technical knowledge about the models.

1.2 Aim

The objective of this project is to evaluate the technical feasibility of such a tool. More specifically, to develop a user interface application consisting of a speech data collection pipeline, finetuning of TTS and STT models, and finally model evaluation (both subjective and automatic). This process is intended to be iterative: the user provides data by recording text prompts, the models are finetuned, the models are evaluated, and then the process may be repeated to increase accuracy. The aim is that each iteration should take approximately 30 minutes[1].

Given that data for TTS and STT finetuning requires transcribed audio, we also explore a method for quickly generating such data. Specifically, we explore how large language models (LLM) can be used for generating text prompts in a specified domain, and then let these be recorded by the user. Here, the ultimate aim is to reduce the amount of manual work required by the user.

We hypothesize that such an application is not only technically feasible, but will improve performance of the finetuned TTS and STT models. More specifically, when comparing to the base models, we expect to see higher performance on both automatic metrics (loss, word error rate, and character error rate) and subjective metrics (naturalness, intelligibility, and vocal artifacts). For the TTS, we expect the model to better pronounce specific Swedish vowel sounds, such as ”å”, ”ä”, and ”ö”, as well as to learn to pronounce domain specific words.

1.3 Approach

Our overall approach is as follows. For TTS, Chatterbox Multilingual is used. A custom finetuning script is implemented for finetuning the TTS model. This implementation is based on [4].

[1] This time target is fairly arbitrarily chosen.

For STT, KB-Whisper-Large is used. Optuna hyperparameter search is used for finding the optimal parameters for finetuning the STT model. Unsloth is used for faster and less memory intensive finetuning of the STT.

The application is divided into different Docker containers [5]: a web browser front-end running on Angular, a python Flask back-end application, a FastAPI application for TTS, and a FastAPI application for STT.

1.4 Delimitations

To limit the scope of the project a decision was made to focus on creating an application that works for Swedish, because we are native Swedish speakers in the group.

Furthermore, no user studies were done on the developed application.

2 Method and Implementation

This section covers the user flow and implementation of the application, the TTS and STT models, their finetuning and evaluation.

2.1 User Flow of the Personalization Pipeline

Data collection, finetuning, and evaluation is done as a single pipeline. We denote this the *Personalization Pipeline*. It consists of the following steps.

Create Profile: The first task presented to the user is to create a new personalization profile. The purpose of this is to ease the management of the data and the finetuned models associated with this personalization session.

Generate Prompts: The user is then able to automatically generate a sample of text prompts for a given topic. The user provides the topic, the number of sentence, and optionally some custom, domain specific words to be used in the prompts. The system then generates these prompts via an LLM (see subsection 2.3).

Validate Prompts: The generated text prompts are then displayed to user. The user can validate these by either removing them from the sample, or by editing their textual content.

Record Voice Samples: After validation, the user records voice samples for each of these text prompts. This is done prompt by prompt. After a prompt has been recorded, the user can listen to the recording and validate that its quality is sufficient, and re-record the prompt if it was not.

Finetune: The recorded samples are used for finetuning both the TTS and STT models. The finetuning process is displayed as two progress bars, one for each model.

Evaluate: Finally, the user can evaluate the models and compare them to the baseline. The process can then be repeated if results were not satisfactory.

2.2 System Architecture

The application consists of several Dockerized [5] environments. This was done primarily to encapsulate the business logic of each component. In particular, we treated each AI model (TTS, STT, LLM) as

an isolated microservice which is accessible via an API. The docker design is illustrated in 2.2, with endpoints connecting to the different dockers containing the STT and TTS functionality.

Frontend: The frontend was written in Angular and runs locally in the user's web browser. The frontend only calls the backend.

Backend: The backend was written in Python Flask. The backend calls the TTS, SST, and LLM microservices. A mounted volume is shared by the backend and the TTS and the STT microservices. The backend is responsible for preparing the data (for example for finetuning or evaluation) and models for the microservice, before calling their APIs.

