OST
Ostschweizer
Fachhochschule

19.12.2025 - Student Research Project Autumn Semester 2025/26

Real-Time Speech Translation with Voice Cloning

Students: **David Bürge** david.bürge@ost.ch
Tareq Kattit tareq.kattit@ost.ch

Supervisors: **Prof. Dr. Mitra Purandare** mitra.purandare@ost.ch
Prof. Dr. Markus Stolze markus.stolze@ost.ch

Abstract

Advances in neural speech synthesis have made it increasingly easier to clone a person's voice using only a brief audio sample. While this opens new possibilities, it also introduces risks such as impersonation, fraud, misinformation, and privacy violations. At the same time, it remains unclear whether open-source models can produce results that are truly convincing. This work aims to develop and evaluate a real-time system for voice conversion and speech translation based on open-source models running on consumer hardware, with a focus on generating speech that is natural, intelligible, and perceptually convincing.

To achieve this goal, a system is developed integrating and orchestrating open-source models for automatic speech recognition (ASR), machine translation, and text-to-speech synthesis, including Whisper, MarianMT, and text-to-speech models supporting zero-shot voice cloning such as F5-TTS and VoxCPM. The models are selected based on their tradeoff between transcription or synthesis quality and low latency for real-time use. Because the selected models do not support fully streaming inference, the continuous audio input must be segmented before processing. Voice activity detection (VAD) is therefore used to detect short pauses in the incoming speech and split the audio stream into manageable segments. These segments are then processed sequentially by the ASR, machine translation, and speech synthesis components. To reduce transcription errors caused by short noise bursts, the VAD was configured to emit segments with a minimum duration of 300 milliseconds and with reduced sensitivity. The quality and credibility of the generated speech are evaluated through a combination of automatic metrics and human assessments, focusing on naturalness, intelligibility, and voice similarity, while also accounting for end-to-end latency.

The system shows that real-time voice conversion can be achieved on consumer hardware using open-source models. For short German audio samples, the generated speech is perceptually convincing, achieving an average naturalness score of 4.28/5.0 and a voice similarity score of 0.8/1.0, with comparable results for English synthesis. When speech translation is enabled, the end-to-end system achieves an average latency of approximately 1.7 seconds, rising to about 3.9 seconds when speech duration is taken into account, which remains suitable for real-time use. While the overall translation quality is acceptable for short, isolated segments, limitations emerge for longer speech. The machine translation models used do not support providing preceding segments as context, resulting in more literal, sentence-by-sentence translations, while nuances spanning multiple segments are lost. In addition, the modular design allows individual models to be swapped, enabling experimentation with alternative configurations and supporting future extensions. Overall, these results confirm the feasibility of building real-time voice cloning systems with current open-source models and highlight both their accessibility and potential risks they entail.

Contents

Part I: Introduction

1 Management Summary

1.1 Problem Statement

Recent advances in speech synthesis and speech translation have enabled the recreation of a person's voice from a short audio sample, allowing for its use in applications such as real-time speech translation with voice cloning. These capabilities become particularly sensitive when they are easily accessible, as widespread availability increases the potential for misuse, including impersonation, fraud, misinformation, and violations of personal privacy.

While commercial solutions for real-time speech translation already exist, they are predominantly cloud-based and are therefore less accessible. In parallel, open-source technologies for speech recognition, translation, and speech synthesis have matured considerably. However, it remains unclear whether these openly available technologies can provide real-time, low-latency speech translation with convincing voice cloning on consumer-grade hardware.

The objective of this thesis is to implement a prototype of such a system. The goal is to enable real-time translation of spoken language between German, English, French, and Italian while preserving the speaker's voice, and to evaluate whether the resulting speech is sufficiently natural, intelligible, and responsive for practical use.

1.2 Approach

This thesis explored the development of a web-based system that translates spoken language in real-time while maintaining the speaker's voice recognizability. The project evaluated existing open-source machine learning models to select the most suitable ones. It integrated them into a single system that processes spoken input continuously, translates it, and plays back the translated speech in real time. The development followed an iterative process with regular planning and review to ensure steady progress and early identification of issues. After implementation, the system was tested to see how well it met the goal of providing fast, understandable, and natural-sounding translations while preserving the speaker's voice.

1.3 Technologies

The system runs entirely on consumer-grade hardware and was developed using widely used programming tools and frameworks. Key components include a web-based interface for users, machine learning models for understanding spoken input, translating it, and generating output in the speaker's voice.

Backend	Python with FastAPI
Frontend	React Typescript
Automatic Speech Recognition	Faster-Whisper
Text to Speech	F5-TTS, VoxCPM, XTTSv2
Machine Translation	MarianMT, M2M100
Environment	Docker
Documentation	Typst

Table 1: Technologies used

1.4 Results

The system successfully achieves the intended goals. Translations are delivered quickly, typically within a few seconds, and maintain a high level of clarity and naturalness. The speaker's voice is preserved reliably. The system's modular design allows future extension, such as testing different machine learning models.

This work demonstrates that a fully open-source, real-time voice translation system is feasible on consumer hardware. It also highlights important societal and ethical considerations, such as the potential misuse of voice cloning. Future development could expand the system to support additional languages, improve speed, or include safeguards to prevent misuse.

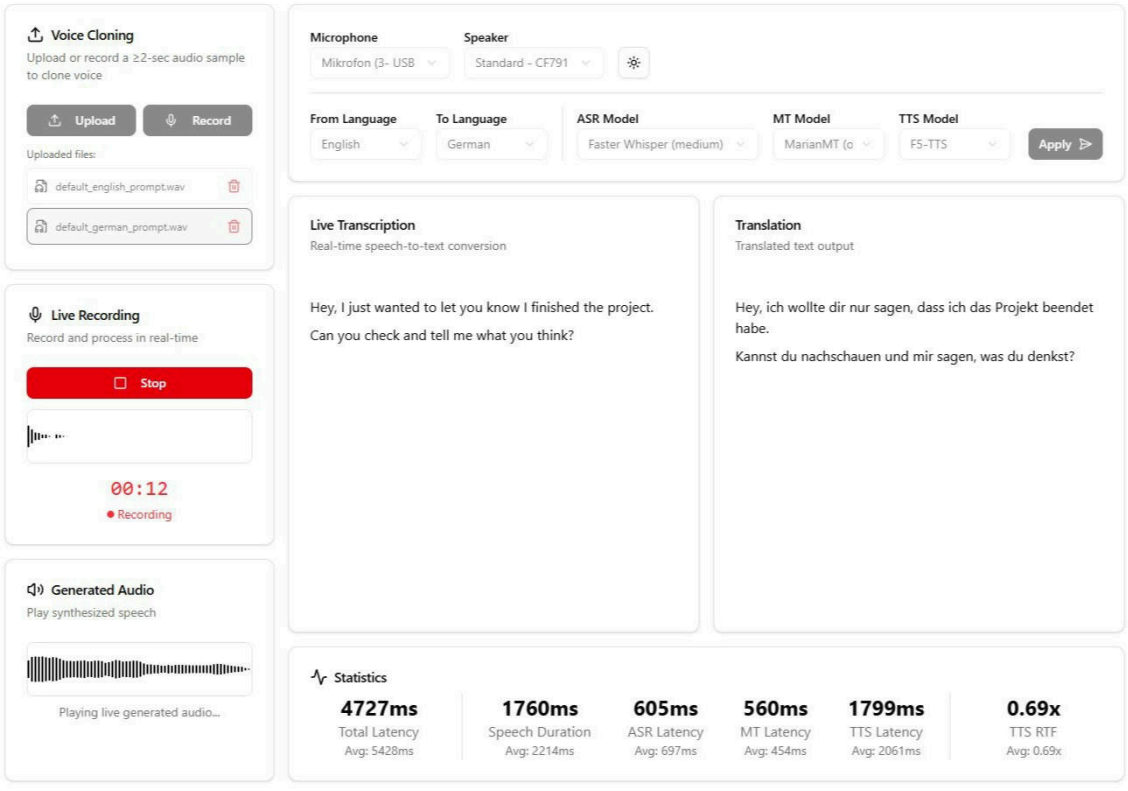


Figure 1: Application screenshot: recording state

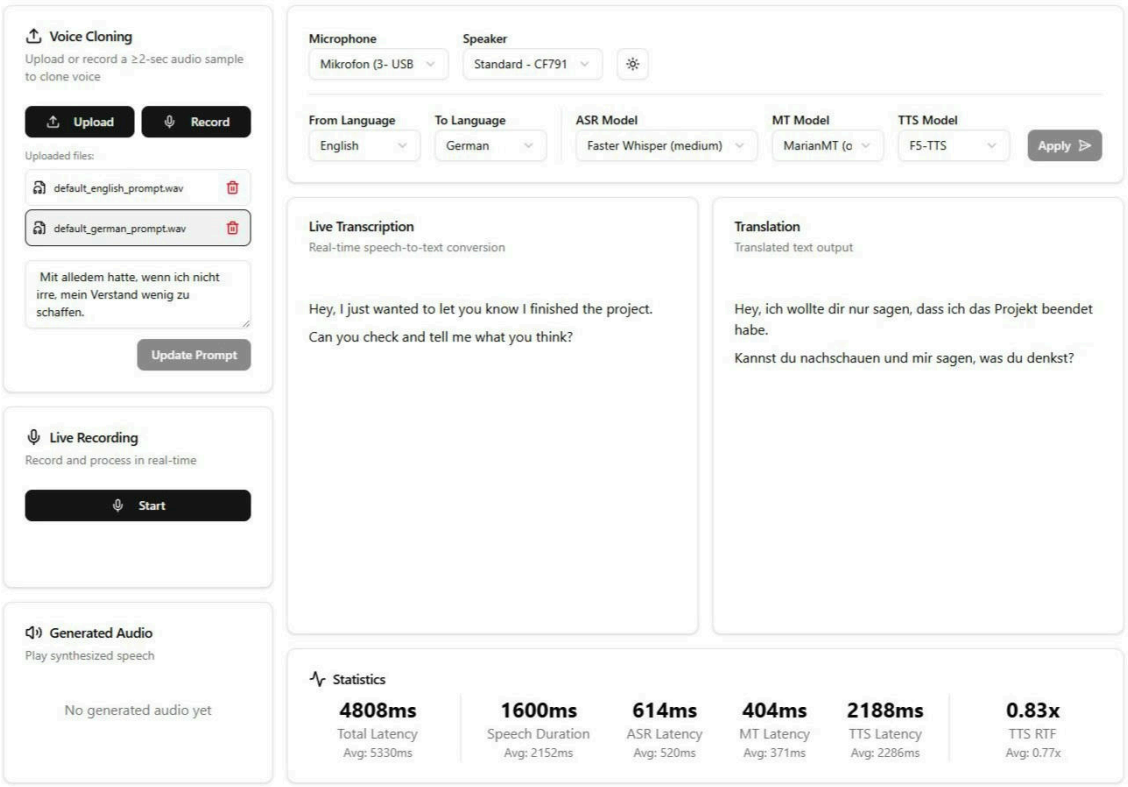


Figure 2: Application screenshot: idle state

2 Introduction

2.1 Initial situation

The increasing capabilities of neural speech synthesis have enabled voice cloning from just a brief audio sample, opening opportunities for applications such as real-time speech translation with voice preservation. However, these capabilities also raise significant concerns, including potential misuse for impersonation, fraud, or misinformation, as well as broader privacy implications.

While commercial solutions for real-time speech translation exist, they often require cloud-based infrastructure, limiting accessibility and transparency. At the same time, open-source models for automatic speech recognition (ASR), machine translation (MT), and text-to-speech (TTS) synthesis have improved substantially, but their performance in fully real-time, low-latency pipelines is not yet comprehensively evaluated. Existing research often focuses on individual components (e.g., voice cloning or speech translation), leaving limited evaluation in the practical feasibility of real-time, voice-preserving translation on consumer hardware.

This work aims to address this gap by investigating whether open-source models can be orchestrated in a system to provide low-latency, perceptually convincing real-time speech translation while preserving the speaker's voice.

2.2 Task

The main objective of this thesis is to develop and evaluate a system for low-latency real-time speech translation with voice preservation. The system is designed to transcribe speech through a microphone in German, English, French, or Italian, translate it into a target language, and synthesise the translated speech while preserving the speaker's voice.

Specific goals include:

1. Real-time speech-to-text transcription with minimal latency.
2. Incremental translation of the transcribed text into a target language (German, English, French, or Italian).
3. Speech synthesis with natural prosody and voice preservation.
4. Integration of open-source models into a modular pipeline that can run on consumer-grade hardware.
5. Evaluation of the system's transcription- and translation quality, naturalness, intelligibility, voice similarity, and end-to-end latency.

The work follows a structured development process, including analysis of functional and non-functional requirements, model selection and evaluation, architecture design, system implementation, and performance testing. Detailed requirements and evaluation criteria are presented in Chapter 3 and Chapter 4.

2.3 Technologies

Backend	Python with FastAPI
Frontend	React Typescript
ASR Model	Faster-Whisper
TTS Model	F5-TTS, VoxCPM, XTTSv2
MT Model	MarianMT, M2M100
Enviroment	Docker
Documentation	Typst

Table 2: Technologies used

2.4 Framework Conditions

This student research project is carried out as a two-person project, with each student contributing approximately 240 hours, totalling 8 ECTS credits per student. The work is supervised by Mitra Purandare and co-supervised by Markus Stolze at OST Rapperswil. For further information about the module, see Appendix Section G.4.

The duration of the project was from September 15, 2025, to December 19, 2025. It was organised using the software development methodologies SCRUM and Rational Unified Process (RUP), with weekly planning, progress tracking, and regular review meetings to ensure iterative development and early identification of issues. More information can be found in Appendix Section G.1.

2.5 System Context and Domain

The system is implemented as a web application and is accessed through a standard web browser. Its primary external interface is a continuous audio input provided by the browser. All processing steps are performed locally within the system via well-defined internal interfaces between the speech recognition, translation, and text-to-speech components.

Real-time speech translation imposes specific requirements on system design, particularly concerning continuous data processing, incremental result generation, and strict latency constraints. The system must balance competing objectives such as accuracy, naturalness of synthesised speech, and overall latency. An additional domain-specific challenge is the preservation of the speaker's voice through zero-shot voice cloning, which extends beyond conventional text-to-speech use cases and significantly influences architectural and design decisions.

Part II: Product Documentation

3 Functional Requirements

3.1 Actors

As the system under development is designed to prove the concept of real-time speech-to-speech with voice conversion, only one actor is using the prototype.

3.2 Epics

Name	Prio
Automatic Speech Recognition	1
Text to Speech	1
Machine Translation	2
Configuration / Statistics	2

Table 3: Epics

3.3 Stories

Identifier	Epic	Requirement	Prio
FR01	Automatic Speech Recognition	I want to record myself, so I can use my speech as source input	1
FR02	Automatic Speech Recognition	I want to see a transcription of what I say in real time, so I have visual feedback	1
FR03	Text to Speech	I want to upload a voice reference so that I can use that as the reference input	1
FR04	Text to Speech	I want to hear the output speech in real time, so that I can sound like the target speaker	1
FR05	Machine Translation	I want to choose the output language between English, German, French and Italian, so that the system can translate my speech into that language.	2
FR06	Machine Translation	I want to see the translated transcription in real time, so I have visual feedback	2
FR07	Configuration / Statistics	I want to switch between different ASR, MT and TTS models, so that I can optimise the pipeline for my requirements	2
FR08	Configuration / Statistics	I want to observe the relevant stats of the process, such as latency and real-time factor, so I have information about the performance of the output.	2

Table 4: Stories

4 Non Functional Requirements (NFR)

ID	**NFR-01** End to End Latency
Category	Performance
Description	The system must provide near real-time ASR, MT, TTS with voice cloning.
Goal	Minimal: 5s latency Target: 4s latency Outstanding: 3s latency
Verification Method	Measure average total end-to-end latency.
Priority	High

Table 5: NFR-01 Performance

ID	**NFR-02** Transcription Accuracy
Category	Functionality
Description	ASR should accurately transcribe spoken input.
Goal	Minimal: 85% word accuracy Target: 90% Outstanding: 95%
Verification Method	Compare ASR output to human reference transcripts.
Priority	High

Table 6: NFR-02 Functionality

ID	**NFR-03** Translation Quality
Category	Functionality
Description	Translations must be correct and understandable.
Goal	Minimal: BLEU 25 Target: BLEU 30 Outstanding: BLEU 35
Verification Method	Automatic metric evaluation and human assessment.
Priority	Medium

Table 7: NFR-03 Functionality

ID	**NFR-04** Natural-Sounding Speech
Category	Functionality
Description	The system must generate natural and intelligible speech.
Goal	Minimal: MOS 3.0/5 Target: MOS 4.0/5 Outstanding: MOS 5/5
Verification Method	Conduct user MOS tests on synthesised speech.
Priority	High

Table 8: NFR-04 Functionality

ID	**NFR-05** Voice Preservation
Category	Functionality
Description	Synthesised speech should retain similarity to the original speaker.
Goal	Minimal: 0.6 similarity Target: 0.7 Outstanding: 0.8
Verification Method	Run automated cosine similarity calculation and perform a subjective similarity test with human evaluators.
Priority	High

Table 9: NFR-05 Functionality

ID	**NFR-06** Intelligibility
Category	Functionality
Description	Synthesised speech should be intelligible.
Goal	Minimal: 85% word accuracy Target: 90% Outstanding: 95%
Verification Method	Run ASR on speech output and calculate WER by comparing it with the input.
Priority	High

Table 10: NFR-06 Functionality

ID	**NFR-07** Usability
Category	Usability
Description	New users should be able to perform key actions of the application without having to refer to a manual or getting help from another person.
Goal	
Verification Method	Have people without prior experience with the application achieve certain goals within the application.
Priority	Medium

Table 11: NFR-07 Usability

ID	**NFR-08** Adaptability
Category	Supportability
Description	The software should be usable (all functional requirements fulfilled) on the most used desktop browsers according to Wikipedia: https://en.wikipedia.org/wiki/Usage_share_of_web_browsers
Goal	Minimal: 2 of the most used browsers Target: 4 of the most used browsers Outstanding: 5 of the most used browsers
Verification Method	Manually go through each functional requirement with each browser.
Priority	Low

Table 12: NFR-08 Supportability

5 Model Evaluation

Before the implementation, different ASR, MT and TTS models were evaluated based on performance and quality to decide which models are to be integrated into the system.

5.1 ASR Models

For the evaluation of the ASR models a RTX 2070 Super was used. The latency of the batch-based models is measured with a generated audio of roughly 2 seconds. The **Word Error Rate** values presented in this documentation are taken from publicly available benchmarks.

Model	Languages	Streaming	Word Error Rate	Latency
Faster Whisper (multilingual) [1]	All relevant	Batch based	clean: • tiny: 19% • base: 15% • small: 11% • medium: 8% • large-v3: 5% • large-v3-turbo: 5%	• tiny: 70ms • base: 80ms • small: 110ms • medium: 180ms • large-v3: 280ms • large-v3-turbo: 250ms
Faster Whisper (English only) [1]	English	Batch based	clean: • tiny: 18% • base: 14% • small: 10% • medium: 7%	• tiny: 50ms • base: 50ms • small: 50ms • medium: 60ms
Crisper Whisper [2]	English & German	Batch based	clean: 6%	280ms
Whisper (original) [3]	All relevant	Batch based	• tiny: 19% • base: 15% • small: 11% • medium: 8% • large-v3: 5%	• tiny: 400ms • base: 450ms • small: 600ms • medium: 1s • large-v3: 5s
Vosk [4]	All relevant	Full Streaming	clean: • small: 10% • large: 6%	• small: 30ms • large: 50-120ms
Wav2Vec2 [5]	All relevant	Batch based	5-15% depending on language/noise	1s + 5s chunk
Coqui STT [6]	All relevant	Full Streaming	• clean: 6-8% • real-world: 10-12%	40-200ms
Parakeet-TDT (NVIDIA NeMo) [7]	English	Batch based	• clean: 5-7% • real-world: 10-15%	20-40ms chunk + 50-150ms
Seamless S2TT [8]	All relevant	Batch based	• clean: 6% • real-world: 14%	2s chunk

Table 13: ASR Models Capabilities

5.1.1 Model evaluation

Weight (%)	30	30	20	10	10	100
Model	Word Error Rate	Latency	Language Coverage	Streaming	Ease of Integration	**Total**
Faster Whisper	4	4	5	3	4	4,1
Faster Whisper (EN)	4	5	2	3	4	3,8
Crisper Whisper	5	2	3	3	4	3,4
Whisper (original)	4	1	5	3	4	3,2
Vosk	3	5	5	5	5	4,4
Wav2Vec2	3	2	5	3	3	3,1
Coqui STT	4	4	5	5	4	4,3
Parakeet-TDT	5	5	2	3	3	4
Seamless S2TT	4	3	5	3	3	3,7

Figure 3: Evaluation: ASR models

5.1.2 Evaluation of ASR models with punctuation

Since only the Whisper-based models support punctuation restoration, they were evaluated separately to enable a more fine-grained comparison. At the time of evaluation, punctuation was considered a key requirement, as it was intended to be used for segmenting the input audio stream into semantically coherent units that could be used downstream.

