Universiteit Leiden
The Netherlands

Computer Science & Economics

Improving the Software Requirements Elicitation Process by Integrating AI-Driven Speech Functionalities

Kalsoom Zia (s3214729)

First Supervisor: G.J. Ramackers
Second Supervisor: J.M.W. Visser

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 2024-08-27

Acknowledgments

I would like to express my deepest gratitude to all those who have supported and guided me throughout the process of completing this thesis.
First and foremost, I am profoundly grateful to my supervisors, Guus Ramackers and Joost Visser, whose invaluable insights, unwavering support, and expert guidance have been instrumental in shaping this research. Your patience, encouragement, and constructive feedback have been the driving forces behind my progress.
I would also like to extend my heartfelt thanks to the students, Sven Dam and Mimo Amghar, for their time, valuable suggestions, and helping me with the design and architecture.
A special thank you to my family and friends, whose moral support made the challenges of research more bearable. The countless discussions, brainstorming sessions, and shared experiences have enriched my academic journey.
I am deeply grateful to my family, whose love, patience, and encouragement have been my foundation. To my mother, for her unwavering belief in me, and to my sister, for your endless support, understanding, and for always being my greatest source of strength.
Lastly, I would like to acknowledge LIACS, whose education made this research possible. The resources and opportunities provided have been vital in the completion of this work.
Thank you all for your invaluable contributions to this thesis. This achievement would not have been possible without your collective efforts and support.

Abstract

In today's rapidly evolving software development landscape, the process of eliciting requirements is recognized as a crucial phase in the development life cycle. Accurate and efficient requirements elicitation is essential for developing successful software solutions that meet user needs and project goals. To address the complexities involved in this process, the AI4MDE-Studio program has developed a process to facilitate and streamline the elicitation of requirements.

The primary objective of this study is to enhance this system by integrating an additional tool designed specifically to simplify the requirements elicitation process, particularly for non-technical users. This enhancement aims to make the requirements gathering process more accessible and user-friendly, thereby improving overall efficiency and effectiveness.

To achieve this, the study investigates various open-source Text-to-Speech (TTS) and Speech-to-Text (STT) engines to determine the most suitable options for integration into the system. A comprehensive comparison of these engines was conducted, involving an evaluation against a set of predefined benchmarks. This evaluation was based on several performance metrics, including accuracy, ease of integration, and overall reliability.

Through this comparative analysis, Whisper was identified as the optimal choice for STT, while Google Cloud TTS emerged as the best option for TTS.

Subsequently, an architectural framework for an API was developed to integrate these chosen models into the existing system. This architecture facilitates seamless interaction between the TTS and STT engines and the requirements elicitation tool.

To validate the effectiveness and robustness of the integrated models, a validation study was conducted. Participants were asked to provide two audio recordings each, with each recording addressing a specific question from two distinct locations. This approach was designed to test the models under varied conditions and assess their performance in real-world scenarios.

The results from this validation process will provide insights into the effectiveness of the integrated TTS and STT engines, ensuring that the enhanced system meets the needs of its users and performs reliably in diverse settings.

Contents

1 Introduction

1.1 Context

In today's fast-paced software development landscape, the elicitation of requirements stands as a critical phase in the development life cycle. However, traditional methods of requirements gathering often pose challenges, particularly for users who may not possess extensive expertise in formal modelling techniques. This research builds on the findings of previous studies conducted by other students who explored the challenges of traditional methods for collecting requirements. They have investigated how a Natural Language Processing (NLP) based chatbot technology could transform the process of gathering and streamlining requirements.

1.2 Problem statement

The NLP-based chatbot technology is designed to empower users to articulate their requirements directly and streamline the formal modelling process from its inception. The chatbot is thoroughly engineered to provide to users who may not be well-versed in the complexities of the requirements process, thereby democratizing access to this critical aspect of software development [Bha23]. Through a thoroughly crafted hybrid approach, it seamlessly combines direct input processing with a sophisticated conversation strategy component. This prototype empowers the chatbot to generate UML diagrams, laying the groundwork for further refinement in a collaborative 'human-in-the-loop' approach involving analysts and users [Bha23]).

The thesis project identifies manual steps in the 'human-in-the-loop' approach that can be automated or simplified, aiming to reduce the time users spend on these tasks. By integrating a Large Language Model (LLM) and speech recognition function with the interactive chatbot, the project seeks to not only enhance the efficiency of the elicitation process but also reduce the time burden placed on users, making it more accessible, efficient, and user-friendly.

1.3 Research approach

The methodology for this research involves several key steps. Firstly, a comprehensive literature review will be conducted to gather insights into voice recognition technologies, including their capabilities, limitations, and best practices. This review will encompass academic papers, reports, and technical documentation to inform the development and integration of voice functionality into web programs.

Subsequently, research into various voice recognition APIs, libraries, and platforms will be undertaken to assess their suitability for integration into the system. Different methods and techniques for voice input processing will be explored, such as speech-to-text conversion and natural language understanding, to determine the most effective approach.

Once the necessary background research is completed, the development phase will start. A prototype will be developed using appropriate programming languages and frameworks, with a focus on integrating voice recognition functionality seamlessly into the system. A hybrid approach will be employed in the AI4MDE system for processing voice inputs, combining direct input processing with a conversation strategy component to ensure a user-friendly experience.

Following the development phase, user testing and feedback collection will be conducted. A small group of users will be recruited to participate in usability testing of the voice-recognition model. Users will be given questions to answer by sending a voice recording.

After receiving their recordings, the audio files will be sent to the model to test how well the model works. This methodology systematically investigates integrating voice functionality into the AI4MDE program for software requirements elicitation, ensuring the reliability and validity of research findings. The implementation of the model integration and the results of user testing will be the key factors in addressing the research question.

1.4 Research question

As, through this research, the aim is to transform the landscape of requirements elicitation, making it more accessible, efficient, and user-friendly, the ensuing research questions arise:
How can speech functiontionalities be integrated in the requirements elicitation process, and what are the perceived benefits and challenges?

This overarching question can be further divided into the following research objectives:

1. What are the best practices for designing an API within the requirements elicitation software to communicate with AI-based speech functionalities?
2. What are the advantages and limitations of continuous flow versus batch processing for speech functionalities?
3. How can AI-Driven Speech Functionalities effectively be used in a requirement elicitation process where chat-like interaction needs to be combined with complex tabular and/or visual information?
4. How can the accuracy of speech models be effectively measured and which metrics are most relevant for assessing performance?

1.5 Deliverables

1. **Tool Development/Code**
   - Python code, along with the necessary requirements, to create a functional prototype of the enhanced software requirements elicitation process.
2. **Performance Benchmark**
   - A performance benchmark comparing the chosen Speech-to-Text (STT) engines based on various criteria such as accuracy, speed, robustness, and background knowledge.
3. **The Comprehensive Report**

- A clear outline of the methodologies, techniques, and steps that will be employed to create the code for building the chatbot and integrating voice recognition.
- Challenges encountered during the project.
- The specific use cases that the chatbot will address.
- The decision-making process, including the choices made.

4. **Presentation**
   - A presentation showcasing the result.
   - A live demonstration of the tool.

1.6 Thesis overview

This thesis is organized into several key sections, each focusing on a distinct aspect of the research. Chapter 2, titled "Background," reviews prior research relevant to the study, explores the potential of voice technologies, and investigates the process of identifying the most effective Speech-to-Text (STT) and Text-to-Speech (TTS) engines. Chapter 3, labeled "Methodology," details the data-gathering methods, the criteria for selecting suitable technologies, and the benchmarking process used to evaluate the STT and TTS engines. Chapter 4, "Architecture," describes the conceptual and technical framework of the proposed system, offering an in-depth overview of its design and components. In Chapter 5, "Validation," the experimental setup is outlined, the candidates selected for testing are discussed, and the results of the validation process are presented. Chapter 6, "Discussion," interprets the results, explores their implications, examines the impact of the findings on the research objectives, and provides recommendations for future work.. The final chapter, Chapter 7, "Conclusion," summarizes the thesis's key findings, and reflects on the research outcomes. Lastly, the References section includes all the sources cited throughout the thesis.