TTS: This microservice encapsulates the TTS logic, that is, voice synthesis, finetuning, and evaluation. The Python FastAPI framework was used for defining the API calls. Data and cached models are stored on a shared mounted volume.

STT: This microservice encapsulates the STT logic, that is, voice transcription, finetuning, and evaluation. The Python FastAPI framework was used here as well for defining the API calls. Similarly, data and cached models are stored on a shared mounted volume.

LLM: This is a vLLM [6] microservice that runs qwen3-8b LLM [7]. This is only used for generating text prompts.

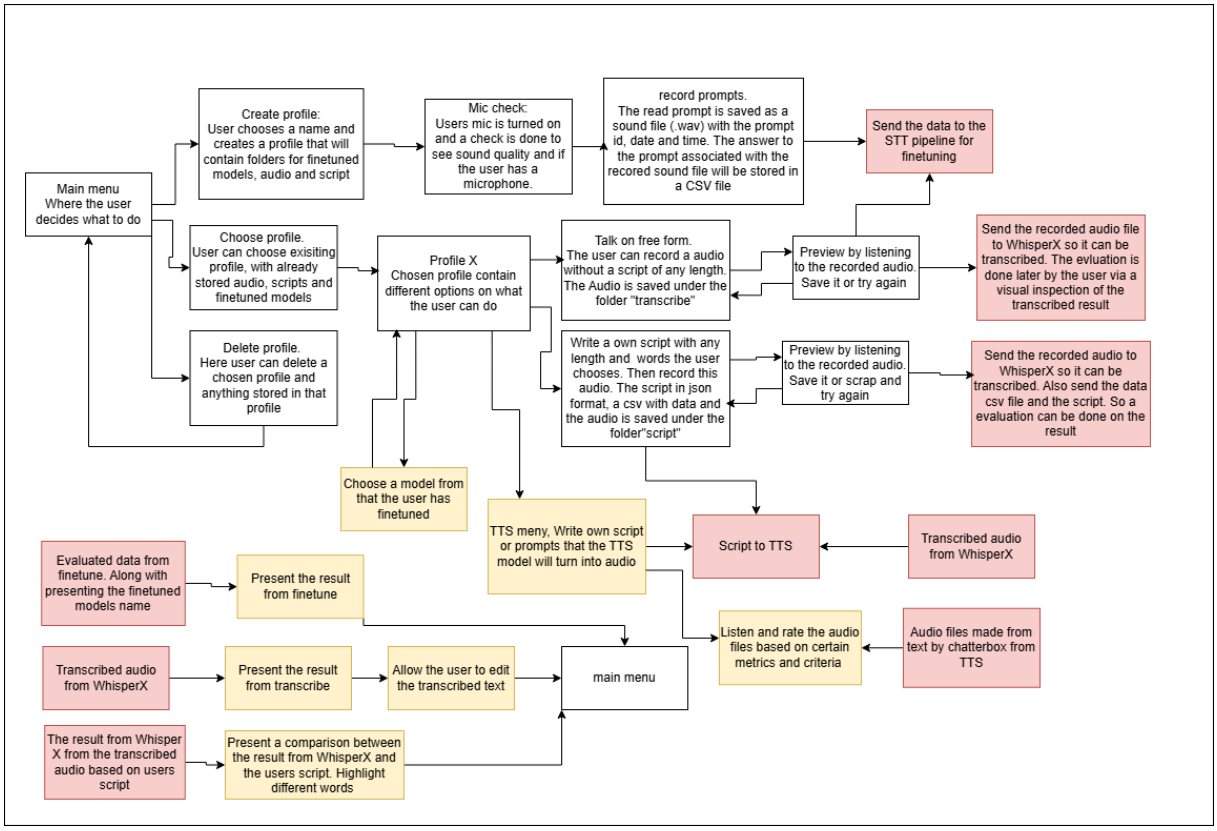

Figure 1: A diagram over the products front/backend docker. Endpoints in red and yellow not yet implemented