The Original Whisper model was not included in this evaluation because, as observed previously, its latency is significantly higher than that of the other models, making it impractical for the targeted real-time use case.

To allow a more differentiated comparison, decimal values were introduced for the evaluation scores, and a higher weighting was assigned to the Word Error Rate compared to latency.

Weight (%)	60	40	100
Model (Size)	Word Error Rate	Latency	**Total**
Faster Whisper (all languages)			
tiny	1	4,6	2,44
base	2,9	4,4	3,50
small	3,5	3,9	3,66
medium	4,1	3,1	3,70
large-v3	5	2	3,80
large-v3-turbo	5	2,3	3,92
Faster Whisper (English)			
tiny	1,3	5	2,78
base	3,2	5	3,92
small	4	5	4,40
medium	4,4	4,8	4,56
Crisper Whisper (English & German)			
large	5	2	3,80

Figure 4: Evaluation: ASR models with punctuation

5.1.3 Selected ASR Models

Based on the evaluation, the following ASR models are implemented.

- **Faster Whisper (English, medium)** is chosen for English-only recognition. It provides very low latency and high transcription accuracy, making it ideal for real-time speech processing.
- **Faster Whisper (Multilingual, medium)** is selected as the general-purpose model for multilingual transcription. It maintains good accuracy across various languages with decent latency.
- **Faster Whisper (Multilingual, large-v3-turbo)** is included as a high-accuracy option for demanding use cases where slightly higher latency is acceptable.

Notable rejected option

- **Vosk** offers full streaming support and extremely low latency, but it lacks punctuation restoration. At the time of evaluation, punctuation was considered to be essential for splitting the input stream into semantic segments, which are then passed as coherent batches to the machine translation model.
- **Crisper Whisper** may also seem like a good option, especially since it does not omit disfluencies and aims to transcribe every spoken word exactly as it is, including fillers, pauses, and stutters. However, in practice, it frequently produces unusable output: it can omit spaces between words entirely, mishandle punctuation by leaving it out or inserting commas between every word, and thus produce text that is not suitable for downstream processing.

5.1.4 Design evolution and methodological adjustment

During later stages of system development, the segmentation strategy of the system was revised. Instead of relying on punctuation marks to split the input at contextually meaningful positions, the final system uses a VAD-based segmentation approach. This method proved to be more robust in streaming scenarios and reduced the dependency on punctuation quality.

As a consequence, punctuation is no longer required for stream segmentation in the final architecture. Nevertheless, punctuation remains relevant for transcript readability and downstream processing. The punctuation-focused evaluation is therefore retained to document the design considerations and trade-offs explored during model selection.

5.2 MT Models

Model	**Languages**	**Streaming**	**Translation Quality**	**Latency**
MarianMT [9]	All relevant	Batch based	30-40 BLEU	100-300ms
OpenNMT [10]	English ↔ German	Batch based	25-30 BLEU	50-150ms
Argos Translate [11]	All relevant	Batch based	25-30 BLEU	200-500ms
M2M100 [12]	All relevant	Batch based	30-35 BLEU	200-500ms
MBART [13]	All relevant	Batch based	30-35 BLEU	200-600ms
LLaMA 2 (LLM) [14]	All relevant	Batch based	30-35 BLEU	1-2s
MPT-7B-Instruct (LLM) [15]	All relevant	Batch based	30-35 BLEU	1-2.3s

Table 14: MT Models Capabilities

5.2.1 Model evaluation

Weight (%)	35	35	20	10	100
Model	Translation Quality	Latency	Language Coverage	Ease of Integration	**Total**
MarianMT	5	4	5	4	4,55
OpenNMT	3	5	2	4	3,6
Argos Translate	3	3	5	4	3,5
M2M100	4	3	5	4	3,85
MBART	4	2	5	3	3,4
LLaMA 2 (LLM)	4	1	5	1	2,85
MPT-7B-Instruct (LLM)	4	1	5	1	2,85

Figure 5: Evaluation: MT models

5.2.2 Selected MT Models

Based on the evaluation, the following machine translation models are implemented.

- **MarianMT** is selected as the main translation model for all target languages. It provides higher translation quality (BLEU ∼30–40) and lower latency (∼100–300 ms), making it the preferred choice for the real-time pipeline.
- **M2M100** is used as a complementary model. It has lower translation quality (BLEU ∼30–35) and higher latency (∼200–500 ms).

Notable rejected options

- **MBART** and **Argos Translate** deliver reasonable translation quality but show higher latency and less consistent performance across languages.
- Large language models such as **LLaMA 2** or **MPT-7B-Instruct** achieve good translation quality but introduce excessive latency and need a lot of resources, making them unsuitable for real-time speech translation.
- **OpenNMT** was considered for English–German translation, but it was not implemented due to time constraints. While it performs well, the focus was placed on MarianMT and M2M100 to ensure a working real-time translation pipeline.

5.3 TTS Models

For the evaluation of the TTS model a RTX 4080 Super was used, and the models were executed in WSL. The latency is measured with a generated audio of roughly 2 seconds and doesn't include the 2 seconds needed for the text to be present. The speech quality metric is based on our subjective impression.

The real-time factor is the inference time / generated audio time. For streaming, a real-time factor of below 1 is needed, or else the output will have empty pauses.

The latency defines the time until the audio can be played. For batch-based models, this is just the inference time, while for streaming models, it's the time until the first chunk is generated.

Model	Voice Cloning	Languages	Streaming	Latency	RTF
XTTSv2 [16]	Zero-Shot	All relevant	Output streaming	0.9s	0.9
Chatterbox [17]	Zero-Shot	All relevant	Batch based	2.6s	1.3
VoxCPM [18]	Zero-Shot	English	Output streaming	300ms	0.4
Cosyvoice 2 [19]	Zero-Shot	English	Output streaming	1.7s	2.2
FishSpeech [20]	Zero-Shot	All relevant	Batch based	3.8s	1.9
VoXtream [21]	Zero-Shot	English	Full Streaming	300ms	1.6
F5 TTS German [22]	Zero-Shot	German	Batch based	1.1s	0.5

Table 15: TTS Models Capabilities

5.3.1 English Models

Weight (%)	40	20	20	10	10	100
Model / Framework	Real Time Factor	Latency (seconds)	Voice Preservation	Speech Quality	Ease of integration	**Total**
XTTSv2	4 (0.9)	5 (0.9)	4	3	3	4
Chatterbox	2 (1.3)	3 (2.6)	5	5	5	3,4
VoxCPM	5 (0.4)	5 (0.3)	4	4	4	4,6
Cosyvoice 2	1 (2.2)	4 (1.7)	4	3	2	2,5
Openaudio s1 mini	1 (1.9)	2 (3.8)	4	3	1	2
VoXtream	1 (1.6)	5 (0.3)	4	2	4	2,8

Figure 6: Evaluation: TTS Models (English)

5.3.2 German Models

Weight (%)	40	20	20	10	10	100
Model / Framework	Real Time Factor	Latency (seconds)	Voice Preservation	Speech Quality	Ease of integration	**Total**
XTTSv2	4 (0.9)	5 (0.9)	3	2	3	3,7
Chatterbox	2 (1.3)	3 (2.56)	5	5	5	3,4
F5 TTS German	5 (0.5)	4 (1.1)	4	4	4	4,4
Cosyvoice-EU	1 (2.2)	4 (1.7)	4	3	4	2,7
Openaudio s1 mini	1 (1.9)	2 (3.8)	4	3	1	2

Figure 7: Evaluation: TTS Models (German)

5.3.3 Selected TTS Models

Based on the evaluation, the following TTS models are implemented.

- **VoxCPM** is the most performative of all the models with a very low latency and RTF and an acceptable speech quality, making it perfect for a real-time application. Unfortunately, the model is only trained on English data.
- **F5 TTS German** is the best option for German speech synthesis, although it doesn't allow for streaming, the latency and RTF are still quite low.
- **XTTSv2** is the allrounder option. It has acceptable latency and RTF, works with all relevant languages and has not the best but acceptable speech quality.

Notable rejected options

- **Chatterbox** had the best speech quality and supported most European languages, but it fell short in performance, which doesn't make it viable for real-time use.
- **VoXtream** allows for full streaming (input and output can be streamed), which, in theory, is perfect for real-time applications, but the output speech is very robotic and doesn't hold up to the other models. Additionaly the RTF during our tests was quite high and would have needed more powerful hardware and optimisation.

5.4 STS Models

There are some direct Speech to Speech models that allow for voice conversion without going through the ASR - TTS pipeline, but they are limited and do not perform well with noisy reference audio and have worse quality in general. In addition, these approaches copy the pacing and speed of the input speaker and not the reference speaker, which makes them less convincing.

Based on the above, we decided not to implement STS Models in our solutions.

Model	**Voice Cloning**	**Languages**	**Streaming**	**Radio Quality**	**Latency**
SeedVC [23]	Zero-Shot	English	Full Streaming	Poor	900ms
RVC [24]	Training	English	Full Streaming	Not tested	Not tested

Table 16: STS Models Capabilities

6 Architecture

6.1 Container Diagram

Figure 8: High-Level architecture container diagram

Each container in the diagram handled by the Orchestrator (Backend) represents one model instance. The total number of containers grows as more models are added.

6.2 Sequence Diagrams

6.2.1 Initiating connection

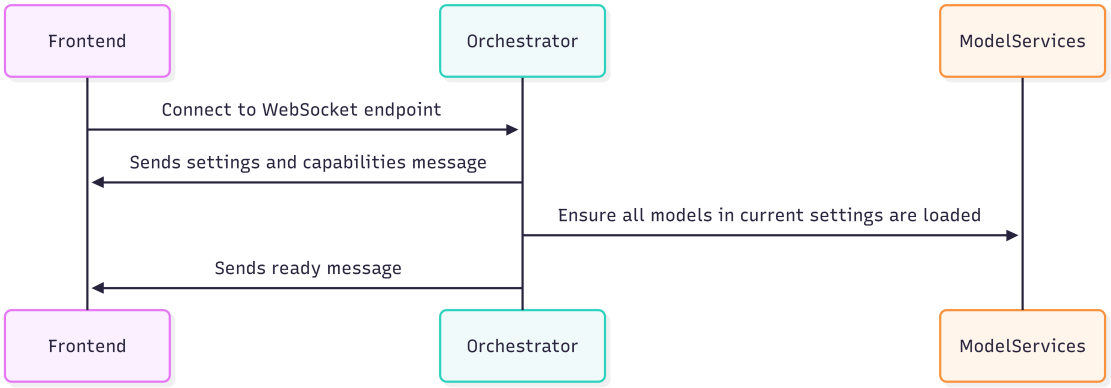


Figure 9: Activity Diagram: Initiating connection

In the diagram above, Model Services refers to the different containers hosting the models, such as ASR and TTS, while the Orchestrator refers to our Backend.

6.2.2 Updating Settings

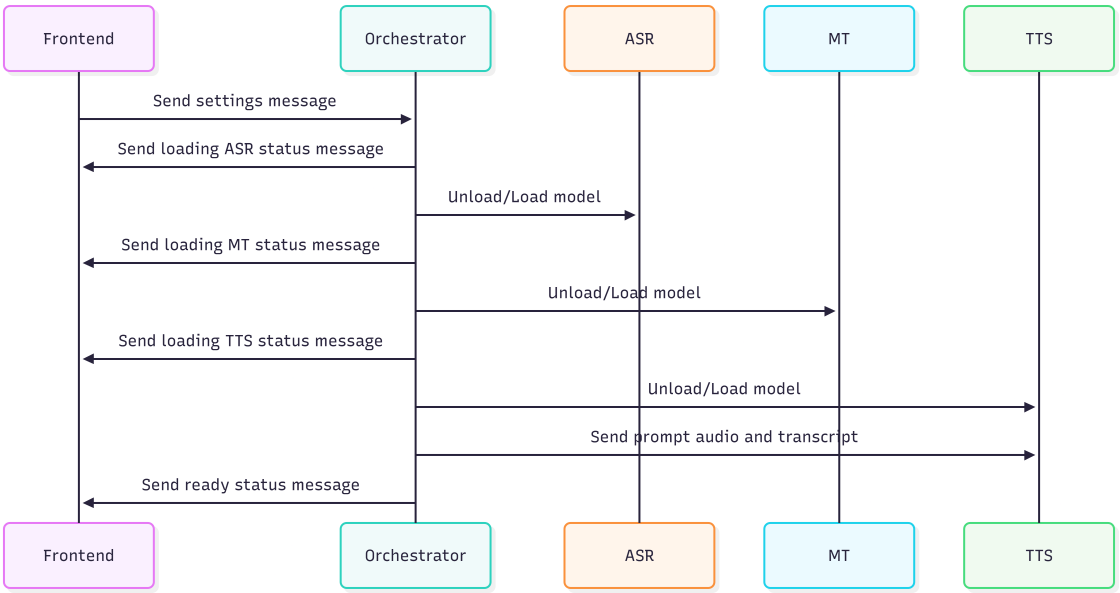


Figure 10: Activity Diagram: Updating Settings

6.2.3 Updating Prompt

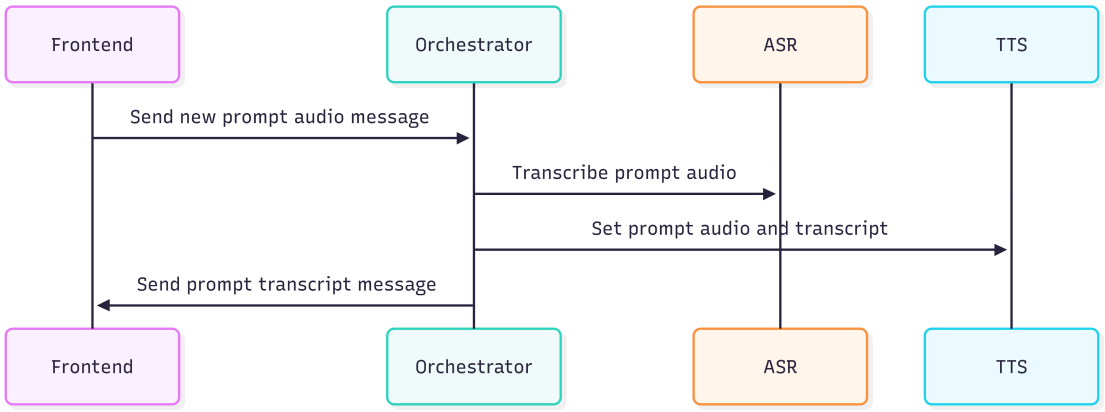


Figure 11: Activity Diagram: Updating Prompt

6.2.4 Streaming Audio

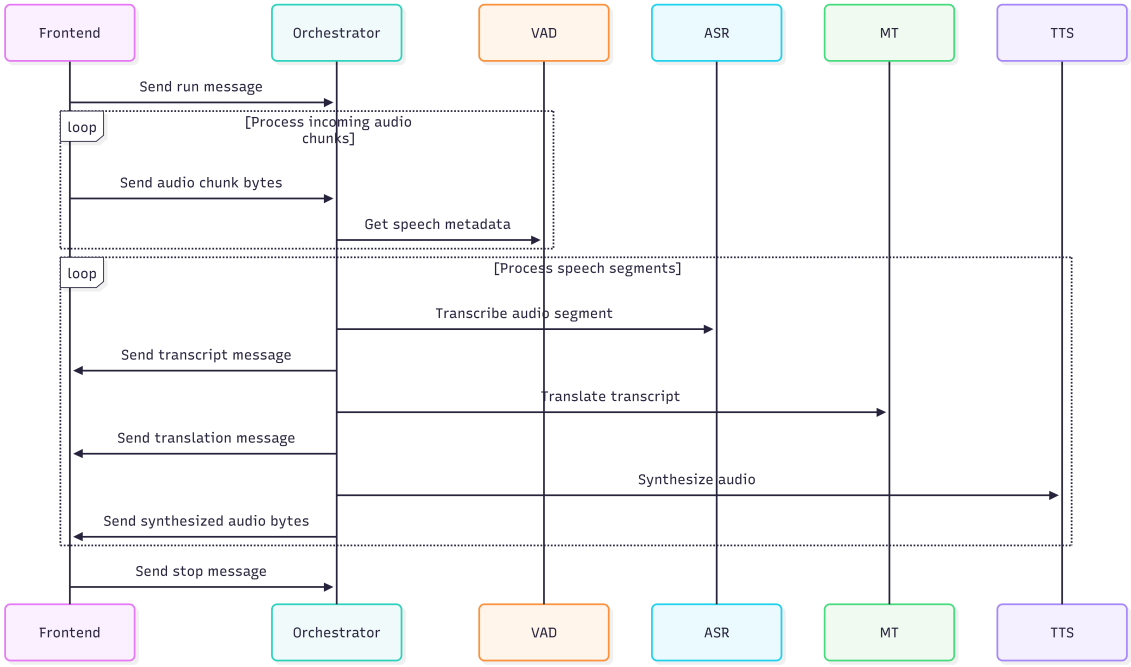


Figure 12: Activity Diagram: Streaming Audio

In the first loop, the orchestrator uses the VAD to detect speech segments, and in the second loop, the orchestrator loops over the detected speech segment to run the different inferences.

6.3 Orchestrator Component Diagram

The backend is designed to allow for multiple model implementations and to allow them to be swapped out with minimal code changes.