2 Background

In this section, an overview is provided of various prior research studies related to requirements elicitation, with a particular focus on the evolution and application of chat and voice functionalities in this domain. Requirements elicitation is a critical phase in the development of systems, where understanding and capturing user needs is essential for creating effective solutions. Recent advancements have introduced new tools and techniques, including those that leverage voice functionalities, which have significantly impacted how requirements are gathered and analyzed [MS19].

2.1 Related Work

The paper by Rietz and Maedche (2019) introduces LadderBot, a system designed to support self-elicitation of requirements. LadderBot leverages a structured, digital approach to the laddering interview technique, aiming to improve the process of capturing user needs and insights. This tool is intended to enhance the efficiency and effectiveness of requirements gathering by automating aspects of the elicitation process [MS19]. According to Rietz and Maedche, the best methods for Requirement Elicitation (RE) is the laddering interview technique, which has been recognized for its ability to produce structured and comprehensive insights due to its hierarchical nature. This technique involves an interviewer starting with a seed attribute—an initial topic or feature—and using a series of probing ”why...?” questions to explore deeper layers of user needs and underlying attitudes [MS19]. Tim Rietz highlights that chatbots are an increasingly effective form of conversational agents (CAs) designed to connect with a broad audience of end-users. Their immediate accessibility and compatibility across various platforms enable smooth and barrier-free interaction. Over the years, several types of chatbots have developed, distinguished by their specific forms and functions. A notable example that Rietz looked into is the frame-based bot, which employs question templates to communicate with users to retrieve as many information from the user as possible [Rie19].

In 2019, Arruda, Marinho, Souza, and Wanderley conducted a study on goal-oriented requirements engineering (GORE), focusing on the KAOS framework. Their research highlighted the advantages of using GORE methodologies to improve the elicitation, documentation, and modification of requirements. The key motivations for adopting GORE include a better understanding of client objectives, enhanced communication among stakeholders, improved traceability of software requirements, and effective conflict management among requirements [AMSW19]. Over the last years, Software vendors increasingly rely on user feedback to discover requirements. They can contain helpful information for the product team, collecting valuable, informative, and actionable feedback, but the user feedback is often vague, emotional, or is missing important information, such as contextual information [WFM22]. The KAOS framework, a prominent GORE approach, facilitates the development of comprehensive system models that extend beyond the software components. Despite its benefits, the complexity of goal modeling poses challenges for novice requirements engineers. To address these challenges, the study explored the application of natural language processing (NLP) techniques, particularly text mining, to streamline and automate requirements engineering tasks. As part of their work, Arruda et al. developed KAOSbot, a chatbot-based tool designed to support the KAOS framework. KAOSbot leverages NLP techniques to engage in conversations with users, facilitating the extraction of relevant information about specific domains.

This tool is instrumental in identifying goals, expectations, requirements, and agents, thereby enhancing the efficiency of the requirements elicitation process [AMSW19].
Silva and Canedo identified several key challenges that come with requirements engineering in their research. Based of the experience and opinions of 12 people, they note that aligning stakeholder expectations with chatbot functionality is difficult due to communication issues and evolving client needs. Also, requirements are often poorly defined and lack clarity, leading to rework and scope changes. Business management issues, such as delays and inconsistent client engagement, disrupt the process. Technological limitations, including missing APIs and inadequate tools, further complicate implementation. Additionally, creating a fitting context and designing characters for the chatbot poses significant challenges. These issues highlight the need for better methods and tools in chatbot development [SC20].

2.2 Voice opportunitities

As of September 27th, 2023, ChatGPT(-4.0) has introduced voice and image functionalities alongside its existing GPT-3.5 features [Ope23]. This expansion means that, in addition to leveraging Large Language Models (LLMs), ChatGPT will now offer capabilities such as voice recognition, speech synthesis, image analysis, and image generation. But will these functionalities also be efficient? Previous research has delved into the analysis of 37 empirical studies on speech-recognition chatbots and proposed a conceptual framework, known as the GEM- framework, which consists of three key components for classifying speech recognition chatbots— goal-orientation, embodiment, and multimodality. These studies have provided substantial evidence supporting the notion that employing and integrating chatbots into various applications results in more benefits [JLC23].

In another study is said that the integration of artificial intelligence (AI) chatbots with speech recognition technologies represents a significant advancement in conversational technology, offering unparalleled intelligence and adaptability [I+12]. According to Gobinath A. et al., their research highlights how this synergy enables sophisticated voice interactions, allowing users to engage in complex dialogues, seek clarifications, and receive highly personalized assistance, thus making interactions more intuitive and related to human conversation. This development is also crucial in enhancing accessibility for individuals with impairments, promoting greater inclusivity across various applications. By incorporating voice assistants with conversational AI, the quality of customer service and support is notably improved and the technology becomes more adept at understanding and responding to the subtleties of human speech [BM22]. The impact of this integration extends to multiple fields, reinforcing its role as a transformative tool in technological innovation and user interaction [ACJ+24].

The paper by S. J. du Preez, M. Lall, and S. Sinha looked into the optimal framework for developing a voice recognition chatbot. By their research the optimal framework consists of three main components: the front-end application, the decoder, and the language model. The front-end captures and processes voice input, while the language and acoustic models translate this input using a dictionary and a look-up table (LUT). The decoder's search manager then decodes the input into a comprehensible result set. Additionally, the front-end allows users to configure the chatbot by loading components stored in XML format. This setup provides a robust system for intelligent voice interaction within a web service context [PLS09]. The last few years, the use of

Voice Assistence (VA's) like Siri, Alexa, Google Assistant, Cortana, and Bixby are also increasingly becoming popular among the general mass. Due to this increase, there is significant pressure to integrate voice functionalities into other web applications and chatbots as well [PKJ22]. Pal et al. conducted research on the experiences of people using Voice Assistants (VAs). Their study focused on a large group of users, including both native English speakers and non-native speakers who use English as their primary mode of communication. The findings revealed that there were no significant differences in usability between these two user groups and they shared the same experience [PAV+19].

2.3 The Best Speech-to-Text & Text-to-Speech Engines

2.3.1 Speech-to-Text

In the research to identify the most effective Speech-to-Text (STT) engines, various well-performing sources were consulted. The findings are summarized as follows, with each source providing its top engine recommendations:

- **AssemblyAI.com**: AssemblyAI, Google Speech-to-Text, AWS Transcribe, DeepSpeech, Kaldi, and SpeechBrain [Ass24].
- **Notta blog**: Whisper, DeepSpeech, Kaldi, SpeechBrain, and Coqui [Not24].
- **Speechify**: Speechify, Amazon Polly, Google Cloud Text-to-Speech, Microsoft Azure, and IBM Watson Text-to-Speech [Spe24].
- **Gladia**: Whisper ASR, DeepSpeech, Wav2vec, Kaldi, and SpeechBrain [Gla24].
- **Deepgram**: Deepgram Speech-to-Text API, Whisper, Microsoft Azure, Google Speech-to-Text, and AssemblyAI [Dee24].
- **Rev**: Deepspeech, Wav2letter, and Kaldi [Rev24].
- **EdenAI**: Deepspeech, Kaldi, Wav2letter, SpeechBrain, Coqui, and Whisper [Ede24].
- **Fosspost.org**: Deepspeech, Kaldi, Julius, and Wav2letter [Fos24].
- **Blog.spheron.network**: Deepspeech, Kaldi, SpeechBrain, Coqui, Whisper, and Julius [Net24].

To identify which Speech-to-Text (STT) engines were most frequently mentioned, a list of the top five was created. The engines that appeared most often are:

1. **DeepSpeech** - Mentioned 9 times
2. **Kaldi** - Mentioned 9 times
3. **Whisper** - Mentioned 7 times
4. **SpeechBrain** - Mentioned 6 times

5. **Google Cloud STT** - Mentioned 3 times

Here's a breakdown of where these engines were mentioned:

- **DeepSpeech**: AssemblyAI, Notta blog, Gladia, Deepgram, Rev, EdenAI, Fosspost.org, blog.spheron.network [Ass24, Not24, Gla24, Dee24, Rev24, Ede24, Fos24, Net24].

- **Kaldi**: AssemblyAI, Notta blog, Gladia, Deepgram, Rev, EdenAI, Fosspost.org, blog.spheron.network [Ass24, Not24, Gla24, Dee24, Rev24, Ede24, Fos24, Net24].