2.3 LLM for Prompt Generation

To automatically generate prompts, the qwen3-8b [7] LLM was used. It accepts a text query and provides a text response.

$$TextResponse \sim Qwen\,(TextQuery)$$

To generate a batch of text prompts for a given domain, we provided it with the system and user prompt in Table 1. The system and user prompts describe the task (generate domain specific prompts), the number of prompts, a list of custom words, and the domain. The LLM outputs the prompts as a JSON text string. Essentially, we get:

$$Prompts \sim GeneratePrompts\,(Topic, CustomWords, n)$$

where n is the number of prompts. Note, however, that the process is not guaranteed to generate exactly n prompts. This is because LLMs in general have problems with counting and enumeration [8].

Table 1: Examples of transcription errors by the finetuned KB-whisper model. <N> is replaced with the actual number of prompts to generate. Similarly, <DOMAIN> is replaced with a text string denoting the domain.

Prompt Type	Prompt
System Prompt	You are an expert in technical domains and the Swedish language. You generate high-quality, vocabulary-rich training data for speech-to-text models in Swedish.
User Prompt	Generate <N>short, distinct sentences in Swedish about <DOMAIN>. The sentences should be suitable for reading aloud. IMPORTANT: Use specific technical terminology, jargon, and concepts unique to domain. Avoid generic sentences. Do not just repeat the word <DOMAIN>. Return ONLY a JSON array of strings. Example: ["Det neurala nätverket konvergerar efter femtio epoker.", "Den aerodynamiska lyftkoefficienten är kritisk för flygstabilitet."]

2.4 STT

For STT, the project uses KB-Whisper, a Whisper model by KBLab [3]. Given an audio file, it transcribes it and outputs it as a text.

$$TranscribedText \sim KB\text{-}Whisper(Audio)$$

For finetuning and running the KB-Whisper model, the Unsloth framework was used. The main reason for this choice is that Unsloth speeds up the training process and reduces memory usage through optimization techniques such as Low-Rank Adaptation (LoRA) [9].

The dataset was created by recording your own voice, reading one sentence at a time from a predefined script. This approach was chosen because when finetuning with Unsloth it requires each audio clip to be no longer than 30 seconds.

For the finetuning process, the dataset was split into three subsets: 60% for training, 15% for validation, and 25% for testing. The HuggingFace (HF) Trainer class was then used to carry out the training, as it is optimized for transformer-based models. The loss function during training was chosen as the Word Error Rate (WER). To decide which training arguments to change when finetuning Whisper, the most common ones for machine learning tasks was chosen. For the Optuna parameter ranges, we ran some finetuning tests to find which one impacted the model the most when finetuning, see table 2.

To run the base or finetuned model, a framework called WhisperX is used. This framework adds word-level timestamps using wav2vec2 alignment and also adds batching when running the model inference, enabling real-time transcriptions [10]. The STT docker endpoints for Optuna and WhisperX can be seen in the figure 2 below.

Table 2: Standard training parameters and Optuna search ranges.

Parameter	**Standard Value**	**Optuna Range / Choices**
Warmup ratio	0.3	[0.05, 0.3]
Learning rate	1.65e-4	[1e-6, 5e-4]
Weight decay	0.1	[0.0, 0.15]
Training batch size	4	[2, 6]
Evaluation batch size	6	[4, 8]
Training epochs	5	[3, 8]
Max gradient norm	1.0	[0.5, 2.0]
Label smoothing factor	0.0	[0.0, 0.1]
Learning rate scheduler	cosine	{cosine, linear, constant}
Gradient accumulation steps	4	[2, 4]