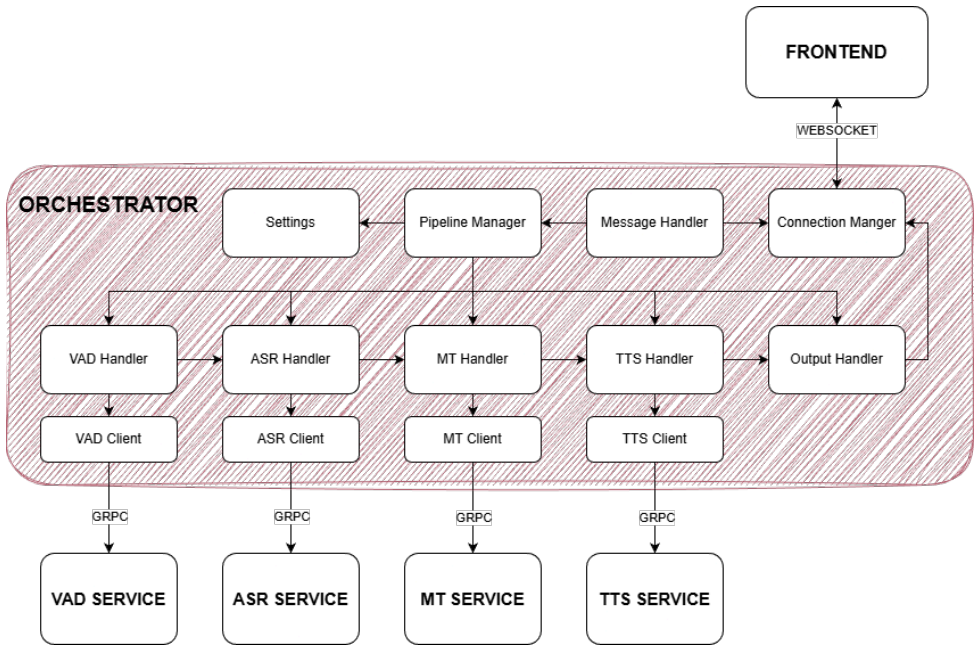


Figure 13: Orchestrator component diagram

6.4 Frontend Component Diagram

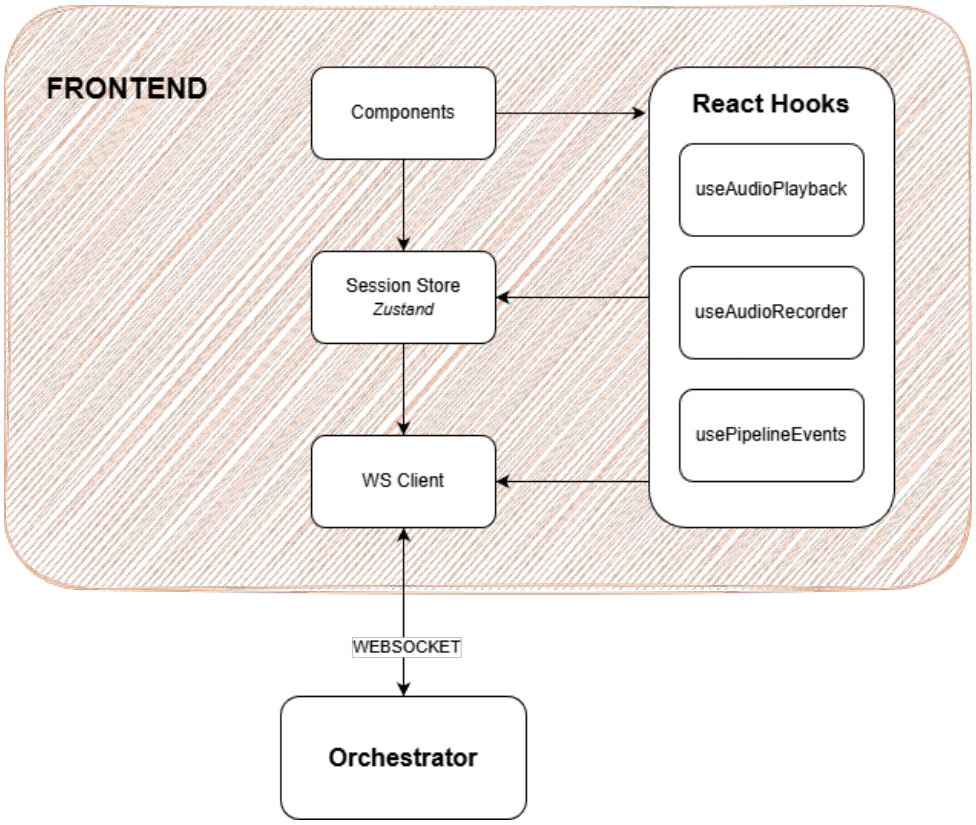


Figure 14: Frontend component diagram

6.5 Model Services

For each model type, a GRPC Protocol Buffers are defined. Models implement these and the model they are hosting, and serve it as a GRPC service. TTS model services are designed to use the model in a subprocess, so that the process can be cancelled when unloading the model, and it is ensured that the memory is freed.

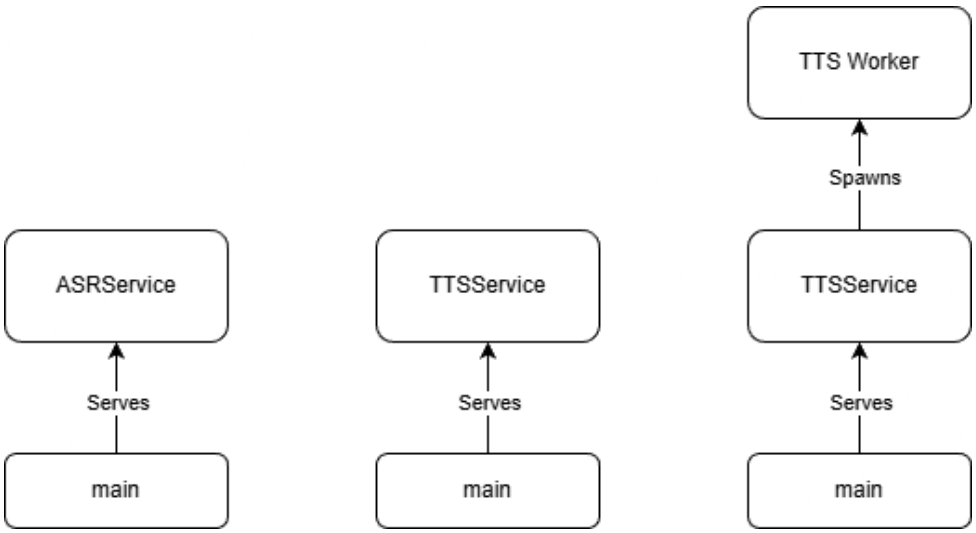


Figure 15: Model services component diagram

6.5.1 ASR Protocol Buffer

Endpoint	Request	Response
LoadModel	Empty	Success boolean
Inference	Audio segment, language	Transcript
UnloadModel	Empty	Success boolean
ChangeCheckpoint	Checkpoint	Success boolean

Table 17: ASR Protocol Buffer

6.5.2 MT Protocol Buffer

Endpoint	Request	Response
LoadModel	Empty	Success boolean
Inference	Text, source language, target language	Translation
UnloadModel	Empty	Success boolean
ChangeCheckpoint	Checkpoint	Success boolean

Table 18: MT Protocol Buffer

6.5.3 TTS Protocol Buffer

Endpoint	Request	Response
LoadModel	Empty	Success boolean
Inference	Text, language	Generated audio segment
UnloadModel	Empty	Success boolean
SetPrompt	Prompt audio and transcript	Success boolean

Table 19: TTS Protocol Buffer

6.6 Docker Setup

The Docker setup is configured using the docker-compose.yaml. For both production and specific model tests, there are separate Docker Compose configurations.

Container	Base Image	Note
Frontend	Node Alpine / Caddy Alpine	In the production Docker Compose, the frontend is hosted using caddy while for development vite is used
Orchestrator	Python slim	
Model Services	PyTorch	Each service hosting a model is in its own container and the GPU is exposed to the container.

Table 20: Docker Containers

6.7 Websocket

For communication between the frontend and backend, WebSocket is used, which allows bidirectional communication and is perfect for low-latency use cases like this. As different messages and streams should be transferred through one WebSocket connection, we defined the following message types:

6.7.1 Frontend Messages

These are the messages sent from the frontend to the orchestrator.

Message	Type	Content
Settings	JSON	Translation, model selection
Prompt	JSON	Prompt audio
Prompt Transcript	JSON	Prompt transcript adjustments
Start	JSON	Start flag
Audio input	Bytes	Input audio stream
Stop	JSON	Stop flag

Table 21: Websocket frontend messages

6.7.2 Orchestrator Messages

These are the messages sent from the orchestrator to the frontend.

Message	Type	Content
Settings	JSON	Initial settings
Capabilities	JSON	Supported models and their capabilities
Ready	JSON	Ready signal when the orchestrator has loaded all models
Stat output	JSON	Segment latencies and RTF
ASR output	JSON	Transcript segment
MT output	JSON	Translated segment
TTS output	Bytes	Output audio segment
Stat output	JSON	Segment statistics
Loading Info	JSON	Loading subject and percentage

Table 22: Websocket orchestrator messages

6.8 Frontend Mockups

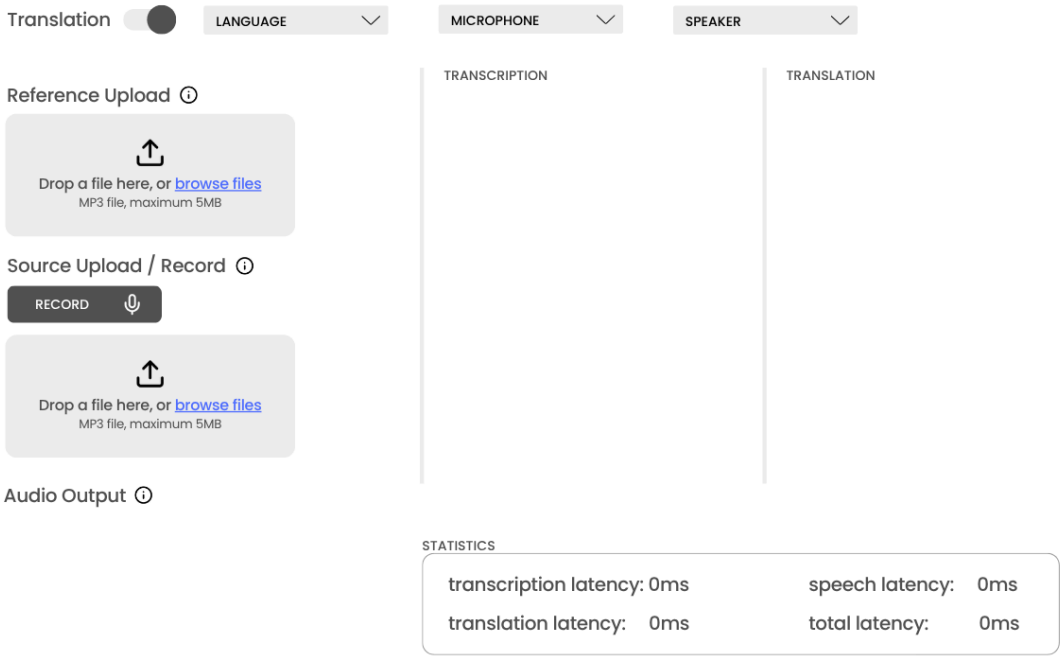


Figure 16: Mockup: Initial State

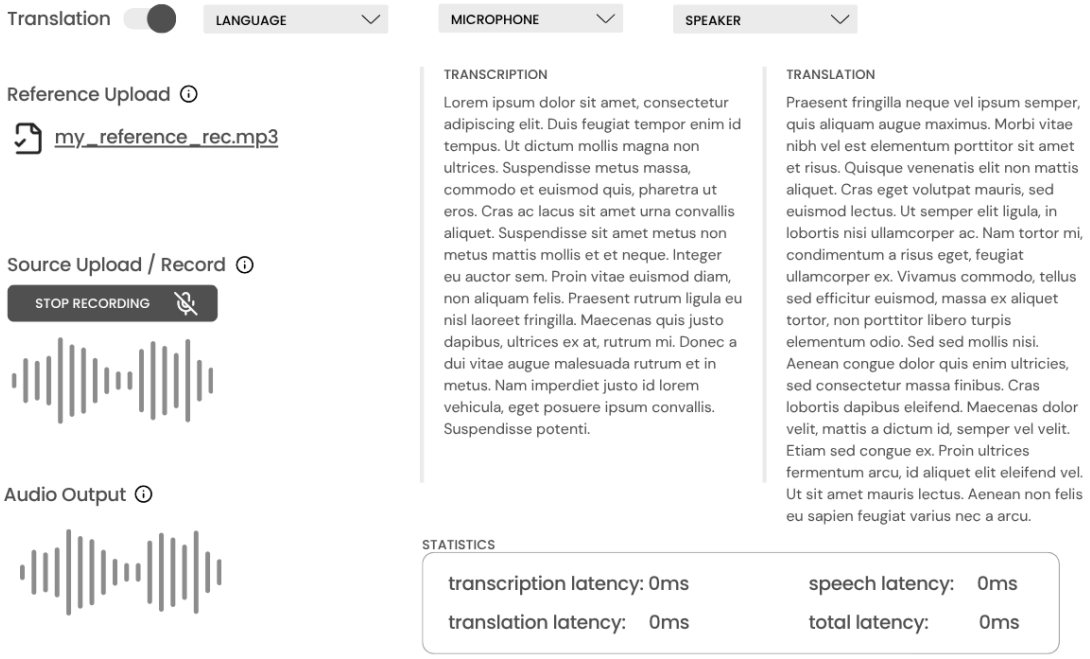


Figure 17: Mockup: Recording State

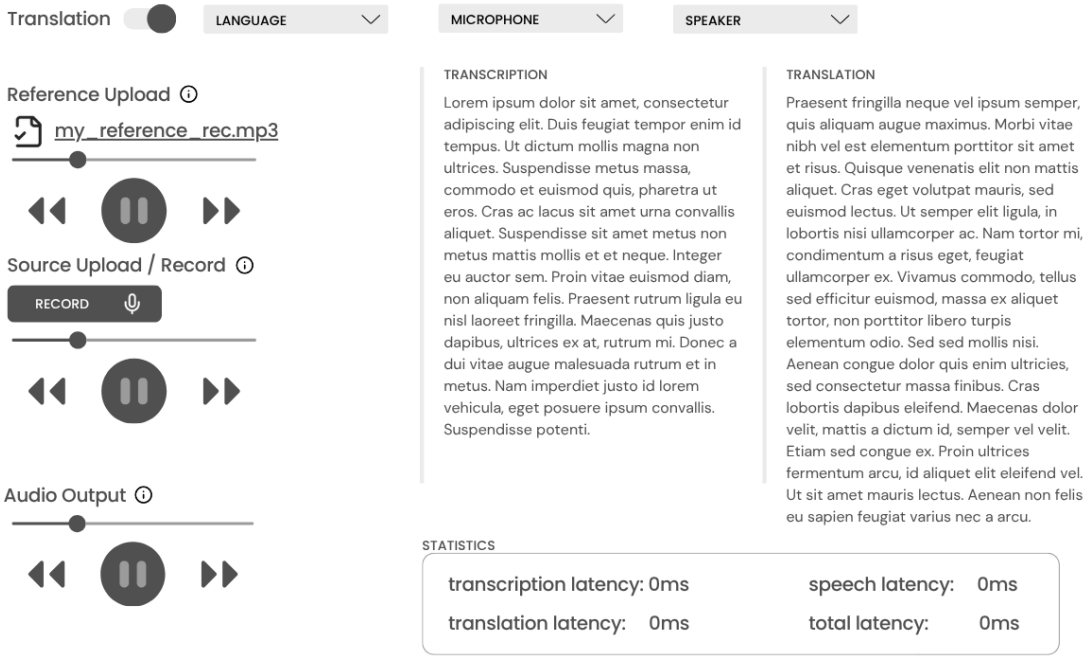


Figure 18: Mockup: End State

6.9 Scaling

The system is intentionally designed for a single active user at a time. In its current implementation, the backend allows exactly one concurrent WebSocket connection, which simplifies orchestration, state handling, and model interaction. This design choice was made deliberately to focus on feasibility and low latency rather than throughput.

Scaling this system is therefore not a matter of infrastructure alone; it would require changes to the application logic and model execution.

6.9.1 Vertical Scaling

In theory, the system could be scaled vertically by deploying it on more powerful hardware. This approach would primarily require GPUs capable of handling multiple concurrent inference workloads, especially for TTS, which is the most computationally demanding component.

However, vertical scaling introduces several challenges:

- Most open-source ASR, MT and TTS models typically expose research-grade APIs designed for single requests. These models are not optimised for concurrent inference. Supporting multiple parallel requests would require loading separate model instances into VRAM, significantly increasing memory consumption.
- The entire orchestrator and model service code would have to be refactored to be able to handle concurrency.

6.9.2 Horizontal Scaling

Horizontal scaling represents a more realistic but still complex approach. This would involve:

- The entire orchestrator and model service code would have to be refactored to be able to handle concurrency.
- Running multiple instances of the model services and the orchestrator
- Distributing requests across instances using a load balancer.

While this approach avoids running multiple concurrent inferences within a single model instance by isolating each inference in its own container, it is inefficient in terms of resource utilisation and increases operational complexity.

6.9.3 Scaling for the Number of Models

Independent of user concurrency, the current architecture also faces limitations when scaling the number of supported models. Each model is hosted in its own container, which simplifies isolation and experimentation but does not scale well as the system grows.

As more models are added, the deployment becomes increasingly unwieldy due to:

- A rapidly growing number of containers to manage and orchestrate.
- Increased startup times and higher baseline resource consumption.
- Redundant dependencies across containers, particularly for transformer-based models with similar runtime requirements.

Future implementations could address this by adopting alternative model-hosting strategies. Where dependencies allow, multiple models could be served from shared containers to reduce overhead. More fundamentally, dedicated model-serving solutions optimised for inference workloads—particularly for TTS—could provide a cleaner and more scalable abstraction for hosting many models without linear increases in deployment complexity.

6.9.4 Conclusion

Scaling real-time, ML-driven systems is inherently complex and differs significantly from scaling conventional web applications. In this thesis, scalability was consciously deprioritised in favour of developing a functioning prototype within the available time frame, with an emphasis on low latency and experimental flexibility.

A production-ready system would require both concurrency-aware execution models to support multiple users and more scalable model-hosting strategies to manage a growing set of models. Addressing either dimension properly would involve significant architectural changes and likely constitute a substantial research effort on its own, and is therefore considered out of scope for this thesis.

6.10 Architectual Decision Records

During the thesis, we faced a lot of architectural decisions, so-called "Big Decisions" as well as smaller ones. We have made all of them at the Most Responsible Moment and recorded them in the Y-statement style.[1]

Date:	2025-10-05
Context:	Repository
Facing:	The need to define the structure of our repository
We decided:	To keep our code inside one repository
And neglected:	Splitting frontend and Backend into 2 separate repositories
To achieve:	Less complicated build process
Accepting that:	A lightly bigger code repository

Table 23: ADR 001

Date:	2025-10-05
Context:	Frontend
Facing:	The decision between a monolithic application and a modular web-based architecture
We decided:	To use React with TypeScript and WebSocket
And neglected:	a monolithic application
To achieve:	• Real-time, low-latency communication between frontend and backend • Less time needed for familiarization as the team already has experience with React • Easier scalability and maintainability over time • Strong typing with TypeScript, reducing runtime errors • Smooth integration with Python AI backend services via WebSocket
Accepting that:	• React's flexibility requires clear architectural conventions to maintain consistency • WebSockets introduce complexity in managing connections and error handling • Slightly more network overhead compared to a fully monolithic application

Table 24: ADR 002

[1] Olaf Zimmermann, "Y-Statements," Medium, 2020, https://medium.com/olzzio/y-statements-10eb07b5a177.

Date:	2025-10-05
Context:	Frontend, Orchestrator
Facing:	The need for a communication protocol between the frontend and orchestrator
We decided:	To use WebSocket
And neglected:	REST with SSE, GRPC, REST with WebRTC
To achieve:	• Streaming audio between the frontend and orchestrator • Low latency • Bi-directional communication
Accepting that:	• A message protocol has to be defined. • There is no strong contract between the frontend and the orchestrator.

Table 25: ADR 003

Date:	2025-10-05
Context:	Orchestrator
Facing:	The need for a backend framework
We decided:	To use Python with FastAPI
And neglected:	.NET and Django or Flask with Python
To achieve:	• Less time needed for familiarization as there is already experience with Python in the team • Fast API supports WebSocket communication • One less technology in the stack as the model services need to use Python
Accepting that:	• Python enforces no typing by itself • FastAPI is relatively young compared to Flask or Django

Table 26: ADR 004

Date:	2025-10-05
Context:	Documentation
Facing:	The need to ensure easy documentation
We decided:	To use Typst
And neglected:	LaTeX, Word
To achieve:	• Saving time when writing documentation • Live collaboration using the Typst web editor • Pragmatic and intuitive syntax
Accepting that:	There is no automatic versioning of the documentation.

Table 27: ADR 005

Date:	2025-10-07
Context:	Orchestrator, Model Services
Facing:	The decision if the application is to be developed to support concurrency
We decided:	To only support one user at a time
And neglected:	An application supporting multiple users concurrently
To achieve:	• Significantly reduced complexity, as there is no need to try to make the ML models work concurrently
Accepting that:	• Only one user at a time can use the application • Making the application scalable would need a lot of effort

Table 28: ADR 006

Date:	2025-10-07
Context:	Orchestrator, Model Services
Facing:	The need to establish a boundary between the orchestrator and the model services
We decided:	To separate the model services into different containers
And neglected:	Monolith, Containers hosting multiple model services
To achieve:	• Separation of concerns • Ensure that there are no dependency conflicts between the model services • Low latency
Accepting that:	Having a container for each model supported adds a huge amount of complexity and makes development time-consuming.