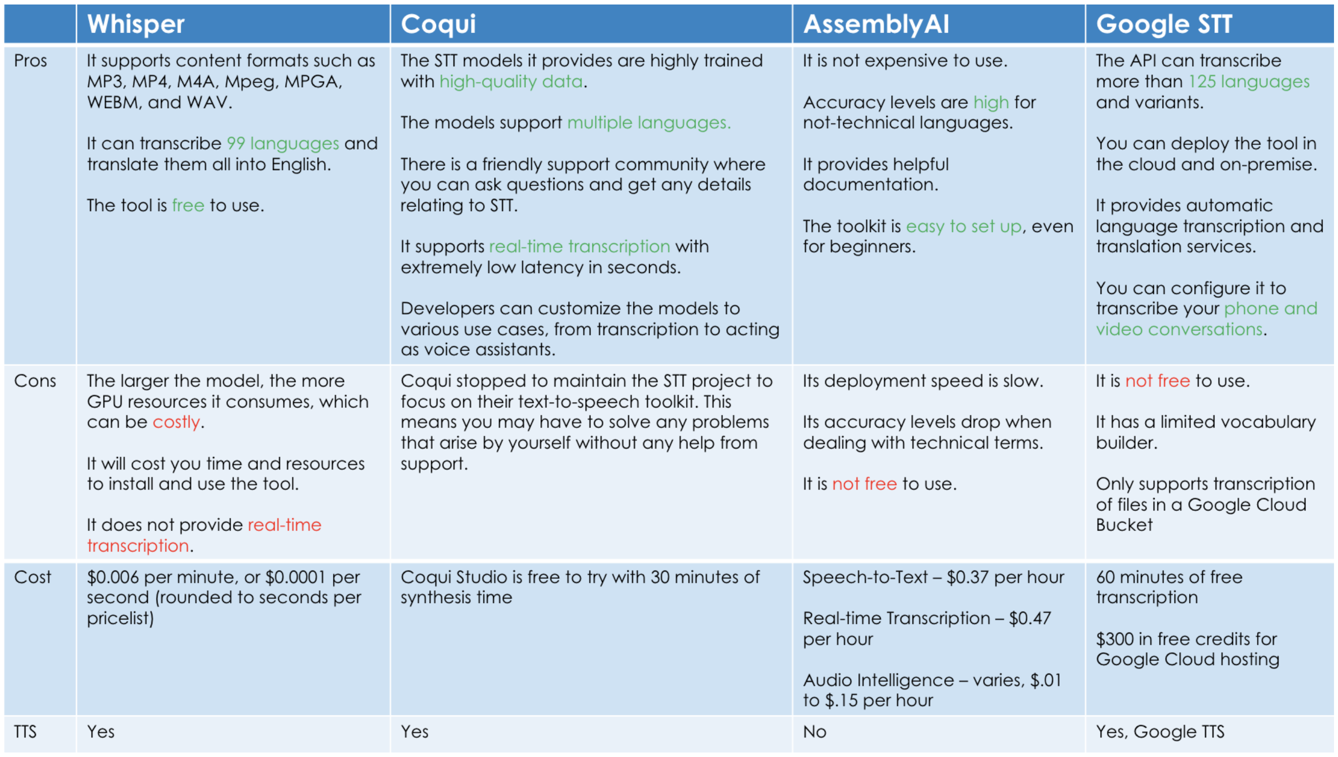
- **Whisper**: Notta blog, Gladia, Deepgram, EdenAI, blog.spheron.network [Not24, Gla24, Dee24, Ede24, Net24].

- **SpeechBrain**: AssemblyAI, Notta blog, Gladia, EdenAI, blog.spheron.network [Ass24, Not24, Gla24, Ede24, Net24].

- **Google Cloud STT**: AssemblyAI.com, Speechify, Deepgram[Ass24, Spe24, Dee24].

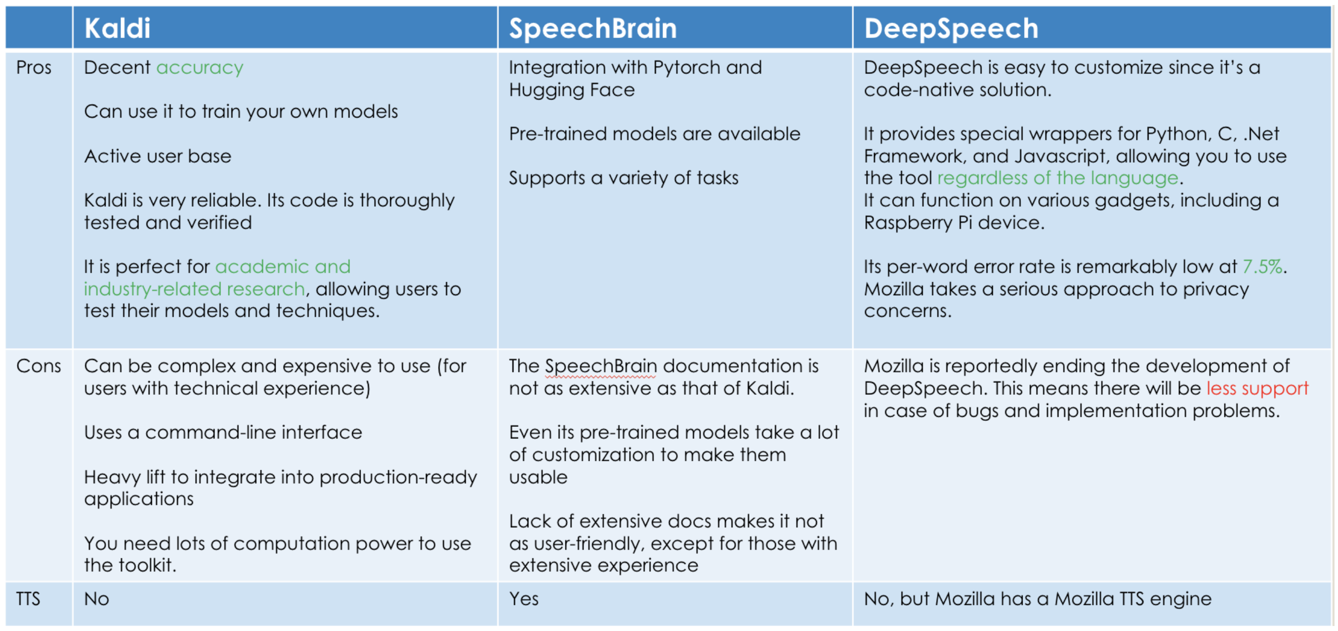
Given these findings, two tables were created to compare the advantages and disadvantages of the various engines. The first table includes four engines that require associated costs and the second table consists of three engines that are Open Source.

Figure 1: Comparison of Speech-to-Text Engines with Costs



	Whisper	Coqui	AssemblyAI	Google STT
Pros	It supports content formats such as MP3, MP4, M4A, Mpeg, MPGA, WEBM, and WAV. It can transcribe 99 languages and translate them all into English. The tool is free to use.	The STT models it provides are highly trained with high-quality data. The models support multiple languages. There is a friendly support community where you can ask questions and get any details relating to STT. It supports real-time transcription with extremely low latency in seconds. Developers can customize the models to various use cases, from transcription to acting as voice assistants.	It is not expensive to use. Accuracy levels are high for not-technical languages. It provides helpful documentation. The toolkit is easy to set up, even for beginners.	The API can transcribe more than 125 languages and variants. You can deploy the tool in the cloud and on-premise. It provides automatic language transcription and translation services. You can configure it to transcribe your phone and video conversations.
Cons	The larger the model, the more GPU resources it consumes, which can be costly. It will cost you time and resources to install and use the tool. It does not provide real-time transcription.	Coqui stopped to maintain the STT project to focus on their text-to-speech toolkit. This means you may have to solve any problems that arise by yourself without any help from support.	Its deployment speed is slow. Its accuracy levels drop when dealing with technical terms. It is not free to use.	It is not free to use. It has a limited vocabulary builder. Only supports transcription of files in a Google Cloud Bucket
Cost	$0.006 per minute, or $0.0001 per second (rounded to seconds per pricelist)	Coqui Studio is free to try with 30 minutes of synthesis time	Speech-to-Text – $0.37 per hour Real-time Transcription – $0.47 per hour Audio Intelligence – varies, $.01 to $.15 per hour	60 minutes of free transcription $300 in free credits for Google Cloud hosting
TTS	Yes	Yes	No	Yes, Google TTS