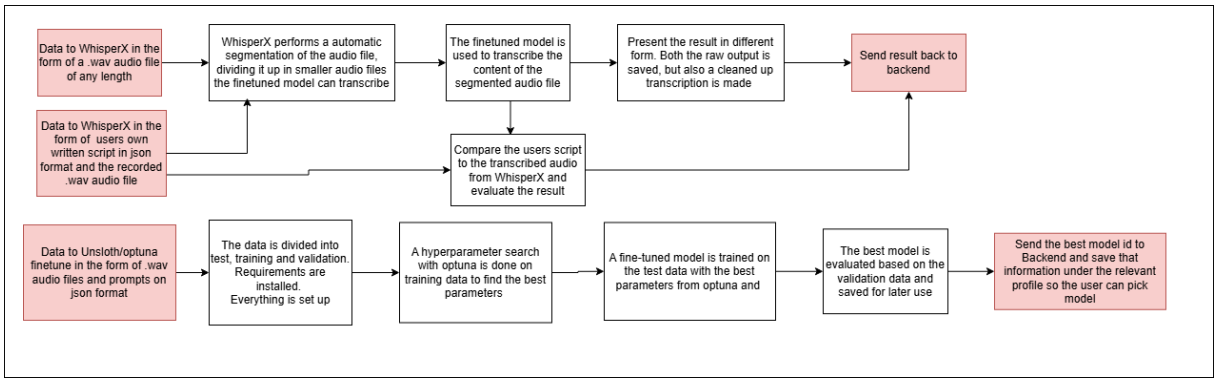

Figure 2: A diagram over the STT docker. Endpoints are red

2.5 TTS

This project relies on the Chatterbox TTS model by Resemble AI[2]. It is a zero-shot cloning model, meaning that, given a short audio voice sample and a text, it will produce a new synthesized audio of that text with a voice matching that of the voice sample:

$$SynthesziedVoice \sim Chatterbox(\text{Text}, \text{VoiceSample})$$

Voice cloning, however, is not enough, so we also finetuned the model to the target voice. At the time of the project, no official finetuning code was available. There was however a fork available for finetuning the English model[4]. Since this was only available for the English model, we adapted the code to finetune the multi-lingual model as well. This adaptation was mainly done with GitHub Copilot.

Finetuning was done with 89 Swedish voice samples of one of the project member's voice. The dataset was split into 10% evaluation and 90% training. Training was done over 10 epochs at a learning of 10^{-5}.

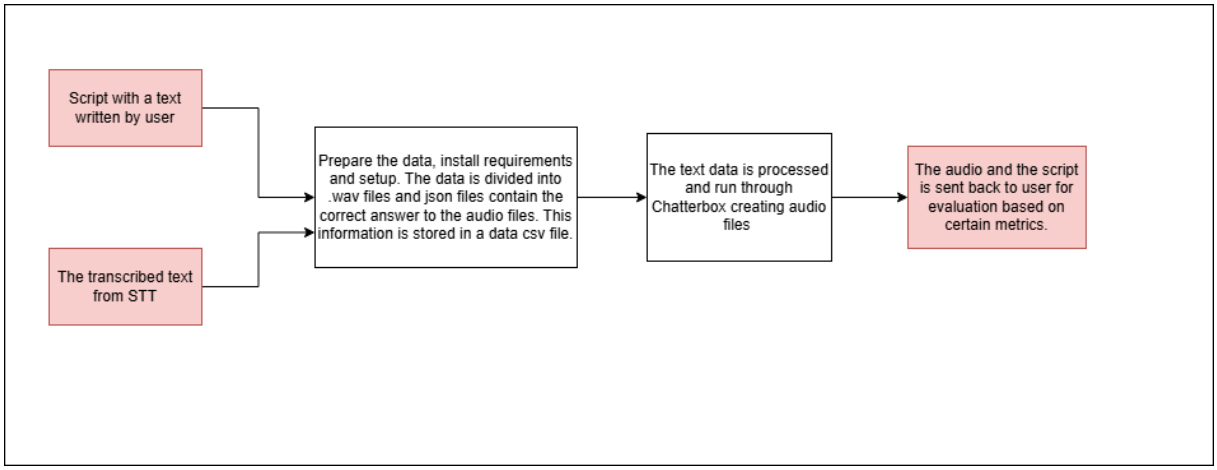

Figure 3: A diagram over the TTS docker. Endpoints are red

2.5.1 Automatic Evaluation

Automatic evaluation was done synthetic data to compare the base model and the finetuned model. First, 10 sentences were synthesized by the base model and the finetuned model. These were then fed to a whisper STT model, giving the transcribed sentences. The transcriptions were then compared with the original sentences.