Table 29: ADR 007

Date:	2025-10-7
Context:	Orchestrator, Model Services
Facing:	The need for a communication protocol between the orchestrator and model services
We decided:	To use GRPC
And neglected:	REST API, WebSocket, Message Queues
To achieve:	• Strongly typed contracts • Streaming requests • Low latency
Accepting that:	Adding another communication protocol adds complexity.

Table 30: ADR 008

Date:	2025-10-18
Context:	Model Services
Facing:	The need for a base image for model containers
We decided:	To the PyTorch base image
And neglected:	NVIDIA PyTorch base image
To achieve:	• A lower image size, roughly 15 GB compared to about 40 GB for the NVIDIA image
Accepting that:	• Not all CUDA tooling is included in the image

Table 31: ADR 009

Date:	2025-10-18
Context:	VAD Service
Facing:	The need for a VAD solution
We decided:	To use Silero VAD
And neglected:	WebRTC VAD
To achieve:	• Splitting the input audio into manageable segments • Configuring sensitivity for speech detection
Accepting that:	• Another model running on PyTorch needs to be integrated

Table 32: ADR 010

Date:	2025-10-18
Context:	VAD Service
Facing:	The need a plan on how to integrate Silero VAD
We decided:	To create a separate VAD Service hosting the VAD
And neglected:	To implement the VAD inside the orchestrator
To achieve:	• Keeping the orchestrator free from PyTorch dependencies
Accepting that:	• Another separate service is running its container, adding complexity

Table 33: ADR 011

Date:	2025-11-14
Context:	Orchestrator
Facing:	The need for a pattern for building a model pipeline
We decided:	To use the Chain of Responsibility pattern with a Pipeline Orchestrator that builds and manages the chain of handlers dynamically based on settings
And neglected:	Other patterns like a monolithic pipeline class or event-driven callback-based orchestration.
To achieve:	• Decouple individual pipeline steps (VAD, ASR, MT, TTS) into independent handlers with a common base class. • Allow dynamic construction and reconfiguration of the handler chain at runtime. • Make the pipeline extensible to add future handlers without modifying existing ones.
Accepting that:	• Handlers must conform to a common base class. • The orchestrator is responsible for task lifecycle management, adding complexity to the manager.

Table 34: ADR 012

Date:	2025-11-14
Context:	Orchestrator, Model Services
Facing:	The need for a way to define the capabilities
We decided:	To define capabilities central in the orchestrator using a static JSON.
And neglected:	Another approach, such as defining capabilities in each model service and querying the capabilities at runtime.
To achieve:	• That the orchestrator can check capabilities to figure out dynamically what models exist and what their capabilities are. • Having a central configuration that can be extended if new model services are added. • The orchestrator code does not have to be changed if a new model service is added
Accepting that:	• There is a static file that needs to be manually updated if the available model services change.

Table 35: ADR 013

Date:	2025-11-24
Context:	TTS Services
Facing:	The need for a solution that ensures memory is released after unloading a model
We decided:	To separate the TTS models into a subprocess
And neglected:	A solution where entire Docker containers could be started and stopped by the orchestrator.
To achieve:	• That it is possible to cancel the subprocess running the model and therefore release the memory
Accepting that:	• Additional complexity is added as the TTS GRPC service has to communicate with the subprocess using queues

Table 36: ADR 014

Date:	2025-12-06
Context:	Frontend, Audio feedback loop
Facing:	The need for a solution that ensures that a user can record speech without artificial pauses
We decided:	To remove the audio echo suppression
And neglected:	Keeping the echo suppression
To achieve:	• That it is possible to speak into the microphone without having to wait for a response after each sentence for a qualitative transcription
Accepting that:	• Audio that is played by the speaker may be picked up by the microphone • The likely need for a user to record using headphones

Table 37: ADR 015

7 Working Environment

7.1 Definition of Ready

For a user story to be allowed to join a backlog, we have to establish a Definition of Ready, which follows the well-established **INVEST** criteria:

- **I** ndependent: The story is independent of other user stories.
- **N** egotiable: The user story has been actively discussed among the team.
- **V** aluable: As this project is in the scope of a module, this is somewhat of a moot point. To accommodate this fact, we will instead count a story as valuable if its demonstrable.
- **E** stimable: The effort of the story is estimable, so it can be planned in a sprint.
- **S** mall: (Ties in with E) The scope of a story should at max be 1 sprint iteration.
- **T** estable: The story has to be defined clearly enough to make testing possible.

7.2 Definition of Done

Code

- Code has been reviewed
- Code has been merged into the main branch
- Code follows the Coding Guidelines 7.3.2

Testing

- New code is covered by unit tests
- Smoke testing has been performed in the development Environment
- All tests are passing

Documentation

- Documentation is updated if needed
- Documentation follows the Documentation Guidelines 7.3.1

7.3 Guidelines

Another way to ensure quality is by setting various guidelines, to speed up workflows, enable cooperative coding and prevent mishaps from happening. These guidelines are only as strict as needed to prevent being too restrictive and make work only as complicated as it needs to be.

7.3.1 Documentation Guidelines

- Every team member is responsible to document their own work.
- Chapters are added with the following template:

```
=== Chapter Title
<chapter-title>

text
```

- Keep the wording concise and grammatically correct as well as orthographically correct.
- Tables and images all need a caption.
- Resources are to be saved in the /resources subfolder. Additional subfolders may be added inside.
- Sources have to be credited via the bibliography.

- AI chat tools may be used for convenience, but not for generating content.
- The basic pattern provided by the template is to be applied. New files may be created if such a need arises.

7.3.2 Coding Guidelines

Since the OST does not provide any coding guidelines, we have to define them ourselves. We align ourselves with the industrial standards, especially the standards provided by the respective languages. We will make use of tools such as linters to enforce proper syntax and IDEs for simpler coding. The consequent use of all available products and guidelines ensures a pragmatic application of **clean code** [25].

- Ensure that the appropriate **linter** (depending on the current IDE) is applied before commits.
- All Functions bar getters and setters, are to be explained with their language-specific syntax. The input, output and the logic of the function specifically.
- **Comments** are to be added sparsely and only if necessary.
- Keep the **span of functions** only as long as they need to be; however, also not so small that functions are too split up and readability suffers.
- Whenever possible, code should be created **collaboratively**, although not to a point where it becomes constrictive. Every member has to decide for themselves at what level they're comfortable with.

7.3.3 Version Control Guidelines

All these rules apply to any kind of version control.

- Use feature branches following the scope of the given task.
- Feature branches at completion will be merged into the main branch after a successful merge request.
- Basic version controlling rules apply as defined during the SEP1 module (keeping commits short, frequently fetch and push, etc.).
- Make use of our merge request template.
- Follow the process for merge requests.
- Keep commits so short that they can be described in one simple sentence.
- Add tags corresponding to review submissions

Merge Requests

For the sake of clarity, the process for merge requests has to be defined. A merge request happens when a feature branch is to be merged into the main branch.

- A merge request requires an assigned reviewer
- The merge has to pass the pipeline
- Commits get squashed
- The reviewer either approves or disapproves of the merge
- After approval, the feature branch is merged into the main branch

Merge Request Template

```
### Description
This merge request addresses and describes the problem or user story being
addressed.

### Changes Made
Provide code snippets or screenshots as needed.

### Related Issues
```

```
Provide links to the related issues or feature requests.

### Additional Notes
Include any extra information or considerations for reviewers, such as impacted
areas of the codebase.

### Merge Request Checklist
- [ ] Code follows project coding guidelines.
- [ ] Documentation reflects the changes made.
- [ ] I have already covered the unit testing.
```

8 Test Concept

8.1 Introduction

The test concept defines the test strategy for our project, covering different test levels and their execution. The goal is to ensure the system meets both functional and non-functional requirements while maintaining a high level of quality and usability.

8.2 Test Scope

The test concept covers:

- **Unit Testing:** Ensuring that code units work as expected.
- **Component Testing:** Ensuring that UI components render and behave correctly.
- **Acceptance Testing:** Ensuring requirements are met.
- **Usability Testing:** Evaluating user experience and ease of use.
- **Model Testing:** Evaluating the latency and quality of the models.

8.3 Test Execution Time Plan

Automated Tests	
Unit Tests	On each pushed commit, merge request and deployment
Component Tests	On each pushed commit, merge request and deployment
Automated Model Tests	During Iteration 5
Manual Tests	
Manual Model Tests	End of Iteration 6
Usability Tests	End of Iteration 6
Acceptance Tests	End of Iteration 6

Table 38: table of when what tests get executed

8.4 Automated Testing

Test Type	Backend Unit Tests
Test Framework	Pytest
Execution	On each pushed commit, merge request
Purpose	Unit tests ensure the correctness of business logic in the scope of functions.

Table 39: Backend Unit Tests

Test Type	Frontend Unit Tests
Test Framework	Vitest
Execution	On each pushed commit, merge request
Purpose	Unit tests ensure the correctness of business logic in the scope of functions.

Table 40: Frontend Unit Tests

Test Type	Frontend Component Tests
Test Framework	Vitest
Execution	On each pushed commit, merge request
Purpose	Component testing in frontend ensures that individual UI components function correctly.

Table 41: Frontend Component Tests

8.5 WER Testing

Word Error Rate (WER) is a standard metric for evaluating the performance of ASR systems. It quantifies the distance between the ASR output and the reference transcription, with lower values indicating higher transcription accuracy. It is widely used in speech recognition research because it captures all types of transcription errors at the word level.

Formally, it is defined as:

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

Where:

- `S` = number of **substitutions** (words incorrectly replaced)
- `D` = number of **deletions** (words missing in the hypothesis)
- `I` = number of **insertions** (extra words added)
- `N` = total number of words in the reference transcript

Test Type	WER Test
Test Framework	python script with jiwer [26]
Execution	Locally after changes to ASR model implementation
Purpose	The WER test measures the word error rate to verify the NFR3.

Table 42: WER Test

8.5.1 Dataset

We evaluate the ASR models using four distinct speech datasets, selected to capture a range of languages, recording conditions, and difficulty levels. In total, we used 200 audio samples of speech (50 per dataset), with audio durations between roughly 3 and 12 seconds:

ATCO2

English air-traffic communications (ATC), consisting of pilot and controller speech with real background noise and non-ideal conditions. Short utterances typical of ATC radio traffic make automatic transcription challenging. Dataset details and documentation can be found via the ATCO2 project, which collects and preprocesses annotated ATC speech for ASR research. [27]

We use this dataset because it challenges our system, allowing us to evaluate its performance at voice impersonation in the particularly difficult context of noisy pilot-controller communications.

ASR German Mixed

A composite German speech dataset created by merging multiple open-source corpora, including Common Voice and Multilingual LibriSpeech. It contains natural German speech with everyday vocabulary and mild real-world noise characteristic of mixed-source audio. [28]

LibriSpeech Clean

A partition of the well-known LibriSpeech corpus comprised of high-quality read English speech from audiobooks, selected for clear enunciation and low noise. [29]

LibriSpeech Other

The counterpart to the clean subset. This LibriSpeech partition contains more challenging segments characterised by slight speaker variability, accents, and less predictable articulation, making automatic transcription more difficult. [29]

8.5.2 Normalisation and Preprocessing

Before computing WER, both model outputs and references were normalised to reduce differences unrelated to semantic transcription quality:

- **Punctuation removal:** All punctuation characters were removed from both hypothesis and reference transcripts to avoid penalising minor punctuation differences.
- **Number normalisation:** Numeric expressions were normalised (e.g., “one two” → “12”), reflecting the behaviour of Whisper models, which often transcribe spoken digits as concatenated numerals rather than as separate words.
- Other tokens were lowercased and whitespace trimmed to ensure the error metric focuses on word content rather than formatting.

These normalisation steps refer to and predict texts more similar by reducing differences that are not real recognition errors. This gives a clearer and easier-to-interpret WER score.

8.6 BLEU Testing

BLEU (Bilingual Evaluation Understudy) measures the similarity between a machine-generated translation and one or more human reference translations by comparing overlapping n-grams (sequences of words). Higher scores indicate translations closer to human reference translations, with a score of 100 being a perfect match.

Test Type	BLEU Test
Test Framework	python script with sacreBLEU [30]
Execution	Locally after changes to MT model implementation
Purpose	The BLEU test measures translation quality to verify the NFR4.

Table 43: BLEU Test

sacreBLEU is a standardised implementation of the BLEU metric for machine translation evaluation and ensures reproducibility and consistency in BLEU evaluation by:

- Standardizing tokenization and normalisation of text across languages.
- Providing a fixed versioning system for BLEU computations to prevent discrepancies across different BLEU implementations.
- Supporting evaluation against multiple reference translations

8.6.1 Dataset

- **Tatoeba** is a large, community-driven collection of example sentences and their translations, covering hundreds of languages and language pairs. It is designed to support multilingual research by providing short, parallel sentence pairs that link sentences in one language to their translations. The sentences are contributed and reviewed by volunteers. [31]

  From this dataset, we have used a small subset of 100 sentence pairs per language pair (e.g., English–German, French–Italian, …) to test our models.

8.7 Cosine Similarity Testing

The cosine similarity results indicate how closely the synthesised voices match the reference speaker embeddings. Higher values suggest that the model is better at capturing speaker-specific characteristics, while lower values suggest more variation or drift from the target voice.

Test Type	SIM Test
Test Framework	python script with ECAPA-TDNN [32]
Execution	Locally after changes to TTS model implementation
Purpose	The SIM test measures speaker similarity to verify the NFR5.

Table 44: SIM Test

8.7.1 Dataset

We compiled a custom evaluation dataset containing 10 different speakers and sentences. We then ran these samples on 10 different sentences per sample to calculate the cosine similarity.

(10 speakers × 10 generated samples per model)

8.8 Intelligibility Testing

Intelligibility testing evaluates how well a TTS system preserves the content of an original utterance. The procedure uses an ASR in a closed-loop:

1. Original Speech ⟶ ASR: The ASR model *large-v3-turbo* transcribes the original audio into text.
2. TTS ⟶ Speech Synthesis: The text is fed into the TTS system to generate synthetic audio.
3. Synthetic Speech ⟶ ASR: The synthetic audio is transcribed back into text using the same ASR system.
4. Comparison: The two ASR outputs (original and TTS-generated) are then compared using WER.

This approach provides a quantitative measure of TTS intelligibility without requiring human listeners. Lower WER indicates that the TTS system produces speech that is similarly intelligible to the ASR as the original human-spoken audio.

Test Type	Intelligibility Test
Test Framework	python script with jiwer [26]
Execution	Locally after changes to TTS model implementation
Purpose	The Intelligibility test measures how understandable the TTS output is compared to the original audio without having to rely on human testing.

Table 45: SIM Test

8.8.1 Dataset

We used the same datasets as in the WER tests:

- ASR German Mixed: 50 German utterances with everyday vocabulary and mild background noise.
- LibriSpeech English Clean: 50 English utterances, read speech with low noise.

8.9 Mean Opinion Score (MOS) Testing

The Mean Opinion Score (MOS) test is a subjective evaluation method used to measure the perceived quality of generated speech. In this project, the test is applied to assess whether the system produces natural and intelligible speech that closely resembles human pronunciation and prosody. Evaluators listen to audio samples produced by the model and assign a score based on their perception of overall quality.

8.9.1 Rating Scale

Each audio sample is rated on a five-point scale. The higher the score, the more natural and human-like the output is perceived to be:

Score	Description
5	Excellent - completely natural and easy to understand
4	Good - mostly natural with minor artefacts
3	Fair - intelligible but with noticeable synthetic elements
2	Poor - difficult to understand or unnatural
1	Bad - severely distorted or unintelligible

Table 46: MOS Rating Scale

8.9.2 Evaluators

A group of 5-10 evaluators with fluent English and German listening skills participates in the test. Evaluators are instructed to use headphones in a quiet environment and are familiarised with the scoring scale before rating.

Due to the unavailability of suitable evaluator groups, Italian and French will not be included in the MOS evaluation. The test will therefore focus on the remaining supported languages for which reliable participant groups are available.

8.9.3 Dataset

- English Samples: 20 recordings (3–6 seconds) from Librispeech ASR [29]
- German Samples: 20 recordings (3–6 seconds) from ASR-GERMAN-MIXED-TEST [28]

8.9.4 Test Setup

Each evaluator is presented with a randomised set of 20 audio clips produced by the system. Evaluators listen to each sample once or twice and assign a rating independently.

To establish a ground truth, 5 real recordings were mixed in among the generated samples.

The MOS for each sample is computed as the arithmetic mean of all assigned scores:

$$\text{MOS} = \frac{\text{sum of all ratings}}{\text{number of evaluators}}$$

The overall MOS of the system is obtained by averaging across all test samples.

8.10 Subjective Speaker Similarity Testing

As speaker similarity is subjective and difficult to capture reliably with automatic metrics such as the performed cosine similarity test, a human evaluation was conducted to provide a more accurate assessment. In this test, evaluators listened to pairs of audio samples and rated how similar the speakers sounded.

8.10.1 Rating Scale

Each trial was rated using a four-point scale, which allows evaluators to indicate confidence while keeping the task simple. For analysis, the responses are collapsed into binary decisions (“Same” vs. “Different”) as is standard practice.

Score	Description
Same Speaker	Definitely the same voice
Same Speaker	Probably the same voice
Different Speaker	Probably a different voice
Different Speaker	Definitely a different voice

Table 47: Similarity Rating Scale

8.10.2 Evaluators

The same 5-10 evaluators as with the MOS testing scored the Speaker Similarity testing.

8.10.3 Dataset

No samples in the MOS Testing were used here.

- English Samples: 20 recordings (3–6 seconds) from Librispeech ASR [29]
- German Samples: 20 recordings (3–6 seconds) from ASR-GERMAN-MIXED-TEST [28]

8.10.4 Test Setup

Each trial consisted of two audio clips: a reference recording and a sample, which could be either a sample generated by the system or a real recording.

To help evaluators anchor their judgments, each session included control trials:

- 3 trials where the reference was followed by a real recording from the same speaker (positive controls)
- 3 trials where the reference was followed by a real recording from a different speaker (negative controls)

All trials were randomised and blinded, so evaluators did not know whether a sample was synthesised or real. Evaluators were allowed to replay clips as needed before assigning their scores.

Similarity is calculated as the ratio of trials with synthesised samples rated as “Same speaker”:

$$\text{Similarity} = \frac{\text{Number of all trials rated positively}}{\text{number of trials}}$$

8.11 E2E Latency Testing

For testing the E2E latency, the system was run as in production, and 8-minute segments of public domain audiobooks were played as real-time input to the system. At the end, the total avg and the avg of each step in the pipeline, which gets calculated by the application was collected. The following configurations were tested. The latency testing was performed on a system with an RTX 4080 Super GPU.

Test	Audio Book	ASR Model	MT Model	TTS Model
German	Gelnhausen, Franz Kafka [33]	Faster Whisper (medium)	None	F5-TTS
English	The Gift of the Magi, O.Henry [34]	Faster Whisper (medium.en)	None	VoxCPM
English to German	The Gift of the Magi, O.Henry	Faster Whisper (medium.en)	MarianMT (opus-mt-en-de)	F5-TTS

Table 48: E2E Latency Test Configuration

8.12 Acceptance Tests

With acceptance tests, we aim to evaluate whether the software meets the agreed upon requirements and satisfies stakeholder expectations. This ensures that the software is ready to be accepted and handed over.

Execution:
- These tests will be executed once the web application has reached a stable state suitable for proper testing. This is planned to occur by Milestone M3 at the end of iteration 6.
- The tests will be performed manually by our industry partners.

Acceptance Criteria:
- All FRs must be fulfilled.

8.13 Usability Testing

With usability testing, we aimed to evaluate how intuitively users can operate the application.

The objective of the test was to observe how well a first-time user understands the workflow without prior instructions.

Execution:
- The test was performed by one external participant.