Figure 2: Comparison of Open Source Speech-to-Text Engines



	Kaldi	SpeechBrain	DeepSpeech
Pros	Decent accuracy Can use it to train your own models Active user base Kaldi is very reliable. Its code is thoroughly tested and verified It is perfect for academic and industry-related research, allowing users to test their models and techniques.	Integration with Pytorch and Hugging Face Pre-trained models are available Supports a variety of tasks	DeepSpeech is easy to customize since it's a code-native solution. It provides special wrappers for Python, C, .Net Framework, and Javascript, allowing you to use the tool regardless of the language. It can function on various gadgets, including a Raspberry Pi device. Its per-word error rate is remarkably low at 7.5%. Mozilla takes a serious approach to privacy concerns.
Cons	Can be complex and expensive to use (for users with technical experience) Uses a command-line interface Heavy lift to integrate into production-ready applications You need lots of computation power to use the toolkit.	The SpeechBrain documentation is not as extensive as that of Kaldi. Even its pre-trained models take a lot of customization to make them usable Lack of extensive docs makes it not as user-friendly, except for those with extensive experience	Mozilla is reportedly ending the development of DeepSpeech. This means there will be less support in case of bugs and implementation problems.
TTS	No	Yes	No, but Mozilla has a Mozilla TTS engine

Based on the evaluation criteria outlined in the tables, three Speech-to-Text (STT) engines were selected for further testing. From the first table, Whisper was chosen due to its inclusion of a Text-to-Speech (TTS) engine and its significantly lower cost compared to Google SST, which also offers TTS capabilities. Additionally, Coqui was excluded from consideration due to its discontinuation [Coq24], and AssemblyAI was not selected as it lacks TTS functionality [Ass24].
From the second table, DeepSpeech and SpeechBrain were selected. Kaldi was not chosen due to its complexity and challenges in integrating with existing applications, whereas ease of integration is a critical requirement for the project.

2.3.2 Text-to-Speech

Before initiating on the search for an optimal Text-to-Speech (TTS) model, existing Speech-to-Text (STT) models were initially investigated to determine if any integrated solutions were available that could handle both STT and TTS tasks. Such a dual-functionality model would have a preference. The models identified as potentially meeting this criterion included Whisper, Coqui, Google Cloud, and Speechbrain. Although DeepSpeech did not directly offer TTS capabilities, it is associated with Mozilla, which does provide a separate TTS model, Mozilla TTS. Thus, this model was also included for the selection.
An in-depth examination of each option revealed that Whisper does not directly offer TTS capabilities. While OpenAI provides a TTS solution, it is not available as open source [Ope24]. Coqui was excluded from consideration due to its discontinuation, as previously mentioned, and same went for DeepSpeech. This left only Google Cloud and Speechbrain as possible TTS Engines. During the implementation phase, significant difficulties were encountered with Speechbrain, leading to a complex implementation process. Because of this, Google Cloud TTS turned out to be the more practical and efficient choice.

3 Methodology

This section outlines the methodology employed to evaluate and compare the top three Speech-to-Text (STT) engines. The methodology provides a systematic approach for assessing the performance of each engine based on various criteria, including transcription speed and accuracy across different file formats and real-time scenarios.

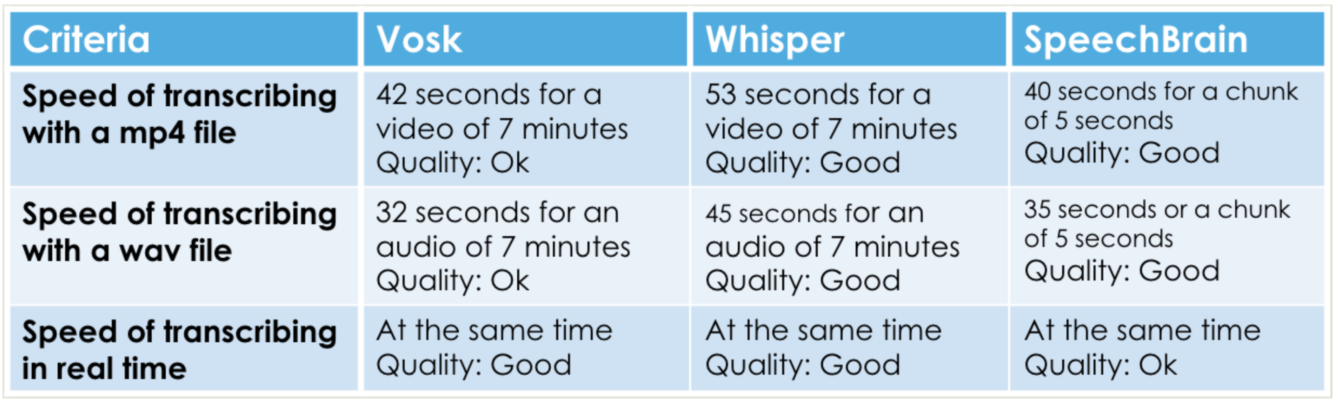
In this evaluation, an initial implementation phase revealed that DeepSpeech, one of the selected engines, was outdated and only functional with a deprecated version of Python. Consequently, DeepSpeech was substituted with Kalki, which is integrated into the Vosk framework. Therefore Vosk became the third model.

The comparative analysis focuses on several key performance metrics:

- Speed of transcription with MP4 files
- Speed of transcription with WAV files
- Speed of real-time transcription
- Accuracy of transcription with MP4 files
- Accuracy of transcription with WAV files
- Accuracy of real-time transcription

To facilitate this evaluation, a sample video was recorded, discussing various services. The performance of each engine was measured against these criteria, and the results are compiled into Figure 3. This table presents a clear comparison of the engines' effectiveness in different transcription scenarios, providing insights into their relative performance and suitability for various applications.

Figure 3: Comparison of Open Source Speech-to-Text Engines



Criteria	Vosk	Whisper	SpeechBrain
Speed of transcribing with a mp4 file	42 seconds for a video of 7 minutes Quality: Ok	53 seconds for a video of 7 minutes Quality: Good	40 seconds for a chunk of 5 seconds Quality: Good
Speed of transcribing with a wav file	32 seconds for an audio of 7 minutes Quality: Ok	45 seconds for an audio of 7 minutes Quality: Good	35 seconds or a chunk of 5 seconds Quality: Good
Speed of transcribing in real time	At the same time Quality: Good	At the same time Quality: Good	At the same time Quality: Ok