These were then evaluated with *word error rate* (WER) and *character error rate* (CER). These are defined as:

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

where S is the number of substitutions (words/characters that the model altered), D is the number of deletions (words/characters that that the model missed), I is the number of insertions (words/characters that that the model added), and N is the total number of words (for WER) or characters (for CER).

2.5.2 Subjective Evaluation

Evaluating a TTS model purely based on objective metrics can be problematic. This is because the STT model is not perfect and might miss words it doesn't recognize. It may also fail to capture if the TTS model synthesizes words in a way that is appealing and sounds right to humans. It is therefore important to have a subjective metric is better at capturing how well the model sounds according to human opinion.

One such metric the is the *Mean Opinion Score* (MoS), in which humans rate sound files from 1 - 5, based on four factors: 1) voice naturalness (how human-like the voice sounds), 2) intelligibility (how easy the voice is to understand), 3) speaker similarity (how the voice resembles the target speaker), and 4) artifacts (random noises and glitches in the sound files). The MoS is then the mean of these four factors.

The subjective evaluation was done on the same synthetic data as in the automatic evaluation in subsubsection 2.5.1.

3 Results

This section presents the results from the evaluations on the STT and TTS models.

3.1 STT

The finetuned KB-whisper model achieved a WER of 17.11% on 23 held-out Swedish audio samples, successfully transcribing 14 samples (60.9%) failing on 9. This represents 2.89% point improvement over the validation set WER of 20.0% indicating good generalization from the 52-sample training set. Training converged in 7.6 epochs over 102 seconds.

A qualitative assessment of the 9 erroneous transcriptions revealed some patterns that are difficult for the model. These are presented in Table 3. Most of the errors came from wrong numeric format. Other errors were split words becoming joint compounded words. There were three errors that involved punctuation differences where trailing commas were omitted. Finally, phonetic substitutions accounted for the remaining errors.

Table 3: Examples of transcription errors by the finetuned KB-whisper model.

Actual	**Transcribed**	**Error type**
svisch	swish	Phonetic substitution
tretton trettio	13.30	Format error
tre komma sju kilometer	3.7km	Format error
noll komma sjuttiofem kilo	0.75kg	Format error
Tjällmo byn	Tjällmobyn	Compound error
i kväll	ikväll	Compound error
Bron	Hon	Phonetic substitution
Hon	de	Pronoun error
de	det	Pronoun error
tjljudet	stelljudet	Phonetic substitution
frisk,	frisk	Punctuation error

Training efficiency was notable, with rapid convergence, demonstrating the viability of parameter efficient fine-tuning for personalized speech recognition on consumer hardware. The improvement from validation to test performance confirms that the model learned generalizable features rather than memorizing the small training set.

3.2 TTS

3.2.1 Word- and character error rate

Table 4 presents the character error rate (CER) and word error rate (WER) for all ten evaluation sentences, comparing the base model with the finetuned version. The finetuned model achieves lower average error rates, with CER decreasing from 0.124 to 0.057 and WER decreasing from 0.237 to 0.116. For the two sentences 001, and 008, both models produce perfect scores. For a few sentences such as 002 and 003 the finetuned model produces CER/WER values that are similar to or higher than those of the base model.

ID	CER Base	CER FT	WER Base	WER FT
000	0.333	0.042	0.667	0.222
001	0.000	0.000	0.000	0.000
002	0.000	0.074	0.000	0.222
003	0.020	0.020	0.125	0.125
004	0.096	0.058	0.125	0.125
005	0.054	0.000	0.222	0.000
006	0.333	0.278	0.400	0.300
007	0.378	0.089	0.750	0.125
008	0.000	0.000	0.000	0.000
009	0.021	0.007	0.077	0.038
Avg	**0.124**	**0.057**	**0.237**	**0.116**

Table 4: CER and WER results for base vs finetuned Chatterbox TTS model.