Usability Criteria:
- Users understand the required steps for core functionality.
- Model and language selection are intuitive.
- System feedback is clear and trustworthy.

Aspect	Description
Objective	Evaluate intuitive use of real-time speech translation with voice preservation.
Participants	1 external user.
Execution Type	Manual test with live observation.
Test Location	Local test session with developer observation.
Observation	User actions, comments, and hesitation points were documented.

Table 49: Usability Test Plan

8.13.1 Test Scenarios

1. Record a voice cloning file so that the generated audio will sound like your own voice.
2. Update the settings and models to allow German-to-English translation.
3. Start a live recording to generate audio.

8.13.2 Usability Test Questionnaire

The participant was asked open-ended questions after completing each task.

Nr.	Question
1	What did you expect to happen on this screen?
2	Was anything unclear or confusing during this task?
3	What would you change to make this easier to understand?
4	Did you feel the application was responding as expected?
5	Was there any moment where you were unsure how to continue?

Table 50: Usability Test Questionnaire

8.14 Browser Testing

To ensure that the application is working across the most used browsers, the application is smoke tested on the different browsers after each iteration passed iteration 4.

9 Implementation Details

Chapter 6 covers, for the most part, how the application works and is implemented. Further details, including code documentation, can be found in the Markdown files in the repository.

Still, a select few notable implementation details are documented here.

9.1 Audio stream segmentation

The selected ASR, MT, and TTS models do not support streaming input. Consequently, an explicit segmentation strategy is required to split the incoming audio stream into segments that can be processed by the individual models. At the same time, these segments must contain sufficient contextual information to allow the MT and TTS models to generate acceptable output.

To achieve this, Silero Voice Activity Detection (VAD) was integrated into the system. Silero VAD operates on fixed-size audio chunks of 32 ms (512 samples at 16 kHz) and outputs a probability indicating whether speech is present in a given chunk.

Once speech activity is detected, the orchestrator begins buffering incoming audio chunks. Buffering continues until the VAD determines that the speech segment has ended. In the implemented logic, a speech segment is considered complete when a chunk classified as speech is followed by seven consecutive chunks without detected speech, corresponding to a pause of approximately 224 ms. At this point, the buffered audio is finalised and forwarded as a single segment through the ASR, MT, and TTS pipeline.

This segmentation approach provides a pragmatic balance between latency and contextual completeness. Human speakers naturally introduce short pauses between sentences or clauses, which are effectively captured by the VAD-based segmentation. Even when segmentation occurs within a sentence, it typically aligns with sub-sentence boundaries, preserving sufficient context for downstream models.

9.2 Pipeline

The main part of the orchestrator is implemented using an adaptation of the chain of responsibility pattern. The pipeline manager is responsible for the chain and handles its creation and changes to it. Each step of the pipeline, ASR, MT, TTS and output implements its own handler and passes its output to the next handler. One change made from the original pattern is that there is a need for the output to fan out to multiple handlers. As seen in the Figure 19, the output from the ASR needs to be forwarded to both the MT handler and the output handler.

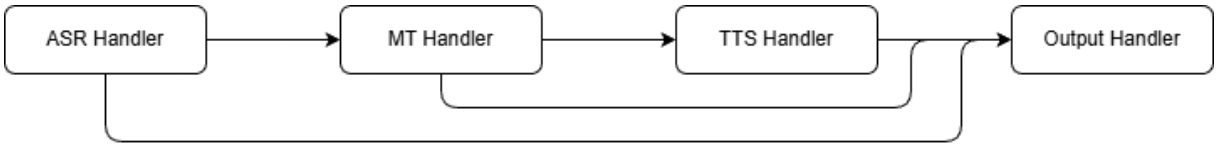


Figure 19: Chain of Responsibility with MT

The pattern also allows the pipeline manager to dynamically change the order of the pipeline. For example, if the target language is equal to the source language, the pipeline manager can rebuild the chain and exclude the MT handler.

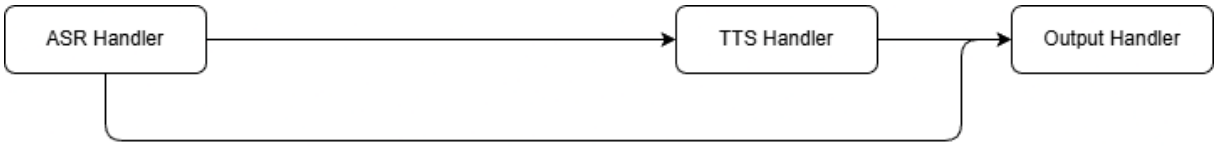


Figure 20: Chain of Responsibility without MT

Additionally, the modularity of the pattern also allowed the scripts for the model testing to build their own chains. For example, it was possible for the intelligibility testing to build a chain where the TTS handler is followed by the ASR handler to measure the WER of the TTS.

9.3 Capabilities and new models

New Models need to implement their corresponding protocol buffers and have to be added to the Docker Compose so they are hosted in the same network.

To allow the orchestrator to work with new models dynamically, the capabilities of new models need to be defined in the `config.json` as in the example below.

```
"asr": [
        {
            "name": "faster-whisper",
            "host": "asr-whisper:50051",
            "display_name": "Faster Whisper",
            "checkpoints": [
                {
                    "name": "medium",
                    "languages": ["en", "de", "fr", "it"]
                }
            ]
        }
    ]
```

Code Fragment 1: Model config entry example

A model needs a host so the orchestrator can reroute its GRPC client when the model is changed. The display name is shown in the frontend. ASR and MT models can currently define multiple checkpoints, which can be swapped at runtime as long as the change checkpoint endpoint is implemented. The capabilities are also used to filter models in the frontend depending on the languages chosen and the languages the available models support.

9.4 ASR Output Hallucinations

Our automatic speech recognition (ASR) model, Whisper, sometimes produces nonsensical or irrelevant text, often referred to as hallucinations. Examples include:

- `Vielen Dank fürs Zuschauen!`
- `Untertitel im Auftrag des ZDF, 2020.`

These hallucinations suggest that the model was trained heavily on certain types of content, including publicly available broadcasts from sources like ZDF. Essentially, the model is drawing on patterns it has seen frequently during training.

Occurrence typically happens under the following conditions:

- Background conversations or overlapping speech.
- Low-quality or noisy audio.
- Very short or transient audio fragments that do not contain intelligible speech.

These hallucinations are problematic because they reduce the accuracy and, with it, the reliability and trustworthiness of the transcription.

9.4.1 Fixes implemented

To reduce said hallucinations, we implemented the following fixes:

1. Minimum Speech Segment Length with VAD: We modified the Voice Activity Detection (VAD) to only forward segments that are at least 300 ms long. Short bursts of noise are now ignored, preventing the ASR from attempting to transcribe meaningless audio.

2. Less Sensitive VAD: The VAD was previously too sensitive to minor sounds, often classifying brief noises or non-speech as valid speech. By lowering the sensitivity, fewer non-speech fragments reach the ASR, which directly reduces hallucinations.

3. ASR Call Modifications: We adjusted some parameters that we give to the ASR on every call that differ from the default values. These include a stricter detection of non-speech, detection of repetitive text, and suppression of non-speech tokens.

```
self.model.transcribe(
    data,
    language=lang,
    no_speech_threshold=0.6,
    compression_ratio_threshold=2.4,
    suppress_tokens=[-1],
    suppress_blank=True,
)
```

Code Fragment 2: ASR Whisper Transcription Call

9.4.2 Outcome

While these hallucinations can still occur, especially in noisy multi-speaker environments, after these changes, we were able to observe a reduction in hallucinations. The ASR now produces more accurate transcripts and avoids generating irrelevant phrases from short or low-quality audio segments.

For the best transcription quality and to minimise background noise, it is advised to use headphones when recording.

10 Results

10.1 Functional Requirements

10.1.1 User Acceptance Test Results

FR ID	Description	Remarks	OK / NOK
FR01	I want to record myself so that I can use my speech as source input		**OK**
FR02	I want to see a transcription of what I say in real time, so I have visual feedback		**OK**
FR03	I want to upload a voice reference so that I can use that as the reference input		**OK**
FR04	I want to hear the output speech in real time, so that I can sound like the target speaker		**OK**
FR05	I want to choose the output language between English, German, French and Italian, so that the system can translate my speech into that language.		**OK**
FR06	I want to see the translated transcription in real time, so I have visual feedback		**OK**
FR07	I want to switch between different ASR, MT and TTS models, so that I can optimise the pipeline for my requirements	One MarianMT checkpoint did not work during the test, but the issue was fixed afterwards	**OK**
FR08	I want to observe the relevant stats of the process, such as latency and real-time factor so I have information about the performance of the output.		**OK**

Table 51: Functional Requirement Test Protocol

10.1.2 Automated Test Coverage and Success Rate

Test Type	Number of Tests	Success Rate	Code Coverage
Backend Unit Tests	49	100%	90%
Frontend Component / Unit Tests	98	100%	82%

Table 52: Automated unit and component test results

10.2 Non Functional Requirements

The non-functional requirements are described in Chapter 4.

10.2.1 NFR-01: End-to-End Latency

ID	**NFR-01** End to End Latency
Category	Performance
Priority	High
Status	**Reached minimal requirement of maximum 5s latency**
Reason	Essential for usability in real-time communication.

Table 53: Validierung NFR-01 Performance

10.2.1.1 Results

The following table shows the results of the three tests. The system latency is the entire inference latency of all steps, but without the speech duration. The total latency measures the time between the first utterance of the input and the first utterance of the output. All measurements are in milliseconds.

Test	English	German	English to German
Speech Duration	2385	2761	2170
ASR Latency	331	373	324
MT Latency	N/A	N/A	196
TTS Latency	1133	1326	1167
System Latency	**1464**	**1699**	**1687**
Total Latency	3850	4460	3857

Table 54: E2E Latency Test Results

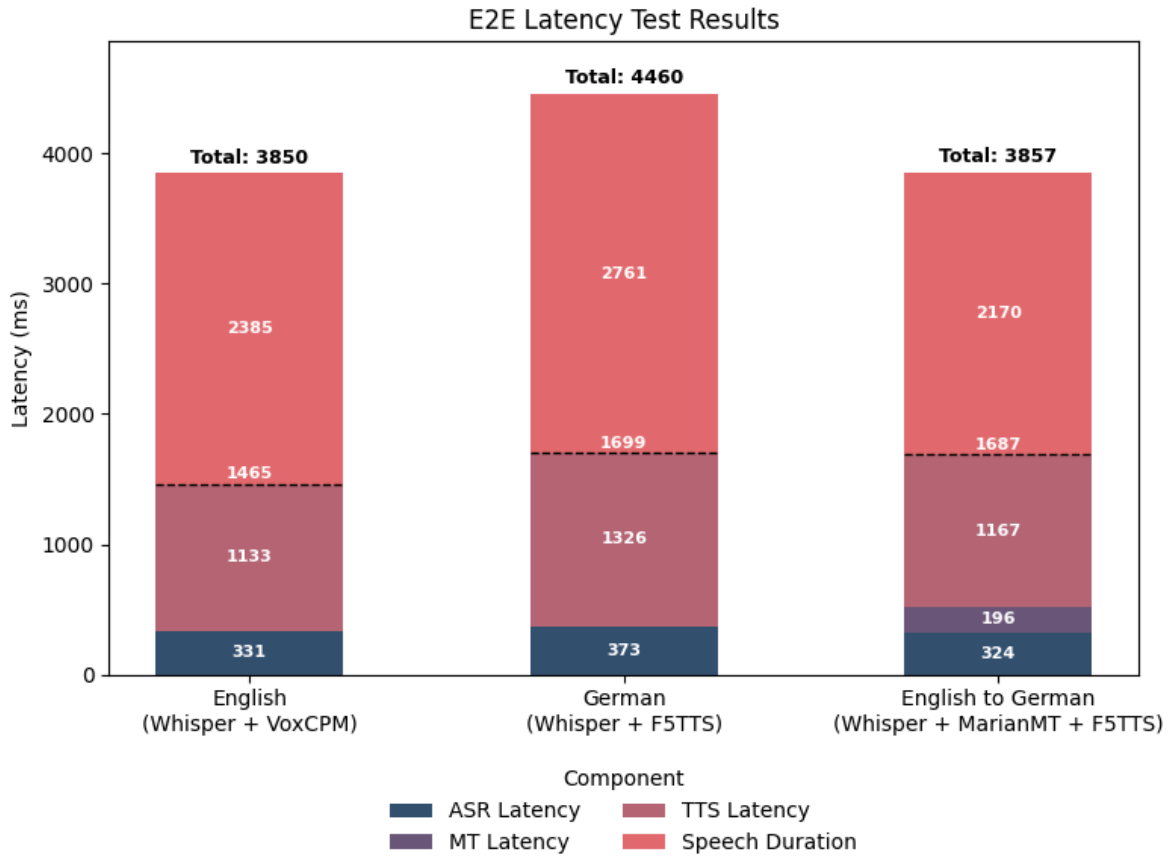


Figure 21: E2E Latency Test Results

10.2.1.2 Conclusion

The end-to-end latency measurements show that the average total latency remains in the range of 3.8 to 4.4 seconds, which is lower than the minimum latency requirement of 5 seconds. In cases with shorter speech length, it reaches the target latency of 4 seconds.

The largest contributor to total latency is the speech duration itself, which cannot be reduced without changing the segmentation strategy or the capabilities of the underlying models. Early or overly aggressive segmentation would increase the likelihood of cutting audio mid-word or mid-phrase, however, this also deprives the MT and TTS models of the necessary linguistic context. As a result, speech duration represents an inherent lower bound on achievable end-to-end latency in the current architecture.

Excluding speech duration, the measured average system latency remains consistently in the range of approximately 1.4 to 1.7 seconds across all test cases. Among the inference steps, TTS clearly dominates system latency, accounting for the largest share of processing time. This behaviour is expected, as neural speech synthesis is computationally more expensive than ASR and MT.

Further reductions in system latency would primarily require hardware improvements, specifically more powerful GPUs, or optimisations to the model implementations themselves. While output streaming for the VoxCPM model could significantly reduce latency, experiments showed that enabling streaming currently results in an unstable output, prone to hallucination, making it unsuitable for use within the scope of this thesis.

10.2.2 NFR-02: Transcription Accuracy

ID	**NFR-02** Transcription Accuracy
Category	Functionality
Priority	High
Status	**Reached target requirement of 90% accuracy for average audio**
Reason	High accuracy is crucial to avoid error propagation to translation.

Table 55: Validierung NFR-02 Functionality

10.2.2.1 Results

Model	ATCO2	ASR German Mixed	LibriSpeech Clean	LibriSpeech Other
large-v3-turbo	40.68%	5.50%	3.01%	7.29%
medium	44.01%	7.13%	3.94%	8.25%
medium.en	41.74%	N/A	3.21%	8.03%

Table 56: WER Test Results

- Clean English audio shows very low WER (∼3%), confirming reliable transcription.
- Noisy conditions (ATCO2) increase WER (∼40%), which is expected, especially since it contains a lot of words / phrases that are not part of day-to-day speech
- **large-v3-turbo** performs best overall, validating NFR3 compliance.

10.2.2.2 Limitations of WER

While WER provides a useful quantitative measure of transcription performance, it has several limitations that have to be kept in mind:

- **Semantic context is ignored:** WER only counts word-level insertions, deletions, and substitutions, without considering whether the overall meaning of the utterance is preserved. For instance, a sentence with one substituted word may still convey the correct message, while WER treats it as an error.
- **Error types are not differentiated by severity:** WER treats all word errors equally. Missing a critical command word in ATC speech has a far greater impact than a minor filler word, but WER cannot reflect this.
- **Fluency and readability are not captured:** WER ignores whether the transcription is grammatically correct or natural-sounding.

10.2.2.3 Qualitative Error Analysis

The qualitative error analysis complements the WER.

The analysis focuses on identifying common patterns in recognition errors, understanding their sources, and highlighting areas for potential improvement.

For that, we analysed the output of the models in the WER testing and grouped the errors into the following qualitative categories:

1. Proper Nouns

Proper nouns, including personal names and place names, represent a significant source of error. Examples include:

- `graeme` ⟶ `graham`
- `azariah` ⟶ `azaraya`
- `elexander blodggett` ⟶ `alexander blodggett`
- `saint ouen` ⟶ `st. louis`

Impact: While a reader might be able to reconstruct the error, they are still particularly detrimental for downstream tasks that require named entity recognition or information extraction.

2. Similar-Sounding Words

Several errors arose from words with similar pronunciation:
- `lock` ⟶ `log`
- `moor` ⟶ `more`
- `gram` ⟶ `graham`

Impact: These substitutions often preserve grammatical structure but alter meaning entirely, potentially making it confusing or misleading for users.

3. Rare or Archaic Words

The system struggles with less common vocabulary:
- `ensnares` ⟶ `snazz`
- `clomb` ⟶ `climb`

Impact: Misrecognition of rare words occurs in complex literary or historical texts, limiting the usefulness of the ASR system for these domains.

4. Callsigns and Phonetic Vocabulary

A large portion of ASR errors in the ATCO2 dataset involves callsigns (e.g., `Oscar Kilo`, `Lufthansa nine tango papa`), where the model either misrecognizes phonetic code words or produces unrelated text:
- `Oscar Kilo Bravo Alfa Lima…` ⟶ `BRL after book up…`
- `Lufthansa 9 tango papa…` ⟶ `Stanza 9HT Papa…`

Impact: Misunderstandings like these can make a message completely incomprehensible and reduce system reliability for safety-critical or time-sensitive tasks.

5. Background Noise

Errors are sometimes caused by the acoustic environment, including background noise, channel distortion, or overlapping speech. This happened especially frequently in the ATCO2 dataset:
- `oscar kilo foxtrot alfa oscar taxi to holding point runway` ⟶ `of Karkov. So... Got it taxi holding on tramway`
- `ruzyne tower ups 2 9 5 ils` ⟶ `Raising tower UPS 2 niner 5 ILS`

Impact: Misrecognitions due to background noise reduce reliability in real-time communication scenarios and cause errors similar to those observed in the other categories.

10.2.2.4 Recommendations for Improving ASR on ATC / Pilot Radio Speech

Based on the observed error patterns in proper nouns, callsigns, rare words, and numeric sequences, there are two main approaches to improve recognition performance for aerial traffic control (ATC) and pilot radio speech:

1. Fine-Tuning Pretrained ASR Models

Fine-tuning a large, pretrained model such as Whisper on domain-specific ATC corpora can improve recognition of phonetic alphabet and callsigns. Studies have shown that even a few hours of annotated ATC speech can significantly reduce word error rate (WER) in specialised domains [35]. This approach leverages the general speech understanding of the base model while adapting it to pilot radio communication.

2. Classical Domain-Specific Models

For extremely low-resource cases or highly constrained vocabularies like ATC speech, classical ASR approaches or statistical language models trained specifically on ATC phraseology can also be effective. These models explicitly encode the limited vocabulary and grammar rules of ATC communication, which helps reduce errors on callsigns and numeric sequences. However, they generally lack the flexibility of large pretrained models for handling unexpected speech patterns [36].

Practical Recommendation: The best results can be achieved by starting with a pretrained model and fine-tuning it on ATC data. Classical models can then help with callsigns and numeric sequences. Studies show both model types are useful, and some ATC systems combine them for better performance [36].

10.2.2.5 Conclusion

The ASR system performs extremely well on common vocabulary. Nearly half of the audio files in the combined dataset achieved 0% WER after normalisation.

Proper nouns, rare or archaic words, and words that have similar pronunciation are frequent error sources. Errors tend to cluster in long or complex sentences with multiple uncommon words.

Downstream Implications: While the system is adequate for general transcription tasks, caution is required for tasks involving named entities, historical texts, or domain-specific numbers and terminology. Without targeted fine-tuning, these errors can make a general ASR model unusable in certain fields.

10.2.3 NFR-03: Translation Quality

ID	**NFR-03** Translation Quality
Category	Functionality
Priority	Medium
Status	**Exceeded outstanding requirement of a BLEU score above 35**
Reason	Ensures semantic correctness across languages.