3.1 Data Gathering

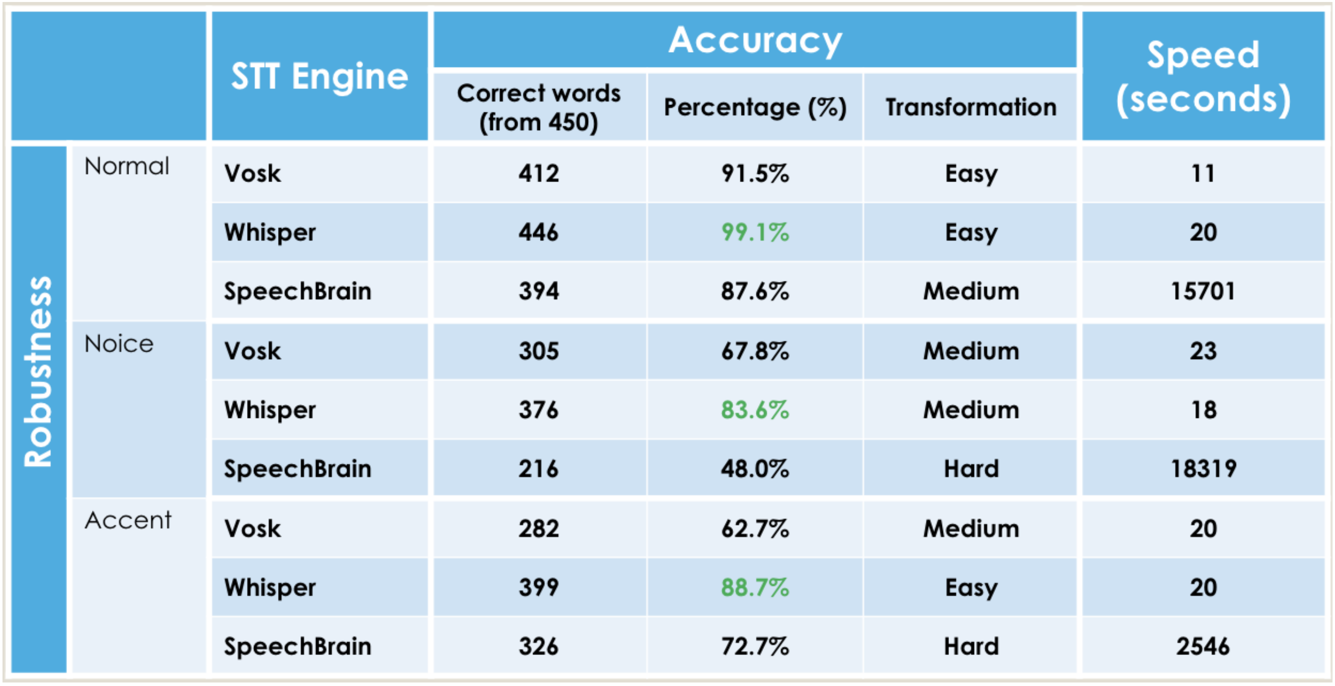
To effectively compare the Speech-to-Text engines based on the specified criteria, a video from YouTube featuring the launch of a new Microsoft service was selected to assess the performance of the engines on a specific and relevant topic. This comparison aimed to identify any significant differences in engine performance when transcribing topic-specific content versus the randomly selected video.

During the data gathering process, it was decided to introduce control variables into the study to enhance the accuracy of the evaluation. The original YouTube video, noted for its clarity and ease in sound, was modified to create two additional versions. The first version included background noise to simulate a crowded environment, and the second featured a robotic accent. These variations were introduced to test the engines' robustness and adaptability under different noice conditions.

3.2 Pre-selection

A comparative table was compiled to systematically evaluate the performance of each engine based on their performance with the three different audio files, as shown in Figure 4.

Figure 4: Comparison of Open Source Speech-to-Text Engines



		STT Engine	Accuracy			Speed (seconds)
			Correct words (from 450)	Percentage (%)	Transformation	
Robustness	Normal	Vosk	412	91.5%	Easy	11
		Whisper	446	99.1%	Easy	20
		SpeechBrain	394	87.6%	Medium	15701
	Noice	Vosk	305	67.8%	Medium	23
		Whisper	376	83.6%	Medium	18
		SpeechBrain	216	48.0%	Hard	18319
	Accent	Vosk	282	62.7%	Medium	20
		Whisper	399	88.7%	Easy	20
		SpeechBrain	326	72.7%	Hard	2546

The table above presents a detailed comparison of the performance of three Speech-to-Text engines—Whisper, SpeechBrain, and Vosk—across three different audio scenarios: a file with a standard accent, one with substantial background noise, and another featuring a robotic accent. The table provides comprehensive metrics, including the accuracy percentage, the number of correctly transcribed words, the extent of necessary corrections and transformations, and the time taken for each engine to complete the transcription.

The results indicate that Whisper consistently outperformed the other engines, achieving the highest levels of accuracy and the fastest transcription times. While both SpeechBrain and Vosk also delivered solid performances, Whisper's overall superiority is evident. It is important to highlight that SpeechBrain exhibited a notably extended transcription time compared to the other engines. This extended duration can likely be attributed to the inherent complexities associated with its implementation. The complex nature of SpeechBrain's architecture and processing algorithms appears to contribute to its slower performance, making it less efficient in terms of transcription speed. Based on this comparative analysis, Whisper has been selected for further development.

4 Architecture

One of the main objectives of this thesis is to build an API framework that can be used by AI4MDE for Speech-to-Text and Text-to-Speech transcriptions. Central to the transcription is the integration with the current AI4MDE-Studio program. In this chapter the flow that the system takes through the application is described, both flow based as technical, focusing on the elements and choices made during this process for a good-working and effective API.

4.1 Conceptual Architecture

To facilitate a highly interactive experience by integrating both Speech-to-Text (STT) and Text-to-Speech (TTS) functionalities, an Audio API is developed including both functionalities. The STT component transcribes spoken input from users into text, while the TTS component converts text responses from the chatbot into spoken output. This dual functionality aims to significantly enhance user interaction by providing both spoken input and output, thereby making the requirement elicitation process more engaging and user-friendly.
The core components of the API are as follows:

- STT Engine: The Whisper model is employed for converting speech into text.
- TTS Engine: The gTTS (Google Text-to-Speech) model is used to generate spoken responses from text.

The decision to utilize these API's to create an additional Audio API was motivated by the need to streamline and centralize access to both Speech-to-Text (STT) and Text-to-Speech (TTS) functionalities. By integrating these capabilities into a single, unified API endpoint, we provide AI4MDE developers with a single access point for both services, thereby simplifying their development process and reducing the complexity of managing multiple APIs. This consolidated approach not only enhances user experience by offering a seamless integration but also future-proofs the system. It enables us to easily switch between different underlying engines or update the technology stack as needed. Lastly, it also allows to add triggering words for the STT or TTS functionality in a single location, eliminating the need to program complex code each time changes are made to the tool. This flexibility ensures that the system can adapt to advancements in technology or changes in requirements, maintaining a consistent and reliable interface for developers, thus wrapping the API to simplify integration.

4.1.1 User Interaction Flow

The interaction with the AI4MDE-Studio program begins with users initiating a voice input by clicking on the microphone icon on their device. This action activates the device's microphone, allowing it to record the user's spoken input. Once recorded, the audio is transmitted to the API, where it undergoes transcription into text via the Whisper model. This transcription process converts the spoken words into written text, which is then sent back to the text area for display to the user.

For sending the transcription to the text area, there are two options. Displaying the transcription instantly as the user speaks (continuous form) or showing the transcription in batches after the user has finished recording. Displaying the transcription continuously provides immediate feedback, which can enhance the user experience by allowing real-time interaction and quicker adjustments to spoken input. This method is particularly useful in dynamic conversation scenarios where immediate text representation helps users to correct or refine their speech on the fly. However, continuous transcription can also be challenging, as it requires consistent processing and may result in interruptions in the transcription, delays if the system struggles with processing speed or accuracy in noisy environments, and distraction when the user starts reading their own transcription instead of focusing on what they want to say.

Whereas, batch processing, where the transcription is displayed only after the user has completed their recording, can be efficient for handling larger volumes of text. It allows for a more streamlined process without the need for constant updates, which can be beneficial in environments with less frequent or less interactive speech inputs. However, this method might delay feedback and reduce interactivity, as users only see the final transcription after completing their recording. If users are dissatisfied with the transcription once it is presented, they may find it frustrating, especially if they are unable to make immediate corrections or adjustments after recording a long answer. Each approach has its advantages and disadvantages, and the choice between them depends on the specific requirements of the application and the context in which it is used. For this research the continuous form is preferred in the extended integration because of the user friendliness it will bring to the chatbot, and the batch processing is used in the basic integration, as this integration is less interactive and the batch processing is efficient for handling larger volumes of text.

4.1.2 Basic integration vs Extended integration

The basic integration does not support autonomous response generation. However, research is ongoing to incorporate this feature. In the extended integration, the transcribed text from the Speech-to-Text (STT) model will be forwarded to a AI4MDE Chatbot, which will leverage a Large Language Model (LLM) to generate relevant and contextual responses.

This process will also include generating follow-up questions based on the ongoing conversation script. By following the generation of a text-based response, the text will be sent back to the API to be transformed into spoken output using the Text-to-Speech (TTS) model. The resulting speech will be transmitted to the front-end, where it is played through the user's speakers, completing the communication loop. To illustrate the process, two visualizations have been created. The first Flow Chart (Figure 5) illustrates the basic integration, which does not include the Text-to-Speech (TTS) functionality. The second Flow Chart (Figure 6) shows the extended integration, incorporating both the Chatbot framework and the TTS implementation.