3.2.2 Mean opinion score

For the MoS evaluation, the people were asked to evaluate 10 different sound files synthesized from the base and the fine tuned TTS-model. The models were then evaluated on 3 different metrics and the averages of these metrics were then calculated to obtain the final MoS. These metrics were naturalness, intelligibility, and artifacts. The results from the MoS evaluation can be seen in table 5. The results clearly show that the fine-tuned model performed better on all three metrics. The overall MoS difference between the two models were 1,31. The greatest improvement for the fine-tuned model compared to the base model is naturalness where the fine-tuned model scored 1,77 points higher than the base model.

Metrics	Base Model	Fine-tuned Model
Naturalness	2.33	4.10
Intelligibility	3.06	4.22
Artifacts	3.35	4.37
Overall MoS	**2.92**	**4.23**

Table 5: Mean Opinion Score (MoS) comparison between base and fine-tuned models across individual perceptual metrics and overall average.

4 Discussion

4.1 STT

The results from the WER from the fine-tuned KB-whisper model shows a WER of 17.11%. This is much higher than the WER from KBLab [3], where they get an WER of 4-5% on the datasets FLEURS [11], CommonVoice [12] and NST [13]. This difference was expected given the nature of parameter efficient fine-tuning on a small, single speaker dataset, but the magnitude of the difference warrants discussion.

KBlabs lower WER was achieved by training on vast diverse datasets, whereas our fine-tuning was based on a small single speaker 52-sample training set. The models difficulty in generalizing across different acoustic environments and linguistic nuances is amplified when training data is so limited. The model demonstrated a strong performance in learning speaker specific voice characteristics and Swedish phonetics but the narrow domain of the training prompts likely limited its ability to handle general Swedish conversation.

Despite high WER compared to the general model, the project was successful in achieving its primary goal, creating a personalized model. The fine-tuned model showed a 2.89 percentage point improvement in WER from the validation set to the test set, indicating good generalization and successful learning of user specific features rather than simple memorization.

As the qualitative assessment of erroneous transcription revealed, one third of the errors were numeric formatting errors and another third was punctuation errors. They are not caused by comprehension failure by the model. These are possible to address with post-processing step. Additionally, the compound word errors are more stylistic variation than errors per se. Only the phonetic errors represent genuine semantic confusion requiring model improvement. The high perfect transcription rate combined with systemic error patterns indicates successful learning learning of speaker-specific voice characteristics and Swedish phonetics.

4.2 TTS

4.3 Word- and character error rate

The results from the CER/WER evaluation (Table 4) indicate that finetuning clearly improves the base model's transcription accuracy, as seen in the overall reduction in both CER and WER. The largest improvements can be seen in sentences with more difficult pronunciations or longer word sequences, suggesting that the finetuned model handles difficult cases more consistently than the base model. However, the evaluation also shows that finetuning does not improve every sentence. For example, sentences 002 and 003 have similar or even slightly higher error rates after finetuning. This suggests that some types of errors are not fully corrected by the current finetuning setup. Additionally, the sentences where both models performed perfectly, 001 and 008 indicate that certain sentences are already easy for the base model, leaving little room for improvement.

In interpreting these results, it is also important to consider several factors that may have influenced the evaluation. The test set consists of only ten sentences, which means the results give an estimate of performance but do not fully represent all types of language use. The recordings were made using a standard microphone, which introduces background noise and audio artifacts that can affect both pronunciation clarity and the accuracy of the transcription. Since the evaluation relies on the STT model whisper to transcribe audio into text, some of the measured errors may come from the STT model rather than the TTS model itself.

4.4 Mean opinion score

The results from the MoS evaluation also indicate that the finetuning had a positive impact on adapting to the users voice and speaking style. The overall metric score increased, and all individual scores for the metrics that were evaluated also increased.