Table 57: Validierung NFR-03 Functionality

10.2.3.1 Results

Source Language	**MarianMT BLEU**	**M2M100 BLEU**
DE	53.15	40.75
EN	56.01	45.30
FR	55.10	43.23
IT	57.76	43.35
Overall Average	**55.50**	**43.16**

Table 58: BLEU Test Results per source language

Target Language	MarianMT BLEU	M2M100 BLEU
EN	60.13	42.14
FR	58.99	46.25
DE	51.79	40.57
IT	51.10	43.68
Overall Average	**55.50**	**43.16**

Table 59: BLEU Test Results per target language

10.2.3.2 Analysis by Source and Target Languages

A closer inspection of the BLEU results reveals that translation quality is not equally stable across individual language pairs, even though the scores per source language appear relatively consistent.

We can see MarianMT varying a lot between language pairs. For example, translations from German to English achieve a high BLEU score of 60.07, while German to Italian drops sharply to 42.26. Similarly, translations into English (e.g., IT–EN: 61.47, FR–EN: 58.86) consistently score higher than translations into German or Italian.

This pattern shows that the target language affects BLEU more than the source language. Pairs with English as the target usually get higher BLEU scores. This is likely because the model was trained more on English data, and its grammar is simpler than German or Italian.

Despite this variability at the language-pair level, no catastrophic performance drops occur. All MarianMT language pairs remain above 42 BLEU points, meaning there are no complete breakdowns in translation quality.

When results are aggregated per source language, these fluctuations partially cancel out. As a result, the per-source-language averages (53.15–57.76 BLEU) appear relatively stable and mask the underlying pairwise differences.

M2M100 follows a similar but consistently weaker pattern. Its BLEU scores vary across language pairs (approximately 38–48 BLEU), with higher scores again observed for translations into English and French.

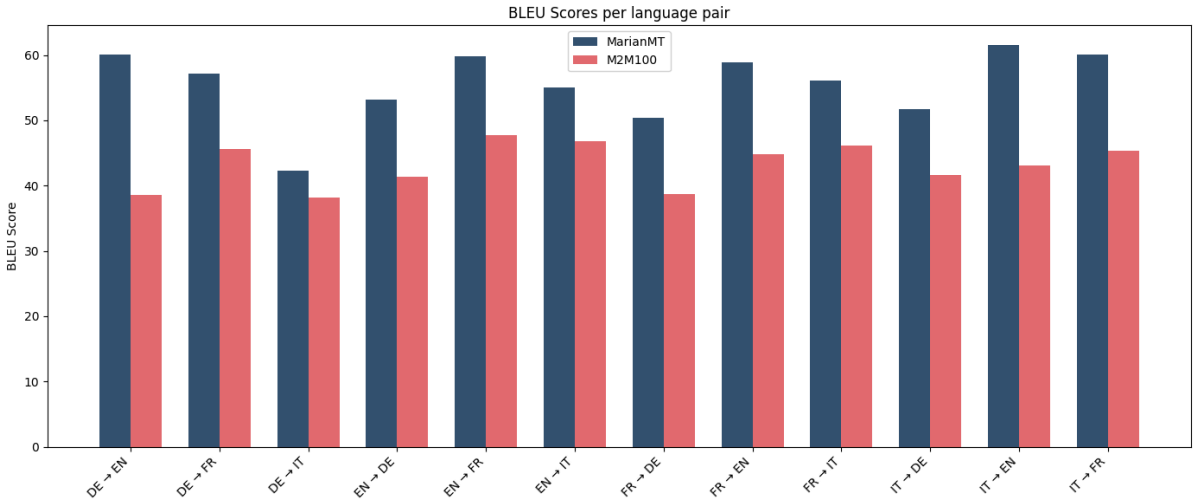


Figure 22: BLEU scores per language pair

10.2.3.3 Limitations of BLEU

While BLEU scores provide a useful quantitative measure of translation quality, they do not capture certain limitations inherent to current neural machine translation models. One key limitation is **context unawareness**: these models generally translate sentences in isolation and do not have the ability to consider the broader textual context. This can lead to ambiguity or errors in translations that rely on previous sentences.

Language	Source Text	Model Translation (MarianMT)

English → German	`"I walked along the bank of the river to clear my head."`
	
	`"The bank was slippery so I had to be careful."`
		`"Ich ging am Ufer des Flusses entlang, um meinen Kopf zu räumen."`
		`"Die Bank war rutschig, also musste ich vorsichtig sein."`

Table 60: Example showing context-sensitive ambiguity (English → German)

In this example, the word **"bank"** is ambiguous in English. The model correctly translates it as **"Ufer"** (riverbank) in the first sentence, but in the second sentence, which repeats **"bank"** without explicit context linking it to the river, it incorrectly translates it as **"Bank"** (financial institution). This demonstrates that the model does **not carry over context between sentences**, which can lead to inconsistencies in longer texts.

10.2.4 NFR-04: Natural-Sounding Speech

ID	**NFR-04** Natural-Sounding Speech
Category	Functionality
Priority	High
Status	**Reached target requirement of a MOS score above 4.0**
Reason	Acceptability of the output depends on voice naturalness.

Table 61: Validierung NFR-04 Functionality

10.2.5 Results

- Average mos score german = 4.28250 +- 0.30443
- Average mos score english = 4.47375 +- 0.37017

10.2.5.1 MOS Test Results

Model	Mean	Std[2]	Min	Max
F5TTS (German)	4.28250	0.30443	3.87	4.87
VoxCPM (English)	4.47375	0.37017	3.73	4.8

Table 62: Human Similarity Evaluation Results

10.2.5.2 Conclusion

The results of the MOS test show that the naturalness of the speech exceeds the target requirement of 4.0 out of 5.0 Points.

The relatively small standard deviations suggest a high level of agreement among evaluators, with naturalness ratings clustering closely around the mean for both models. While both systems perform

[2] standard deviation

well overall, the English model shows a marginally higher average score, indicating slightly more convincing naturalness in the generated speech.

As these results are based on a limited group of eight evaluators, the findings should be interpreted with caution. A larger and more linguistically diverse subject pool would be required to draw more definitive conclusions. Nevertheless, the MOS results provide strong evidence that both TTS models are capable of generating highly natural-sounding speech, meeting the qualitative expectations for the intended application.

10.2.6 NFR-05: Voice Preservation

ID	**NFR-05** Voice Preservation
Category	Functionality
Priority	High
Status	**Reached target similarity of 0.7 for German and outstanding similarity of 0.8 for English**
Reason	Speaker identity preservation is the main goal of the system.

Table 63: Validierung NFR-05 Functionality

10.2.6.1 Cosine Similarity Results

Model	**Mean**	**Std**[3]	**Min**	**Max**
F5TTS (German)	0.7183	0.0648	0.5700	0.8426
VoxCPM (English)	0.7039	0.0600	0.5396	0.8179
XTTSv2 (German)	0.6284	0.0965	0.2657	0.8173
XTTSv2 (English)	0.5788	0.0893	0.3862	0.7421

Table 64: Cosine Similarity Results

From the results:

- **F5TTS (German)** achieves the highest mean similarity ($\sim$0.72), with a relatively low standard deviation ($\sim$0.065), indicating both accurate and consistent speaker cloning.
- **VoxCPM (English)** performs similarly ($\sim$0.70) and shows slightly more consistent speaker similarity than F5TTS, as reflected by its lower standard deviation.
- **XTTSv2 models** show lower means ($\sim$0.63 for German, $\sim$0.58 for English) and higher standard deviations, pointing to less stable or less accurate speaker reproduction. The minimum values ($\sim$0.27 for XTTSv2 German) indicate occasional poor synthesis for certain sentences.

10.2.6.2 Limitations of Cosine Similarity

While useful for measuring embedding similarity, cosine similarity does **does not capture how the voice actually sounds to a human, including tone, rhythm, or overall naturalness**. Two voices can have high embedding similarity yet sound noticeably different to human listeners, especially in intonation, emotion, or timbre nuances. Conversely, lower similarity does not always imply poor perceived speaker match if embeddings are sensitive to certain acoustic variations.

That is why, to complement it, a speaker similarity evaluation with human evaluators was performed, to get a complete picture of the voice preservation quality.

[3] standard deviation

10.2.6.3 Human Similarity Evaluation Results

The average similarity ratio describes which ratio of pairs of real and generated audio was rated as "Probably the same speaker" or "Identical speaker".

Model	Mean	Std	Min	Max
F5TTS (German)	0.79500	0.18197	0.50	1.00
VoxCPM (English)	0.86750	0.08155	0.71	0.93

Table 65: Human Similarity Evaluation Results

10.2.6.4 Conclusion

Both the automated results and the results of the human evaluation show that the speaker similarity reaches the target requirement of 0.7 on average.

Notably, the German model shows substantial variability in evaluator ratings, with similarity scores ranging from 0.50 to 1.00. This indicates a high degree of subjectivity in perceived speaker similarity among evaluators. In contrast, the English model exhibits a narrower score range (0.71–0.93), suggesting more stable and consistent perceptions of speaker similarity.

Overall, the English model outperformed the German model both in terms of average similarity and rating stability. One possible explanation is that all evaluators were native German speakers and, therefore, more sensitive to subtle deviations in German speech characteristics, which may have led to greater variation in ratings for the German model.

As with the results of the MOS testing, although the results provide useful insight into the perceived quality of the voice cloning system, they are based on a relatively small sample size of eight evaluators. Still, the findings offer a reasonable indication of the overall quality of the voice preservation.

10.2.7 NFR-06: Intelligibility Testing

ID	**NFR-06** Intelligibility
Category	Functionality
Priority	High
Status	**The outstanding intelligibility requirement of 95% has been reached for both German and English**
Reason	The correctness of the synthesised speed has a significant impact on the quality.

Table 66: Validierung NFR-06 Functionality

10.2.7.1 Results

TTS Model	F5TTS	VoxCPM	XTTSv2
WER	2.96%	2.24%	6.19%

Table 67: Intelligibility Test Results

10.2.7.2 Qualitative Error Analysis

A closer look at high WER examples reveals patterns that are consistent with the previous qualitative analysis of ASR errors at Chapter 10.2.2.3:

1. Proper Nouns
Proper nouns remain a major source of error in TTS-generated speech, reflecting ASR sensitivity to unusual names:
- German: `wronski` ⟶ `ronski`, `reinekens` ⟶ `reineckens`
- English: `tuppany` ⟶ `toppenny`, `chingachgook` ⟶ `chinggis guk`

These errors tend to inflate WER, even if the rest of the utterance is perfectly intelligible.

2. Similar-Sounding Words
Mispronunciations or phonetic ambiguities in TTS can lead to substitutions that confuse the ASR:
- English: `were i` ⟶ `our eye`
- German: `die warf er eilend` ⟶ `die war vereilend`

These substitutions often preserve grammatical structure but can drastically increase measured WER.

3. Rare or Archaic Words
TTS struggles with less common vocabulary, especially in literary or historical texts:
- English: `adjour` ⟶ `adieu`, `draught` ⟶ `draft`
- German: `betuve linie` ⟶ `projektbetuflinie`

5. Minor Spelling Variants
English shows lower WER overall, but small variants (spelling, tense) still contribute:
- `grey` ⟶ `gray`, `seem` ⟶ `seemed`, `theatre` ⟶ `theater`

These are counted as WER errors even though intelligibility is preserved.

10.2.7.3 Conclusion
The intelligibility testing indicates that the TTS systems F5TTS (German) and VoxCPM (English) generally preserve the content of the original utterances. Most sentences were transcribed with a WER of 0% or very low, demonstrating that the synthetic speech is highly understandable. XTTSv2 shows slightly higher WER, mostly on proper nouns and rare words.

The measurable WER spikes that occur correspond primarily to the ASR's known weaknesses: proper nouns, rare words, and language-specific complexities. These errors are typically not caused by the TTS system itself.

Therefore, we conclude that our two main TTS models do **not introduce major intelligibility issues** beyond what the ASR struggles with. The test confirms that TTS output is largely intelligible to automated transcription, particularly for standard vocabulary and common sentence structures.

10.2.7.4 Limitations of Automated Intelligibility Testing
It is important to note that this evaluation measures intelligibility to a specific ASR model, not to human listeners. Speech patterns that are clear to humans may confuse the ASR, and vice versa.

Moreover, the results are heavily influenced by the accuracy of the ASR itself. Misrecognitions in the ASR can inflate WER scores, giving the impression of lower TTS intelligibility even when the synthetic speech is correct.

As a result, this metric should be interpreted as an approximation rather than a definitive measure of human intelligibility. As with any WER-based metric, this evaluation only tests lexical intelligibility and not the naturalness of the model.

10.2.8 NFR-07 Usability

ID	**NFR-07** Usability
Category	Usability
Priority	Medium
Status	**Usability tests have been performed, and feedback has been implemented**
Reason	

Table 68: Validierung NFR-07 Usability

10.2.8.1 Test Results

Nr.	Observed Issue	User Feedback / Observation
1	Loading feedback	The loading screen felt very long; user wanted progress information.
2	Prompt update visibility	Update button for voice cloning prompt was not discovered because it was hidden.
3	Voice file guidance	User wanted to know how long the voice cloning file/recording should be.
4	Upload prerequisite	It could be unclear for some that a voice file had to be uploaded before starting recording.
5	Statistics behaviour	Statistics were not reset when restarting recording, while the transcription was, which was confusing.

Table 69: Usability Test Results

10.2.8.2 Key Usability Issues

Nr.	Issue	Description	Priority
1	Loading feedback	Long loading times without visible progress information.	Medium
2	Prompt update visibility	Hidden update control for voice cloning prompt reduced discoverability.	High
3	Voice file guidance	Missing guidance for minimum required voice sample length.	High
4	Upload prerequisite clarity	No visible hint that a voice file must be uploaded before recording.	High
5	Statistics reset	Statistics are not reset when starting a new recording session.	Medium

Table 70: Key Usability Issues

10.2.8.3 Next Steps (Iteration 7)

- **High-Priority Fixes**
  - Make the Update prompt button for voice cloning disabled instead of hidden. (Issue 2)
  - Clarify the minimum required length of ≥ 2 seconds for a voice cloning file/recording. (Issue 3)

- Always load a default voice cloning file in the right language so that uploading a voice file is not mandatory. (Issue 4)

- **Medium-Priority Improvements**
  - Add a visible progress indicator or percentage to the loading screen to reduce user uncertainty. (Issue 1)
  - Ensure statistics are reset together with transcription when restarting a recording session. (Issue 5)

All high-priority fixes and medium-priority improvements were successfully implemented and tested with Unit- and Component-tests.

10.2.9 NFR-08 Adaptability

ID	**NFR-08** Adaptability
Category	Supportability
Priority	Low
Status	**Target requirement of compatibility with 4 of the most used Browsers reached**
Reason	The application should be usable regardless of the user's choice of Browser

Table 71: Validierung NFR-08 Supportability

10.2.9.1 Results

Because of time constraints and no available Apple device, the testing for Safari was not performed.

FR ID	Chrome	Firefox	Edge	Opera
FR01	OK	OK	OK	OK
FR02	OK	OK	OK	OK
FR03	OK	OK	OK	OK
FR04	OK	OK	OK	OK
FR05	OK	OK	OK	OK
FR06	OK	OK	OK	OK
FR07	OK	OK	OK	OK
FR08	OK	OK	OK	OK

Table 72: Browser Test Protocol

Part III: Conclusion & Outlook

11 Conclusion

This thesis designed, implemented, and evaluated a low-latency, real-time speech translation system with voice cloning based entirely on open-source models. The primary objective was to demonstrate the technical feasibility of integrating automatic speech recognition (ASR), machine translation (MT), and text-to-speech (TTS) components into a single real-time pipeline that runs on consumer-grade hardware.

The implemented system supports real-time audio input via a web interface, incremental transcription, translation, and speech synthesis with voice cloning. Its modular architecture enables users to experiment with different ASR, MT, and TTS model combinations, and allows developers to extend the system with additional models with minimal effort. All defined functional requirements were successfully met, as verified through user acceptance testing.

Evaluation of non-functional requirements shows that the system achieves end-to-end latencies below the defined maximum threshold of five seconds, with average values ranging from 3.8 to 4.4 seconds. When excluding speech duration, the internal system latency remains between 1.4 and 1.7 seconds. Among all processing stages, TTS inference contributes the largest share of latency, a limitation inherent to current open-source TTS and MT models, which do not support fully streaming input and therefore require longer speech segments for inference.

Looking at the text recognition and translation quality, the system demonstrates strong overall linguistic performance. The ASR component achieves very low word error rates on clean speech benchmarks, with WER values around 3% and below 8% on noisy speech that includes unclear articulation and dialectal variation, confirming high transcription accuracy for general-purpose use. As expected, WER increases substantially in very noisy and domain-specific conditions such as air traffic control (ATC) communication (≈40%), highlighting known limitations of general ASR models when applied to specialised vocabularies and adverse acoustic environments.

Translation quality, measured using BLEU, remains consistently high across all evaluated language pairs. Our main machine translation model, MarianMT, achieves an overall average BLEU score of 55.5, with scores above 42 BLEU for all language pairs, indicating that semantic meaning is reliably preserved despite sentence-level context limitations. Translation quality is generally higher when English is the target language, reflecting both training data bias and simpler grammar. Together, these results show that transcription and translation errors are generally limited and do not prevent the system from producing intelligible and meaningful output in typical usage scenarios.

The evaluation of speech synthesis confirms that both naturalness and speaker similarity requirements are satisfied. Mean Opinion Scores exceed 4.0 for both German and English synthesis, indicating natural and intelligible speech. Cosine similarity scores of 0.72 (German) and 0.70 (English), together with human similarity evaluation, confirm reliable voice preservation. Automated intelligibility testing,

using ASR as the listener, shows low WER for synthetic speech (≈2–3% for F5TTS/VoxCPM), indicating that the TTS output is highly understandable and does not introduce major intelligibility issues.

In summary, this work shows that a fully open-source real-time speech translation system with voice cloning can be implemented and run effectively on consumer hardware. The system meets all functional and non-functional requirements, produces intelligible and natural speech, and provides a robust foundation for future experimentation with alternative models or configurations. While the system demonstrates strong technical feasibility, it also highlights potential risks associated with voice cloning technologies, including impersonation, fraud, and privacy concerns, underscoring the importance of responsible use.

12 Outlook

While the developed system demonstrates the feasibility of real-time speech translation with voice cloning using open-source models, several directions exist for future improvements and extensions.

A key technical limitation of the current implementation is the lack of optimised model serving and concurrency support. Currently, the system only supports a single concurrent user, as the implemented models do not natively support parallel inference through their APIs. Future work could involve developing and integrating a dedicated serving engine for TTS models, similar to vLLM, which would allow more efficient concurrency. Alternatively, parallel inference could be achieved by loading multiple instances of each model, although this approach is inefficient and increases the complexity.

In addition to concurrency, the current deployment strategy raises concerns regarding scalability as the number of supported models grows. Each model is currently hosted in its own container, which becomes increasingly unwieldy when many models are added to the system. Future implementations could explore alternative hosting strategies, such as grouping multiple models within shared containers where dependencies allow, or adopting centralised model-serving solutions optimised for inference workloads. Especially for TTS and other transformer-based models, dedicated serving engines could significantly simplify deployment while improving performance and resource utilisation.

Lower latency could be achieved by training or using future direct speech-to-speech conversion models instead of the current ASR–MT–TTS pipeline. Models that could transform the speaker's voice in real time without intermediate transcription could potentially work in full streaming mode. However, this approach would not allow for language translation, as a translation context is always required, and single words are not enough.

For continuous real-time use cases, such as translating lectures or multi-speaker conversations, the system could be extended to improve translation quality and output flexibility. Machine translation models capable of maintaining context across segments could produce more coherent translations for longer speech. Additionally, supporting multiple input sources and multiple output streams would allow the system to serve multiple listeners simultaneously. Extending the system to handle multiple speakers, with voice cloning for each participant, would require speaker recognition techniques such as speaker embeddings, as well as multiple TTS and MT model instances, increasing hardware requirements.

The current modular design allows users to select different models, but it does not provide direct control over model configuration. Future iterations could allow users to adjust settings such as synthesis speed, voice style, or other model-specific parameters directly through the web interface, enhancing usability and experimentation capabilities.