To ensure a cohesive and smooth user experience, the API is intricately integrated into the back-end of the AI4MDE system. This integration manages all API calls and coordinates the interactions between the chatbot and the user, thereby enabling a seamless flow of information and ensuring effective communication throughout the entire interaction process.

Figure 5: Basic integration Requirement Elicitation with Speech function

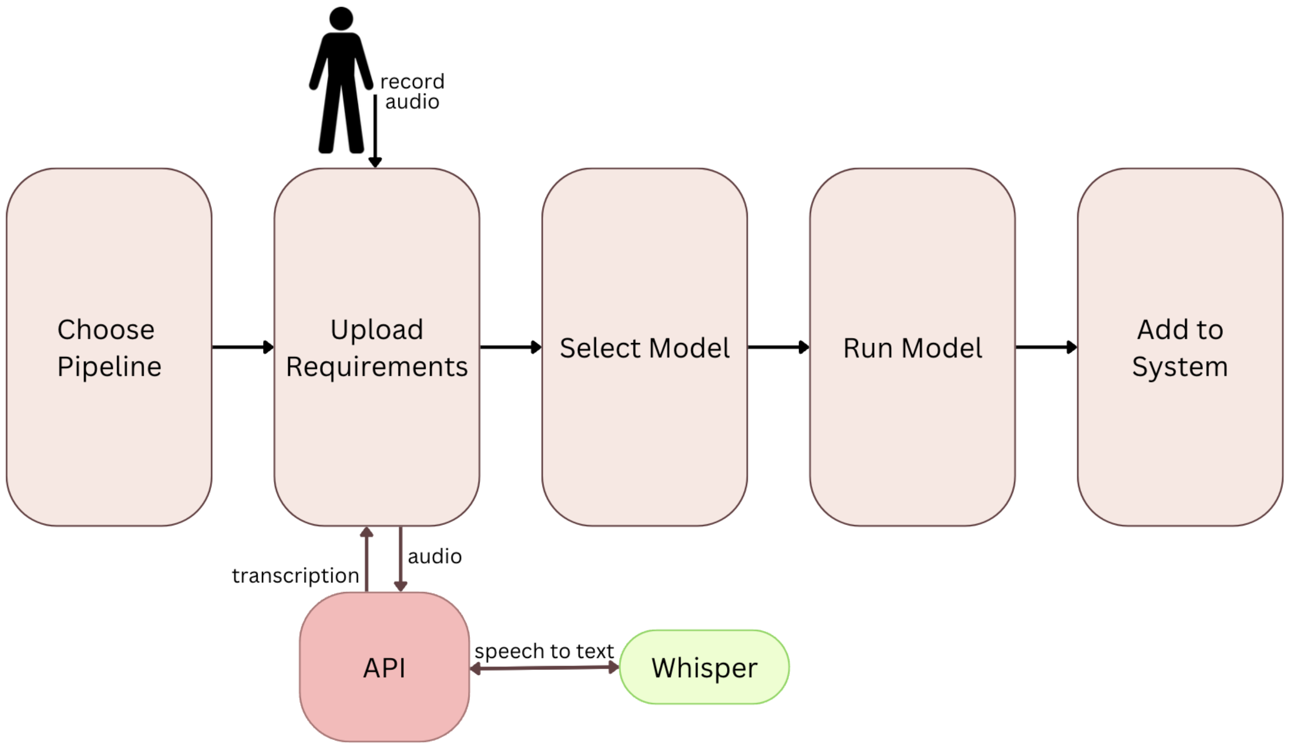
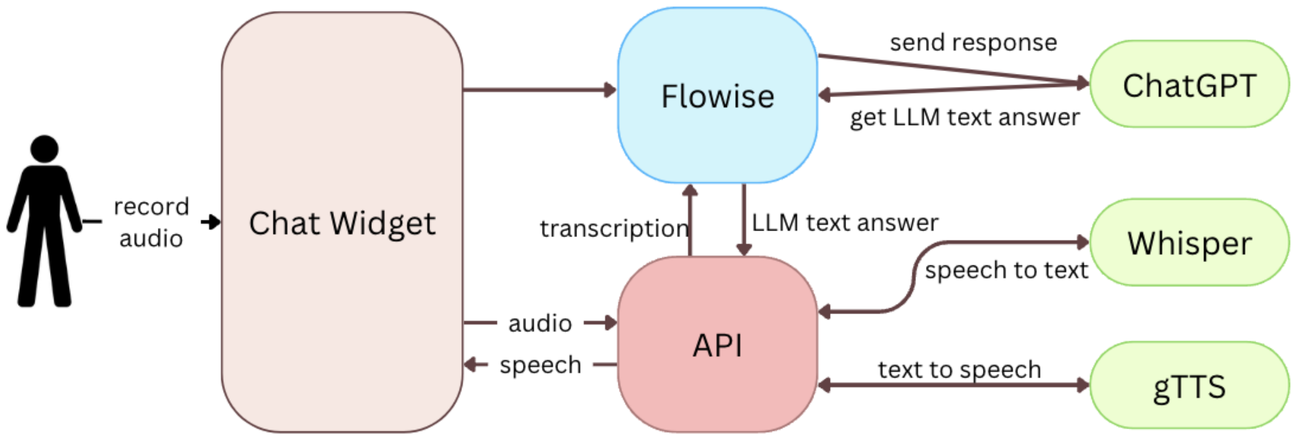


Figure 6: Extended integration Requirement Elicitation with Speech and Chat function

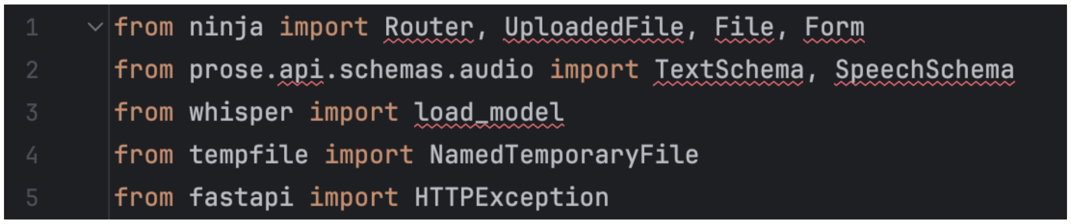


4.2 Technical Architecture

This section delves into the technical foundations of the API framework, detailing the design and implementation of the Speech-to-Text (STT) and Text-to-Speech (TTS) endpoints. The architecture is constructed to ensure seamless interaction between the user and the chatbot, enabling efficient processing of voice inputs and outputs. By examining the code snippets and logic behind these endpoints, we will explore how the API handles speech recognition and synthesis. The discussion will also cover the choices made for various components, highlighting their roles within the overall system architecture.

4.2.1 Framework

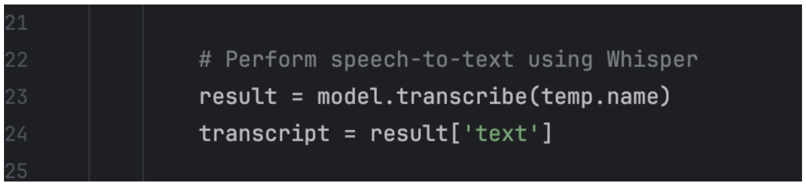
To implement the Speech-to-Text (STT) functionality, the Ninja framework was utilized, along with the Whisper API for transcribing audio. The code starts by importing essential libraries, including Ninja for building the API and Whisper for the transcription model, see Figure 7. The Whisper model is then loaded using the load_model() function, initializing it for use in the application. The core of the STT functionality is encapsulated in an API endpoint defined with @audio.post("/stt"). This endpoint is designed to accept audio files from user recordings. If no file is uploaded, the API responds with an error message. For each uploaded file, a temporary file is created to store the audio data.



```python
from ninja import Router, UploadedFile, File, Form
from prose.api.schemas.audio import TextSchema, SpeechSchema
from whisper import load_model
from tempfile import NamedTemporaryFile
from fastapi import HTTPException
```

Figure 7: Used models and engines for the Speech-to-Text API

The audio data is then passed to the Whisper model for transcription. The model.transcribe() function processes the temporary audio file, converting the spoken words into text, see Figure 8. Finally, the API returns the transcribed text as a JSON response using the TextSchemas, with the transcription stored in a variable named transcript.