The metric that differed the most between the two models was naturalness. The base model achieved a very low score of 2.33. This is probably due to the fact that the base model is bad at correctly pronouncing words containing the letters "å", "ä", and "ö", which frequently appear in the Swedish language. The base model also speaks in a tempo significantly faster than the average Swedish person.

Both models speak Swedish in a manner that makes it easy for the average Swedish speaker to understand. However, as stated earlier, the base model has difficulty with certain letters and speaks in an unnatural tempo. This impacts intelligibility, but not with the same magnitude that it affects naturalness. Users stated that the fine-tuned model creates sound files with a natural speaking tempo and pronounces the letters in question more accurately. Loanwords and slang words proved difficult for both models. This is probably due to the fact that these and similar words are not present in the fine-tuning dataset.

For the final metric, artifacts, the increase in score is mainly due to that the base-model often produced sound files with bizarre and uncanny noises. These are not present in the sound files for the finetuned-model.

The training data for the fine-tuned model were recorded with a low quality microphone. This has led to the model producing sound files that sound like they were recorded with the same microphone. This may

have had an unfavorable effect on score for all metrics, especially artifacts. This may also have a negative impact for users training a model based on their own voice using their own microphone. As most users are not expected to own a high quality microphone.

5 Conclusion

The objective of this project was to develop a user interface application for finetuning and personalizing STT and TTS models. The results confirm that personalization is achievable for both TTS and STT. The fine-tuned STT model showed improved performance over the validation data, though its overall WER of 17.11% is relatively high. A lot of errors were format- or style related. This indicates that accuracy could easily be improved.

The TTS evaluation demonstrated improvements in both objective metrics (CER and WER) and perceived quality (MoS). Result may have been negatively affected by data set size and microphone quality.

Overall, the chosen approaches have been proven to be work. With personalization leading to noticeable performance gains. Future work should focus on improving and expanding training data, and to incorporate these models into the user interface, so that users can perform finetuning themselves directly within the application.

References

[1] Canopy AI (formerly Canopy Labs), "Orpheus–tts: Towards human-sounding speech," https://github.com/canopyai/Orpheus-TTS, 2025, accessed: 2025-11-14.

[2] Resemble AI, "Chatterbox-TTS," https://github.com/resemble-ai/chatterbox, 2025, gitHub repository.

[3] KBLab, "Hugging face repository," https://huggingface.co/KBLab, 2025, accessed: 2025-11-14; kb whisper large.

[4] S. Lohrey. chatterbox-finetuning. [Online]. Available: https://github.com/stlohrey/chatterbox-finetuning

[5] Docker: Accelerated container application development. [Online]. Available: https://www.docker.com/

[6] "vllm-project/vllm," original-date: 2023-02-09T11:23:20Z. [Online]. Available: https://github.com/vllm-project/vllm

[7] Qwen/qwen3-8b · hugging face. [Online]. Available: https://huggingface.co/Qwen/Qwen3-8B

[8] K. Hou, M. Zorzi, and A. Testolin, "Sequential enumeration in large language models." [Online]. Available: http://arxiv.org/abs/2512.04727

[9] "Text-to-speech (tts) fine-tuning — unsloth documentation," Unsloth.ai, 11 2025. [Online]. Available: https://docs.unsloth.ai/basics/text-to-speech-tts-fine-tuning

[10] M. Bain, J. Huh, T. Han, and A. Zisserman, "Whisperx: Time-accurate speech transcription of long-form audio," *INTERSPEECH 2023*, 2023.

[11] A. Conneau, M. Ma, S. Khanuja, Y. Zhang, V. Axelrod, S. Dalmia, J. Riesa, C. Rivera, and A. Bapna, "FLEURS: Few-shot learning evaluation of universal representations of speech." [Online]. Available: http://arxiv.org/abs/2205.12446

[12] Common voice. [Online]. Available: https://www.kaggle.com/datasets/mozillaorg/common-voice

[13] KTH/nst · datasets at hugging face. [Online]. Available: https://huggingface.co/datasets/KTH/nst