Finally, ethical considerations are an important aspect of future development. As the evaluation showed, current open-source models can produce highly convincing voice clones. Research into detecting synthesised audio and implementing safeguards against misuse, impersonation, or fraud will be critical to ensure the responsible deployment of real-time voice cloning systems.

In summary, future work could focus on optimised serving and concurrency, direct speech-to-speech conversion, context-aware translation for continuous audio, multi-speaker support, enhanced user configurability, and addressing ethical risks associated with highly realistic voice cloning.

A Glossary

Glossary A collection of terms that need to be explained.

Term	Definition
ADR	Architecture Decision Record
ASR	Automatic Speech Recognition. Also known as Speech To Text (STT)
ATC	Air Traffic Control
BLEU	Bilingual Evaluation Understudy (MT quality metric)
Cosine Similarity	Metric for comparing vector similarity (used for voice preservation)
Inference	The process of using a trained machine learning model to generate predictions or outputs.
LLM	Large Language Model
MOS	Mean Opinion Score (used for evaluating naturalness)
MT	Machine Translation
Protocol Buffers	Protocol Buffers are language-neutral, platform-neutral, extensible mechanisms for serialising structured data.
RTF	Real Time Factor. Ratio of inference time to audio duration (used for TTS)
RUP	Rational Unified Process, an iterative software development methodology with defined phases
TTS	Text To Speech
VAD	Voice Activity Detection
Voice Cloning	Technique of replicating a speaker's voice from audio samples
WER	Word Error Rate

Table 73: Glossary

B List of tools

Usage	Tool
Coding	Visual Studio Code, WebStorm, ChatGPT, GitHub Copilot
Research	Huggingface, ChatGPT, Google
Translation	DeepL
Documentation, spelling, grammar checking	Typst, Grammarly, ChatGPT
Mockups, diagrams and graphs	Draw.io, Quickchart, PyChart, Microsoft Excel, canva.com
Project management	Jira, Jira Timesheet Plugin, Microsoft Teams
DevOps	Docker, GitLab

Table 74: List of Tools

C List of tables

D List of illustrations

E List of code

F Bibliography

[1] Guillaume Klein et al., *Faster Whisper: Efficient Speech Recognition with CTranslate2*. (2023). Open Source. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/SYSTRAN/faster-whisper

[2] Open Source Contributors, *CrisperWhisper: Disfluency-Preserving Whisper Variant*. (2023). GitHub. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/nyrahealth/CrisperWhisper

[3] Alec Radford et al., 'Robust Speech Recognition via Large-Scale Weak Supervision', Dec. 06, 2022, *OpenAI*. Accessed: Dec. 13, 2025. [Online]. Available: https://openai.com/research/whisper

[4] Alpha Cephei, *Vosk Speech Recognition Toolkit*. (2023). Alpha Cephei Inc. Accessed: Dec. 13, 2025. [Online]. Available: https://alphacephei.com/vosk

[5] Facebook AI Research, *facebook/wav2vec2-base-960h*. (2020). Hugging Face Models. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/facebook/wav2vec2-base-960h

[6] Coqui AI, *Coqui STT: Open Source Speech-to-Text*. (2022). Coqui. Accessed: Dec. 13, 2025. [Online]. Available: https://stt.readthedocs.io/

[7] NVIDIA NeMo Team, *Parakeet TDT-1.1B: FastConformer ASR Model by NVIDIA NeMo*. (2025). Hugging Face Models. Accessed: Dec. 13, 2025. [Online]. Available: https://huggingface.co/nvidia/parakeet-tdt-1.1b

[8] Meta AI, *SeamlessM4T: Massively Multilingual & Multimodal Machine Translation*. (2023). Meta. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/facebookresearch/seamless_communication?tab=readme-ov-file

[9] Hugging Face, 'Hugging Face MarianMT Documentation'. Accessed: Dec. 13, 2025. [Online]. Available: https://huggingface.co/docs/transformers/en/model_doc/marian

[10] malloc, *OpenNMT-py Translator Model*. Hugging Face Models. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/malloc/OpenNMT-py-German-English-2-layer-BiLSTM

[11] Open Source Contributors, *Argos Translate: Offline Neural Machine Translation*. (2022). Argos. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/argosopentech/argos-translate

[12] Meta AI, *M2M100: Many-to-Many Multilingual Translation Model by Meta AI*. (2021). Hugging Face Models. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/facebook/m2m100_418M

[13] Meta AI, *mBART-Large-50 Many-to-Many Multilingual Translation Model by Meta AI*. (2025). Hugging Face Models. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/facebook/mbart-large-50-many-to-many-mmt

[14] Meta AI, *LLaMA 2 (7B): Meta Open Foundation Language Model*. (2023). Hugging Face Models. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/meta-llama/Llama-2-7b

[15] MosaicML NLP Team, *MPT-7B-Instruct: Instruction-Tuned Large Language Model by MosaicML*. (2023). Hugging Face Models. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/mosaicml/mpt-7b-instruct

[16] Coqui AI, *XTTSv2: Multilingual Zero-Shot Text-to-Speech*. (2023). Coqui. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/coqui-ai/TTS

[17] ResembleAI, *Chatterbox TTS*. (2023). Hugging Face. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/ResembleAI/chatterbox

[18] CPM Team, 'VoxCPM: End-to-End Streaming Neural Text-to-Speech', 2023, *Open Source*. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/OpenBMB/VoxCPM

[19] FunAudioLLM / Render-AI, *CosyVoice2*. (2024). GitHub. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/Render-AI/CosyVoice2

[20] fishaudio, *FishSpeech: Neural Text-to-Speech System*. (2023). Hugging Face. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/fishaudio/fish-speech-1.2

[21] G. S. Nikita Torgashov Gustav Eje Henter, *VoXtream*. (2025). GitHub. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/herimor/voxtream

[22] SWivid, *F5-TTS Code Repository: Flow Matching Based TTS*. (2025). GitHub. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/SWivid/F5-TTS

[23] Plachtaa, *SeedVC: Zero-Shot Voice Conversion & Singing Voice Conversion*. (2023). GitHub. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/Plachtaa/seed-vc

[24] RVC-Project, *Retrieval-based Voice Conversion WebUI*. (2023). GitHub. Accessed: Dec. 15, 2025. [Online]. Available: https://github.com/RVC-Project/Retrieval-based-Voice-Conversion-WebUI

[25] Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, 2009.

[26] Open Source Contributors, *jiwer: Word Error Rate Computation*. (2023). GitHub. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/jitsi/jiwer

[27] J. Zuluaga, *ATCO2 Corpus 1h*. (Sep. 08, 2022). Hugging Face Datasets. Accessed: Dec. 13, 2025. [Online]. Available: https://huggingface.co/datasets/Jzuluaga/atco2_corpus_1h

[28] Avemio, *ASR German Mixed Test Set*. (2025). Hugging Face Datasets. Accessed: Dec. 13, 2025. [Online]. Available: https://huggingface.co/datasets/avemio/ASR-GERMAN-MIXED-TEST

[29] OpenSLR, *LibriSpeech ASR Corpus*. (2015). Hugging Face Datasets. Accessed: Dec. 13, 2025. [Online]. Available: https://huggingface.co/datasets/openslr/librispeech_asr

[30] Matt Post, *sacreBLEU: Standardized BLEU Score Evaluation*. (2018). ACL. Accessed: Dec. 13, 2025. [Online]. Available: https://github.com/mjpost/sacrebleu

[31] Tatoeba Project, 'Tatoeba: A Large Database of Sentences and Translations'. Accessed: Dec. 13, 2025. [Online]. Available: https://tatoeba.org/

[32] SpeechBrain Community, *ECAPA-TDNN Speaker Embedding Model*. (2021). Hugging Face. Accessed: Dec. 15, 2025. [Online]. Available: https://huggingface.co/speechbrain/spkrec-ecapa-voxceleb

[33] Franz Kafka, 'Works by Franz Kafka - LibriVox Collection'. Accessed: Dec. 16, 2025. [Online]. Available: https://librivox.org/author/100?primary_key=100&search_category=author&search_page=1&search_form=get_results&search_order=alpha

[34] O. Henry, 'LibriVox 2006 Christmas Short Works Collection'. Accessed: Dec. 16, 2025. [Online]. Available: https://librivox.org/librivox-2006-christmas-short-works-collection/

[35] J. P. Zuluaga-Gomez, 'Lessons Learned in Transcribing 5000 Hours of Air Traffic Control Speech', Oct. 01, 2023, *Aerospace (MDPI)*. Accessed: Dec. 13, 2025. [Online]. Available: https://www.mdpi.com/2226-4310/10/10/898

[36] Z. Wang, 'Enhancing Air Traffic Control Communication Systems with Automatic Speech Recognition', Jul. 01, 2024, *Sensors (MDPI)*. Accessed: Dec. 13, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/24/14/4715

[37] 'Modulbeschreibung: Student Research Project Informatik', Nov. 27, 2025, *OST – Ostschweizer Fachhochschule*. Accessed: Dec. 13, 2025. [Online]. Available: https://studien.ost.ch/allModules/40906_M_SAI21.html

G Appendix

The appendix contains all documents that are not essential for understanding the thesis but nevertheless provide added value and are sometimes needed for the thesis. These include, among others, the authors' personal conclusion, the project plan, full manual test evaluation results and the module description of the thesis.

G.1 Project Plan

G.1.1 Processes

As the project method, we apply a very lightweight variant of Scrum suitable for a two-person team mixed with RUP. Iterations span over two weeks, except for the last iteration. We hold short sprint reviews and planning, and also meet up for a “Daily” two times a week.

G.1.2 Team Roles

Who	Responsibilities
Tareq Kattit	Risk Management, NFRs / Testing, Development
David Bürge	Issue Management, Functional Requirements, Architecture, Development

Team Roles

G.1.3 Meetings

The following table lists the meetings that are held regularly in the project.

When	Who	What
Every Tuesday and Saturday Morning	Team	“Daily” Scrum
Every second Wednesday Evening	Team	Sprint Review / Planning
Every Thursday at 02:00	Team, Advisor	Weekly Review

Meetings

G.1.4 High Level Project Plan

High Level Project Plan		Inception	Elaboration		Construction			Transition
		18 Sep - 1 Oct	2 - 15 Oct	16 - 29 Oct	30 Oct - 12 Nov	13 - 26 Nov	27 Nov - 10 Dec	11 - 19 Dec
	Hours	Iteration 1	Iteration 2	Iteration 3	Iteration 4	Iteration 5	Iteration 6	Iteration 7
Project management	60							
Project planning	30							
Requirement Engineering	20							
Tools / Infrastructure	30							
Software Architecture	20							
Testing	30							
Research / Evaluation	90							
ASR	60							
TTS	60							
MT	30							
Configuration / Statistics	50							
Total:	480							
Milestones:		M1		M2			M3	M4

High Level Project Plan

G.1.5 Milestones

M1 - End of Inception

- Initial project plan is defined

- Initial requirements are defined and reviewed
- Initial risks are identified
- Tools and infrastructure are set up

M2 - End of Elaboration
- Models are evaluated, and it's decided which will be used
- Software architecture is defined
- All quality measures, including test concepts, are defined

M3 - End of Construction
- All features are implemented
- All tests are implemented and/or executed

M4 - End of Transition
- Documentation is finalised
- Product was demonstrated

G.1.6 Risk Management

Various risks are identified for the project, which are categorised in the following matrix. The probability of occurrence is listed on the left, and the impact is listed at the top. The individual risks are then described, along with the measures taken to mitigate them. After each sprint, an evaluation is carried out to determine whether the risks have occurred and whether they have moved within the matrix.

Number	1
Risk	Radio input audio causes poor results
Impact	Cosine Similarity, MOS, WER score low
Measures	R1 occurred during Construction. Radio-quality input produces unusable results (high WER). The requirement was removed for project feasibility during inception. This is a residual limitation, but does not block further development.

Risk 1: Poor audio input quality

Number	2
Risk	High latency
Impact	Latency requirement of sub 1 second can not be reached
Measures	Initial mitigation attempted via separate Speech-to-Speech pipeline. Due to a lack of viable models, the latency requirement was revised to 4 seconds, which was achievable with a batch-based approach.

Risk 2: High latency

Number	3
Risk	Dependency issues with third-party models or frameworks
Impact	Pipeline may break, performance issues, licensing problems
Measures	Test dependencies before adoption, maintain fallbacks

Risk 3: Third-party dependencies

Number	4
Risk	Integration delays between frontend and backend
Impact	Features blocked, incomplete UI, slow development progress
Measures	Provide API mocks early, work on Epics separately

Risk 4: Frontend-backend integration delays

Number	5
Risk	Streaming not feasible since most models are batch-based
Impact	Inability to achieve real-time processing, degraded user experience
Measures	During the evaluation phase, we tested candidate models for latency, stability, and accuracy. While the models were inherently batch-based, we were able to implement incremental processing and buffering techniques that made the system appear to operate in a streaming fashion. As a result, the system achieves near real-time performance and maintains a good user experience, effectively mitigating the risk.

Risk 5: Streaming feasibility

Number	6
Risk	Insufficient testing of edge cases
Impact	Unexpected failures with accents, background noise, or mixed languages
Measures	Test with diverse speakers, test edge-cases

Risk 6: Edge case failures

Number	7
Risk	TTS or STS does not correctly preserve speaker accents
Impact	Speech output sounds unnatural, loss of speaker identity, reduced user satisfaction
Measures	Accent preservation in TTS or STS is not feasible with available models. The system supports speech synthesis and transformation, but may not preserve speaker accents.

Risk 7: Accent preservation failure in TTS

Number	8
Risk	Unknown risks emerge due to unfamiliar subject matter
Impact	Delayed development, missed opportunities, having to work around on otherwise easy solutions
Measures	Conduct early research

Risk 8: Unknown subject-related risks

G.1.6.1 Risk Matrix at the beginning of Iteration 1

	Low	Medium	High	Very High	Critical
Impossible					
Unlikely			R3		
Possible		R6		R2, R5	
Likely		R4, R7	R8		R1
Most likely					

Risk matrix at the beginning of the project

G.1.6.2 Update after Iteration 1

	Low	Medium	High	Very High	Critical
Impossible					
Unlikely			R2, R3		
Possible		R4, R6		R2, R5	
Likely	R1	R7	R8		
Most likely					

Risk matrix after Iteration 1

G.1.6.3 Update after Iteration 2

	Low	Medium	High	Very High	Critical
Impossible					
Unlikely			R3	R5	
Possible		R4, R6			
Likely	R1	R7	R8		
Most likely			R2		

Risk matrix after Iteration 2

G.1.6.4 Update after Iteration 3

	Low	Medium	High	Very High	Critical
Impossible					
Unlikely		R4	R3		
Possible		R6			
Likely	R1	R7	R8		
Most likely					

Risk matrix after Iteration 3 (End of Elaboration)

R2 occurred during Elaboration. As no viable Speech-to-Speech models were available, the original latency requirement was revised. With the updated requirement of 4 seconds, latency was no longer a blocking risk for project success.

R5 was mitigated.

G.1.6.5 Update after Iteration 4

	Low	Medium	High	Very High	Critical
Impossible					
Unlikely		R4	R3		
Possible		R6			
Likely	R1	R7	R8		
Most likely					

Risk matrix after Iteration 4

G.1.6.6 Update after Iteration 6

	Low	Medium	High	Very High	Critical
Impossible					
Unlikely		R4	R3		
Possible		R6			
Likely		R7	R8		
Most likely					

Risk matrix after Iteration 6 (End of Construction)

R1 occurred during Construction. ASR model testing during Construction revealed that such input produces unusable results (high WER).

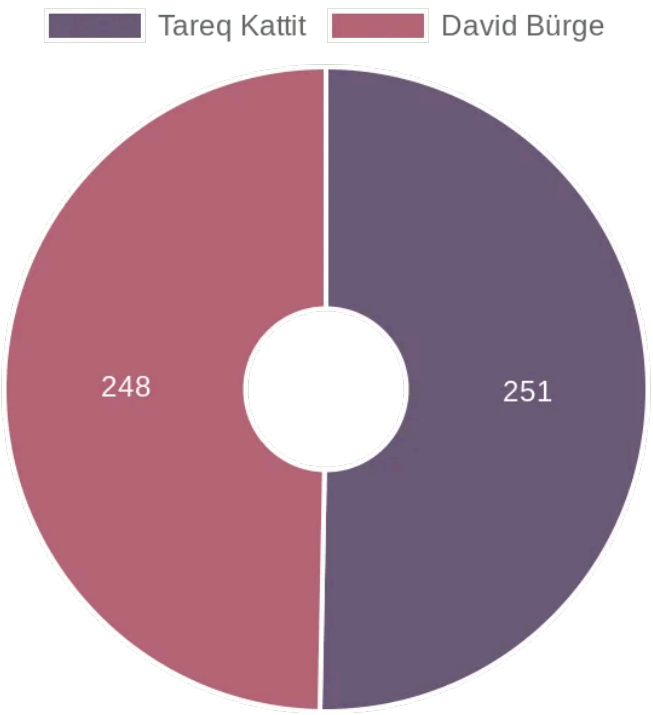
G.1.6.7 Update after Iteration 7

	Low	Medium	High	Very High	Critical
Impossible					
Unlikely		R4	R3		
Possible		R6			
Likely		R7			
Most likely					

Risk matrix after Iteration 7 (End of Transition – Project Completion)