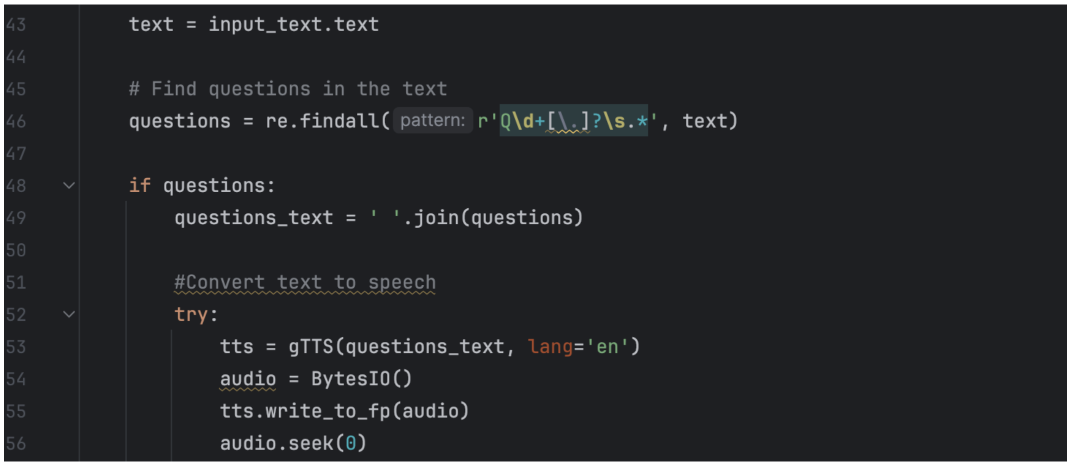
```python

        # Perform speech-to-text using Whisper
        result = model.transcribe(temp.name)
        transcript = result['text']

```

Figure 8: Speech-to-Text API implementation

The technical architecture of the API to handle Text-to-Speech (TTS) functionality also employs the Ninja framework, ensuring efficient and effective text-to-audio conversion. Ninja's high-performance capabilities and modern design facilitate the seamless integration of TTS processes into the framework. This section outlines the TTS endpoint implementation, highlighting key aspects of the code.
The /tts endpoint is carefully designed to handle text input and convert it into speech. This functionality is a key feature of the application, designed to enhance user interaction by providing spoken responses. The core implementation of this endpoint is outlined in Figure 9, which showcases the fundamental code structure and flow.



```
text = input_text.text

# Find questions in the text
questions = re.findall( pattern: r'Q\d+[\.]?\s.*', text)

if questions:
    questions_text = ' '.join(questions)

    #Convert text to speech
    try:
        tts = gTTS(questions_text, lang='en')
        audio = BytesIO()
        tts.write_to_fp(audio)
        audio.seek(0)
```

Figure 9: Text-to-Speech API implementation

The endpoint is defined to handle incoming POST requests that carry text data. Upon receiving a request, the endpoint processes the input to extract specific text segments based on a trigger word pattern. This pattern, represented by a regular expression, is designed to identify and isolate segments of questions asked by the AI4MDE chatbot marked by a "Qx" format, where 'x' stands for any number greater than or equal to 1. The reason only the questions are sent for speech formation, is to maintain a chat-like interaction while incorporating complex tabular and/or visual information. Converting the entire content to speech would be time-consuming and inefficient. The regular expression starts with analysing the input text to find relevant questions, ensuring that only the portions of text that match this pattern are selected for further processing. This method ensures that speech generation focuses on contextually important information.

The extracted text segments are then passed to the gTTS (Google Text-to-Speech) library, which is utilized to convert text into natural-sounding speech. The gTTS object processes the input text, synthesizing it into speech and storing the output in an in-memory byte stream. This approach enables efficient handling of the audio data without the need for intermediate file storage. Once the speech synthesis is complete, the generated audio is prepared for streaming.

4.2.2 Data Flow

The Voice Input Processing stage initiates the interaction by capturing the user's voice through a microphone. This voice input is recorded as a .wav file, a standard format that preserves the quality and integrity of the audio. Once the audio is recorded, it is sent to the API as 'input_audio', which then transmits it to the Speech-to-Text (STT) engine as 'temp.name'. In this case, the STT engine used is the Whisper model. Whisper thoroughly processes the entire audio file by analyzing the spoken words and transcribing them into text. It then sends the transcribed text back to the API in the variable 'result'. The API then returns the transcription of the audio, which is stored in a variable named 'transcript'. This is how the Speech to Text functionality works in the basic integration, as shown in Figure 10.

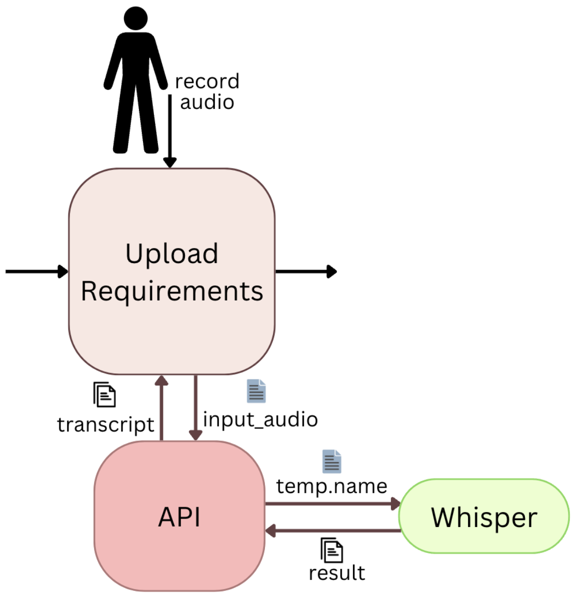


Figure 10: Data Flow of the STT Functionality in the Basic Integration

In the extended integration, after receiving the transcription, the Text Processing and Response Generation phase begins. The transcribed text is passed to the back-end, which utilizes sophisticated Natural Language Processing (NLP) techniques to interpret the user's input. This involves analyzing the text, understanding its context, and determining the most appropriate response. The chatbot's Large Language Model (LLM) plays a central role here, generating responses that are contextually relevant and coherent. These generated responses are then forwarded to an API, which further processes the response for the Text to Speech functionality.

Upon arrival of the response as 'input_text', the API sends the corresponding sentences to the Text-to-Speech (TTS) engine, specifically gTTS, as variable 'text'. The API is designed to recognize specific questions within the generated text, particularly those marked with a "Qx" format (where 'x' denotes the question number greater and equal to one).

After identifying the question in the variable 'questions', the gTTS engine converts only the portion of the text that starts with "Qx" into speech, while the remainder of the text is excluded from the audio file. The API then generates a .wav file containing the spoken version of the chatbot's response and saves it in 'metadata' with the filename within the header.

Finally, during the Voice Output Processing stage, the .wav file generated by the TTS engine is sent to the frontend of the system. This frontend component is responsible for delivering the audio output to the user's device, where it is played through the speakers. This completes the interaction loop, allowing the user to hear the chatbot's response in a natural, spoken format. The integration of these processes ensures a smooth and dynamic user experience, where voice input is seamlessly converted into text, processed, and then transformed back into speech for a truly interactive and engaging conversation.

5 Validation

In addition to the framework's architecture, its robustness and credibility are equally important. This section highlights the validation process, ensuring that the methodologies employed for assessing the Whisper model's transcription accuracy are both reliable and relevant. This involves defining clear criteria for evaluation, ensuring that the data used is representative of real-world conditions, and confirming that the results accurately reflect the model's performance across different scenarios.

In this context, validation encompasses several key aspects: the selection of diverse and representative participants, the consistency of recording conditions, and the precise benchmarking of transcription accuracy. By addressing these elements, the goal is to provide a comprehensive and unbiased assessment of the Whisper model's capabilities and limitations.
Following this validation discussion, the experimental setup will be presented, detailing how data was collected and processed, and describing the diversity of participants involved in the study. This will be followed by a thorough benchmarking analysis to evaluate the Whisper model's performance based on the collected data.

5.1 Experimental set-up

In this study, we aimed to assess the performance of the Whisper model in transcribing voice messages recorded by individuals with diverse backgrounds. To achieve this, participants were instructed to record short voice messages in which they introduced themselves, described their roles, and explained their requirements for elicitation.