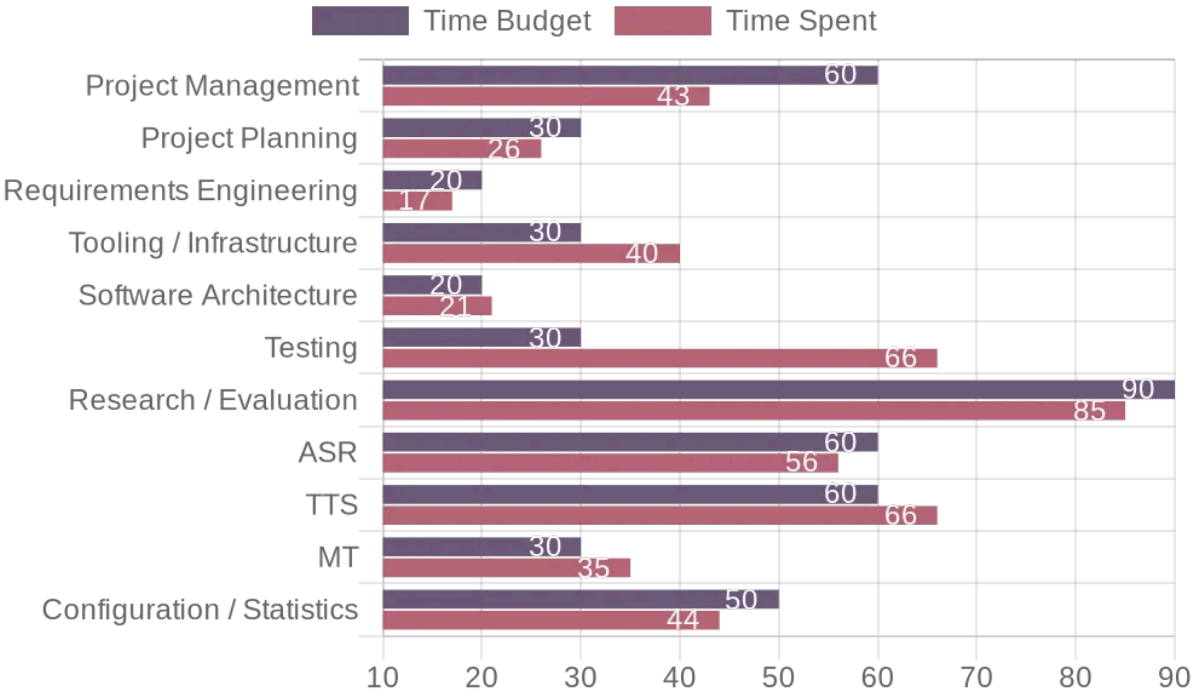
G.2 Time Tracking Report

G.2.1 Time per work log author



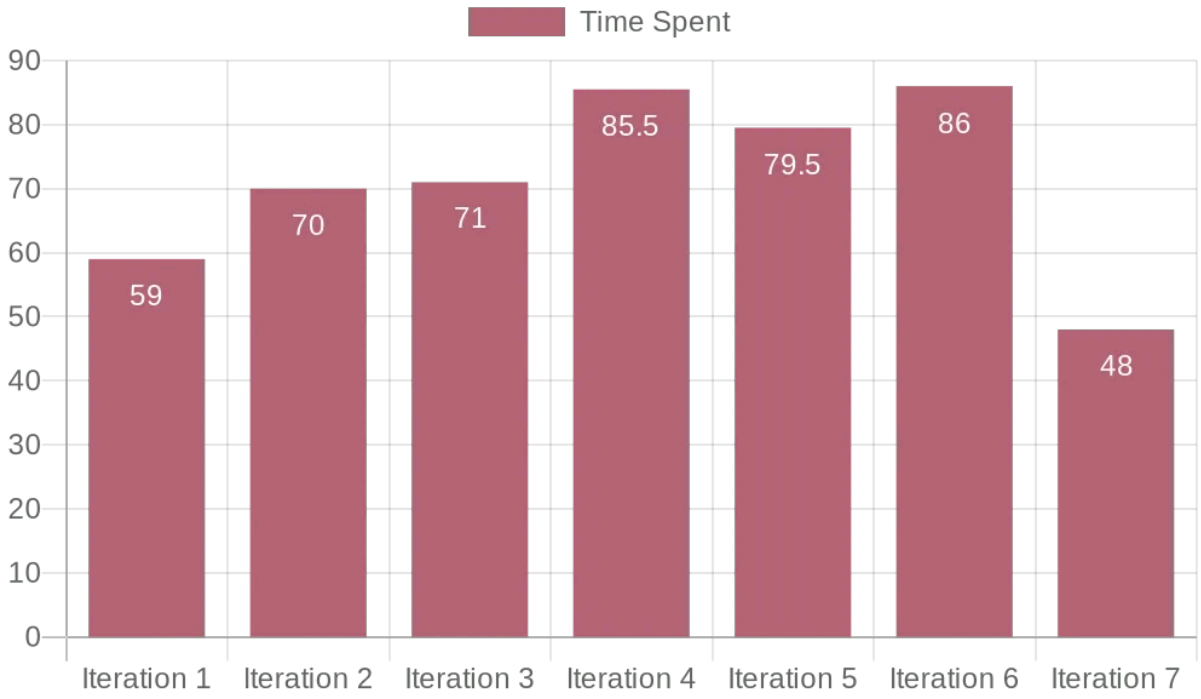
Time spent by work log author

G.2.2 Time per epic



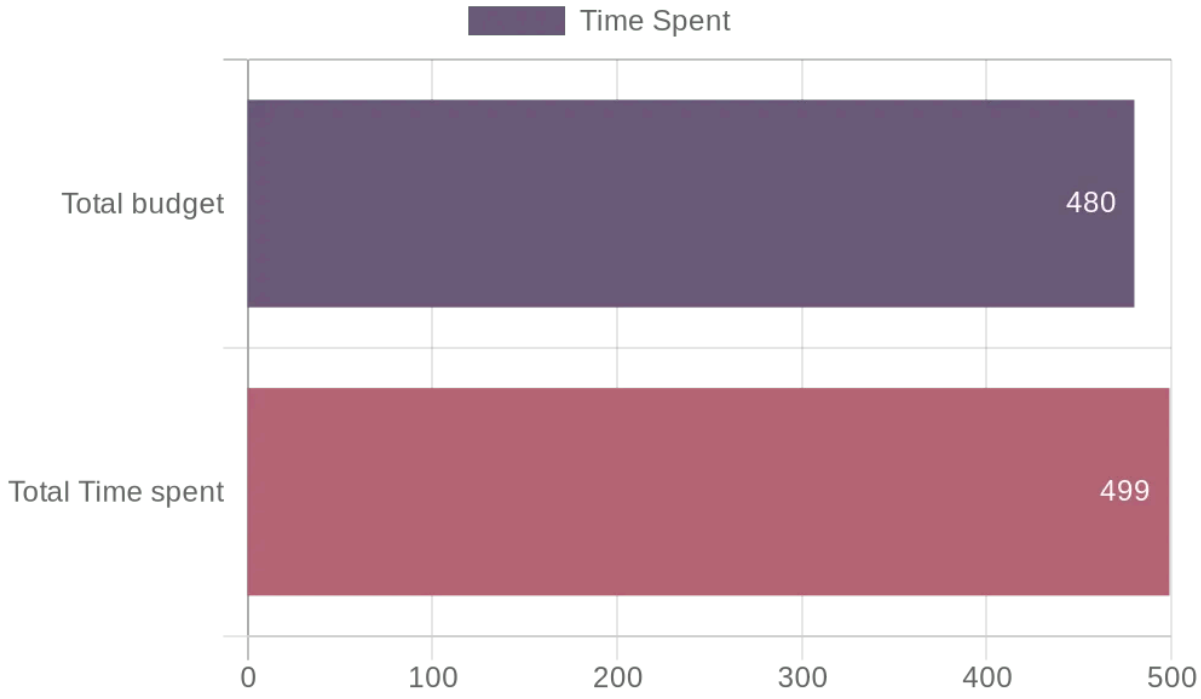
Time spent per epic

G.2.3 Time per iteration



Time spent per iteration

G.2.4 Time total



Total time spent

G.3 Personal Reports

G.3.1 Tareq Kattit

This project has been an intense and highly educational experience. One of the most significant challenges was working in a domain where I had little to no prior experience. While I was familiar with general software development, the fields of automatic speech recognition (ASR), text-to-speech (TTS), and machine translation (MT) were largely new to me. Understanding how these models work, evaluating the available open-source options, and finding combinations that could deliver real-time performance while maintaining acceptable quality was a steep learning curve. This required not only reading through blogs and documentation but also hands-on experimentation to identify models that could work together efficiently in a low-latency pipeline, while balancing speed and quality.

A particularly difficult phase was the final model evaluation, which took much longer than I had anticipated. Testing each model to verify its performance, quality, and then analysing these test results proved more time-consuming than expected. I realised that much of the effort went into evaluating models that were essentially "black boxes". And while these results were interesting to see, the black box nature of the models limited the control we had over the output and made it challenging to fully optimise the system.

On the other hand, our prior experience with React was a significant advantage during the project. It allowed us to implement the web-based interface efficiently, providing a responsive and user-friendly frontend for the system.

It was also rewarding that we designed the prototype to be modular, allowing individual components to be swapped out in the future. This ensures that the system can evolve and incorporate new models, which is especially valuable in the fast-paced field of AI.

Looking back, the project also highlighted areas for personal improvement, such as planning and time estimation for tasks in completely new technical domains. Despite these challenges, the project was highly rewarding. It strengthened my problem-solving skills, deepened my understanding of speech and translation technologies, and improved my ability to work independently in unfamiliar areas. Successfully navigating these difficulties and contributing to a working prototype that demonstrates the feasibility of real-time voice-preserving translation is one of the key achievements of this project.

G.3.2 David Bürge

Working with the available APIs for open-source TTS models turned out to be more challenging than initially expected. Although using the APIs themselves to run inference was not particularly difficult, many of them are not production-ready and are poorly documented. There is little control over how the models are executed internally, which often requires working around existing implementation details rather than designing clean integrations.

Looking back, it also became clear that the extent to which we tested the models did not entirely make sense. Apart from selecting which models to implement, we had very little influence on the actual output quality. As a result, we spent a substantial amount of time testing something we had not truly implemented ourselves, but rather the models themselves. Additionally, there were simply too many aspects to evaluate, such as naturalness, voice similarity, intelligibility, word error rate, and translation quality. Given the limited time available, it was not possible to test each metric thoroughly, which led to only superficial evaluations while still consuming far more time than originally planned.

Also, in hindsight, I would have chosen a different technology for implementing the orchestrator, such as .NET. Python was selected because it seemed simpler and avoided adding another technology to the stack, and because the team was generally more familiar with it. However, implementing even a

moderately clean architecture in Python proved to be difficult. The lack of ahead-of-time compilation and strong type checking led to a worse development experience, where small mistakes could easily slip in, and refactoring was more error-prone. Compared to this, platforms like .NET provide more mature infrastructure, such as better dependency injection and a stronger object-oriented focus, which would likely have improved maintainability and development speed.

Despite these challenges, there were also positives; the modular approach of the solution was one of the most rewarding aspects of the project. Being able to choose different models for each stage of the pipeline is a significant achievement. Though adding many models quickly becomes unwieldy due to each model running in its own container. In some cases, models such as machine translation could potentially be hosted within shared containers. As mentioned in the outlook, this also highlights the need for a proper serving engine specifically designed for TTS.

Also, although the overall system latency is moderately high, mainly because speech segments need to be fully received before processing, it is still impressive that the models were able to run in real time within an asynchronous pipeline. The decision to use voice activity detection (VAD) to split the input into manageable segments was particularly effective and proved to be a key design choice.

Overall, considering the limited time available, the project resulted in an impressive prototype. As is typical for prototype implementations, there remain numerous opportunities for improvement. Nevertheless, the solution successfully demonstrates the feasibility of the approach and provides a solid foundation for future development.

G.4 Module description Computer Science thesis

Excerpt from the module description on https://studien.rj.ost.ch/ [37].

G.4.1 Meta Data

Abbreviation	M_SAI21
Implementation period	HS/25
ECTS points	8
Learning objectives	The thesis is intended to demonstrate the candidate's problem-solving skills using engineering methods. Accordingly, the thesis consists of conceptual, theoretical, and practical components.
Responsible person	Stocker Mirko
Location (offered)	Rapperswil-Jona
Required modules	SE Project (M_SEProj, FS/25)
Additional prerequisite knowledge	none

Module description Computer Science thesis

G.4.2 More detailed description

The term papers are usually completed in teams of two and supervised by a lecturer. The term paper lasts approximately 14 weeks, requiring a total workload of at least 240 hours. The topic of the paper can be proposed by an external company, the lecturer, or the students. The final task is set by the supervising lecturer. The paper should demonstrate problem-solving skills using engineering methods. Accordingly, the paper should include conceptual, theoretical, and practical components. Typically, a computer science project is completed with the following subtasks:

- Familiarization with a new problem
- Project planning
- Requirements analysis (including environment description and definition)
- Design and implementation of the solution (including assessment of the state of the art)
- Testing the solution
- Evaluation and outlook

G.5 Detailed Subjective Similarity Evaluation Results

Similarity Evaluation Test Subject 1

Trial (German)	# Bewertung
1	2
2	3
3	1
4	2
5	4
6	3
7	3
8	4
9	3
10	2
11	4
12	4
13	1
14	3
15	3
16	1
17	3
18	2
19	1
20	4
Generated scored as similar	10
Percentage >= 3	0.7142857143

Trial (English)	# Bewertung
1	4
2	1
3	2
4	3
5	4
6	4
7	4
8	1
9	4
10	4
11	4
12	4
13	3
14	4
15	3
16	4
17	4
18	1
19	3
20	2
Generated scored as similar	12
Percentage >= 3	0.8571428571

Similarity Evaluation Test Subject 2

Trial (German)	# Bewertung
1	4
2	3
3	2
4	4
5	4
6	3
7	2
8	4
9	2
10	1
11	3
12	4
13	1
14	4
15	2
16	3
17	2
18	1
19	1
20	2
Generated scored as similar	7
Percentage >= 3	0.5

Trial (English)	# Bewertung
1	3
2	1
3	1
4	4
5	4
6	4
7	3
8	1
9	2
10	4
11	4
12	3
13	3
14	4
15	2
16	1
17	4
18	1
19	3
20	3
Generated scored as similar	11
Percentage >= 3	0.7857142857

Similarity Evaluation Test Subject 3

Trial (German)	#	Bewertung
	1	4
	2	4
	3	1
	4	4
	5	4
	6	4
	7	2
	8	4
	9	2
	10	3
	11	3
	12	2
	13	3
	14	3
	15	3
	16	3
	17	1
	18	2
	19	1
	20	2
Generated scored as similar		8
Percentage >= 3		0.5714285714

Trial (English)	#	Bewertung
	1	4
	2	2
	3	1
	4	3
	5	4
	6	4
	7	2
	8	1
	9	2
	10	4
	11	4
	12	4
	13	2
	14	4
	15	4
	16	4
	17	4
	18	1
	19	3
	20	3
Generated scored as similar		10
Percentage >= 3		0.7142857143

Similarity Evaluation Test Subject 4

Trial (German)	# Bewertung
1	4
2	4
3	2
4	4
5	4
6	4
7	3
8	3
9	3
10	2
11	4
12	4
13	1
14	4
15	3
16	3
17	2
18	3
19	1
20	4
Generated scored as similar	12
Percentage >= 3	0.8571428571

Trial (English)	# Bewertung
1	4
2	1
3	1
4	4
5	3
6	4
7	4
8	1
9	3
10	3
11	4
12	4
13	4
14	4
15	4
16	3
17	4
18	1
19	3
20	4
Generated scored as similar	13
Percentage >= 3	0.9285714286

Similarity Evaluation Test Subject 5

Trial (German)	Bewertung
1	3
2	4
3	1
4	4
5	4
6	4
7	4
8	4
9	3
10	4
11	4
12	4
13	1
14	4
15	4
16	4
17	3
18	2
19	2
20	3
Generated scored as similar	13
Percentage >= 3	0.9285714286

Trial (English)	Bewertung
1	4
2	2
3	1
4	4
5	4
6	4
7	4
8	2
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	4
17	4
18	1
19	4
20	4
Generated scored as similar	13
Percentage >= 3	0.9285714286

Similarity Evaluation Test Subject 6

Trial (German)	Bewertung
1	4
2	4
3	1
4	3
5	4
6	4
7	4
8	3
9	2
10	3
11	4
12	4
13	1
14	4
15	4
16	4
17	3
18	2
19	2
20	3
Generated scored as similar	12
Percentage >= 3	0.8571428571

Trial (English)	Bewertung
1	3
2	1
3	1
4	2
5	4
6	4
7	3
8	1
9	4
10	4
11	3
12	3
13	4
14	4
15	4
16	3
17	3
18	1
19	3
20	4
Generated scored as similar	12
Percentage >= 3	0.8571428571

Similarity Evaluation Test Subject 7

Trial (German)	Bewertung
1	4
2	4
3	1
4	4
5	4
6	4
7	4
8	4
9	3
10	4
11	4
12	4
13	1
14	4
15	4
16	4
17	3
18	2
19	2
20	3
Generated scored as similar	13
Percentage >= 3	0.9285714286

Trial (English)	Bewertung
1	4
2	2
3	1
4	3
5	4
6	4
7	3
8	2
9	4
10	4
11	4
12	4
13	4
14	4
15	4
16	4
17	4
18	1
19	4
20	4
Generated scored as similar	13
Percentage >= 3	0.9285714286

Similarity Evaluation Test Subject 8

Trial (German)	Bewertung
1	4
2	5
3	5
4	5
5	5
6	5
7	5
8	5
9	5
10	5
11	5
12	5
13	5
14	5
15	4
16	4
17	5
18	5
19	5
20	5
Generated scored as similar	14
Percentage >= 3	1

Trial (English)	Bewertung
1	3
2	2
3	1
4	3
5	4
6	4
7	3
8	2
9	4
10	4
11	3
12	4
13	4
14	4
15	4
16	4
17	4
18	1
19	4
20	4
Generated scored as similar	13
Percentage >= 3	0.9285714286

G.6 Detailed MOS Naturalness Evaluation Results

Naturalness Evaluation Test Subject 1

Trial (German)	Bewertung
1	3
2	4
3	4
4	5
5	5
6	5
7	5
8	4
9	5
10	4
11	5
12	4
13	4
14	5
15	5
16	3
17	4
18	5
19	5
20	4
Sum	65
Average	4.333333333

Trial (English)	Bewertung
1	4
2	5
3	5
4	3
5	5
6	5
7	5
8	4
9	4
10	5
11	5
12	4
13	5
14	3
15	4
16	4
17	5
18	5
19	4
20	4
Sum	63
Average	4.2

Naturalness Evaluation Test Subject 2

Trial (German)	Bewertung
1	4
2	4
3	5
4	5
5	5
6	3
7	5
8	5
9	4
10	5
11	4
12	4
13	4
14	4
15	3
16	3
17	5
18	4
19	5
20	4
Sum	64
Average	4.266666667

Trial (English)	Bewertung
1	5
2	3
3	5
4	4
5	5
6	5
7	5
8	4
9	3
10	5
11	4
12	5
13	5
14	5
15	5
16	4
17	5
18	4
19	4
20	4
Sum	65
Average	4.333333333

Naturalness Evaluation Test Subject 3

Trial (German)	Bewertung
1	5
2	5
3	5
4	5
5	4
6	5
7	5
8	5
9	5
10	5
11	5
12	4
13	5
14	5
15	5
16	5
17	5
18	5
19	4
20	5
Sum	73
Average	4.866666667

Trial (English)	Bewertung
1	5
2	5
3	5
4	3
5	5
6	5
7	5
8	4
9	5
10	5
11	4
12	5
13	5
14	5
15	5
16	5
17	4
18	3
19	5
20	4
Sum	69
Average	4.6

Naturalness Evaluation Test Subject 4

Trial (German)	Bewertung
1	3
2	5
3	5
4	4
5	4
6	4
7	4
8	5
9	3
10	4
11	3
12	5
13	4
14	4
15	3
16	2
17	5
18	5
19	4
20	4
Sum	59
Average	3.933333333

Trial (English)	Bewertung
1	5
2	2
3	5
4	3
5	5
6	4
7	5
8	2
9	4
10	5
11	5
12	3
13	2
14	4
15	5
16	4
17	4
18	3
19	5
20	4
Sum	56
Average	3.733333333

Naturalness Evaluation Test Subject 5

Trial (German)	Bewertung
1	4
2	5
3	5
4	5
5	3
6	5
7	4
8	5
9	3
10	4
11	4
12	5
13	5
14	4
15	4
16	5
17	4
18	5
19	5
20	3
Sum	65
Average	4.333333333

Trial (English)	Bewertung
1	4
2	5
3	5
4	5
5	5
6	5
7	5
8	5
9	5
10	5
11	5
12	5
13	5
14	5
15	4
16	5
17	5
18	4
19	5
20	4
Sum	71
Average	4.733333333

Naturalness Evaluation Test Subject 6

Trial (German)	Bewertung
1	3
2	4
3	4
4	4
5	3
6	4
7	3
8	5
9	2
10	5
11	3
12	5
13	4
14	5
15	4
16	4
17	3
18	4
19	4
20	5
Sum	58
Average	3.866666667

Trial (English)	Bewertung
1	3
2	4
3	5
4	4
5	5
6	5
7	5
8	5
9	5
10	5
11	5
12	5
13	5
14	5
15	4
16	4
17	5
18	5
19	5
20	5
Sum	69
Average	4.6

Naturalness Evaluation Test Subject 7

Trial (German)	Bewertung
1	4
2	5
3	5
4	5
5	3
6	5
7	4
8	5
9	3
10	5
11	4
12	5
13	5
14	4
15	4
16	4
17	4
18	4
19	5
20	4
Sum	65
Average	4.333333333

Trial (English)	Bewertung
1	4
2	5
3	5
4	5
5	5
6	5
7	5
8	5
9	5
10	5
11	5
12	5
13	5
14	5
15	4
16	4
17	5
18	5
19	5
20	5
Sum	72
Average	4.8

Naturalness Evaluation Test Subject 8

Trial (German)	Bewertung
1	4
2	5
3	5
4	5
5	3
6	5
7	4
8	5
9	3
10	5
11	4
12	5
13	5
14	4
15	4
16	4
17	4
18	4
19	5
20	4
Sum	65
Average	4.333333333

Trial (English)	Bewertung
1	4
2	5
3	5
4	5
5	5
6	5
7	5
8	5
9	5
10	5
11	5
12	5
13	5
14	5
15	4
16	4
17	5
18	5
19	5
20	5
Sum	72
Average	4.8