Candidates are asked to discuss these topics because the initial questions posed by the extended integration of the AI4MDE chatbot will be: ”Would you like to introduce yourself and your role within the organization?”, following with ”Could you provide a brief summary of the project and its main goal?” The most effective way to ensure that the engine can accurately understand the responses to these questions is to test these questions. This is particularly important since these are the first two questions asked, and it is crucial for the model to clearly understand their answers to prevent any miscommunication throughout the conversation.

Participants were given specific guidelines to ensure consistency, each voice message should be approximately 30-40 seconds long and include a brief personal introduction, role description, and explanation of their project. Participants recorded their messages in various environments, including both quiet rooms and settings with background noise. The study involved a diverse group of participants, varying in age, gender, professional background, and accents. This diversity was intentionally included to evaluate the Whisper model's ability to handle different accents and speaking styles. Voice messages were collected and securely stored, to protect participant privacy. Each recording was then prepared for transcription by ensuring consistent audio quality and format. Following collection, the voice messages were then input into the Whisper model for transcription.

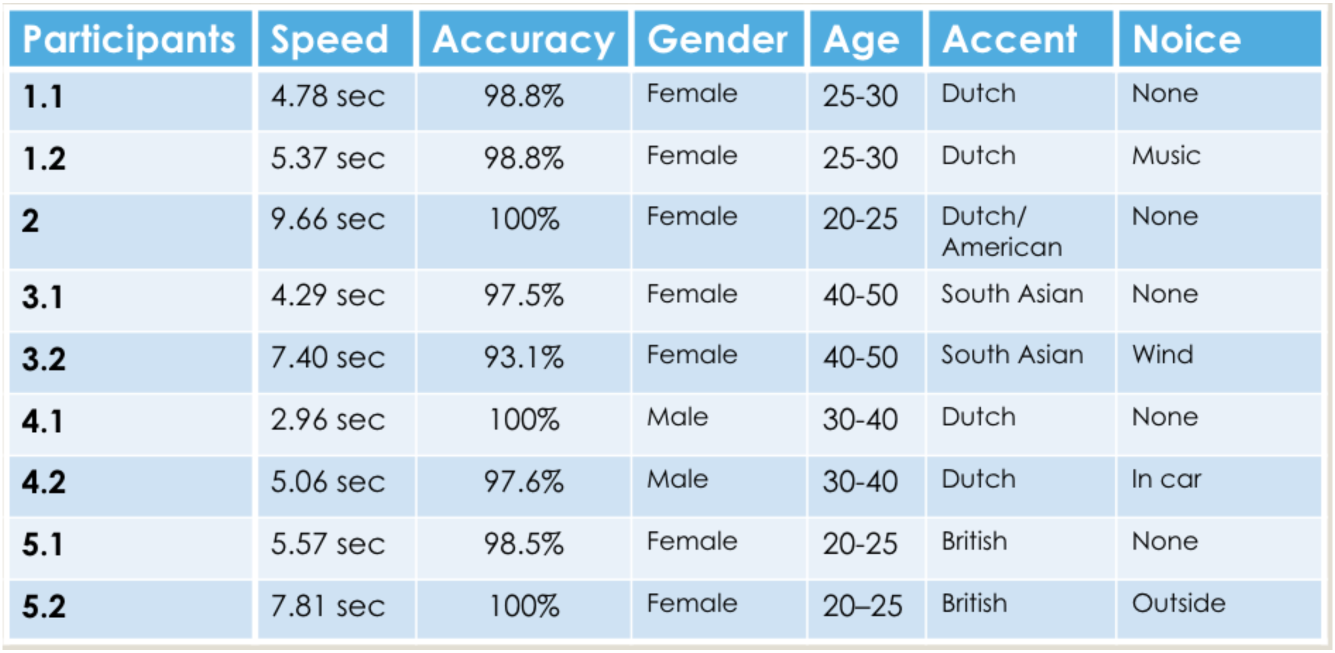
5.2 Candidates

The participants were selected to represent a broad range of demographics, including various accents, regional dialects, and professional fields. This diversity was crucial for evaluating the Whisper model's accuracy across different speech patterns and backgrounds. Participants were asked to describe their roles and the context. Their professional backgrounds varied from IT professionals to academic researchers, which added to the richness of the data and the challenge for the transcription model. The recordings were made under varying conditions, from controlled environments to more casual settings. This variability was important for testing the model's robustness and accuracy in real-world scenarios.

5.3 Benchmark

To evaluate the Whisper model's transcription accuracy, the output of the model was compared with manually transcribed reference texts. Accuracy was measured using metrics such as Word Error Rate (WER) and sentence-level accuracy. The transcription results were systematically compared to the manually transcribed text. Inconsistencies were analyzed to determine patterns, such as particular accents or recording conditions, that might affect performance. Cases in which the Whisper model struggled with were documented, such as inaccuracies in transcribing certain accents or background noise interference. These instances were examined to identify potential limitations of the model. Results are presented in the table below, showing accuracy metrics across different participant accents and recording conditions.

Figure 11: Whisper Transcription results of participants



Participants	Speed	Accuracy	Gender	Age	Accent	Noice
1.1	4.78 sec	98.8%	Female	25-30	Dutch	None
1.2	5.37 sec	98.8%	Female	25-30	Dutch	Music
2	9.66 sec	100%	Female	20-25	Dutch/ American	None
3.1	4.29 sec	97.5%	Female	40-50	South Asian	None
3.2	7.40 sec	93.1%	Female	40-50	South Asian	Wind
4.1	2.96 sec	100%	Male	30-40	Dutch	None
4.2	5.06 sec	97.6%	Male	30-40	Dutch	In car
5.1	5.57 sec	98.5%	Female	20-25	British	None
5.2	7.81 sec	100%	Female	20–25	British	Outside

5.4 Discussion of the Results

The table presents data from five participants, each of whom submitted two audio recordings for evaluation with answers to two questions, resulting in each test case being labeled with a number in the format x.y. Here, x denotes the participant number (ranging from 1 to 5), and y indicates whether it was their first or second audio submission. Notably, Participant 2 deviated slightly by submitting a single audio file containing responses to both questions.

Several observations can be drawn from the data presented in the table. First, it is evident that the model performs exceptionally well in scenarios where there is no background noise, with accuracy percentages reaching 100% accuracy scores. This outcome is logical, as the absence of noise disruption allows the model to focus solely on the spoken words, thereby enhancing its ability to accurately transcribe the audio.

When examining accent recognition, the model shows a clear expertise in understanding Dutch/ American, and British accents. In contrast, it struggles somewhat with Dutch accents and South Asian accents, with an exception on participant 4 who has a Dutch accent, and an accuracy of 100% without background noice. The lowest accuracy rate, 93.1%, was observed in a test case involving a participant with a South Asian accent accompanied by wind noise. This specific case's lower accuracy can be partly explained by the participant's frequent use of work-related abbreviations, such as "FFA" which may not be well-represented in the model's training data.

Another noteworthy observation is the model's ability to automatically cleanse transcriptions by omitting filler words like "uh" and repeated words. This feature is particularly advantageous for the requirement elicitation process, as it ensures that the transcribed text is clear and concise, without unnecessary verbal clutter that can detract from the quality of the data collected.

In terms of transcription speed, there is a noticeable variation, with the fastest transcriptions being completed in approximately 3 seconds and the slowest taking up to 10 seconds. This variation is primarily influenced by the number of words spoken during the recordings. Although participants generally spoke for 30-40 seconds, those who used a greater number of words experienced longer transcription times. This suggests that the model's processing time increases in direct proportion to the complexity and length of the spoken input.

Lastly, it is important to highlight that the model's performance was consistent across different age groups and genders. This indicates that the model does not show any inherent biases related to these demographic factors, validating its robustness and reliability in diverse user scenarios.

6 Discussion and Limitation

In this section, the findings from the evaluation of the Text-to-Speech (TTS) and Speech-to-Text (STT) models are analyzed, interpreted, and contextualized. This discussion aims to provide insights into how the results reflect the effectiveness of different models and their implications for practical applications. Additionally, it identifies potential avenues for future research and highlights any limitations that may affect the validity of the findings.

6.1 Interpretation of Results

The development and integration of the API for the AI4MDE system highlighted Whisper and gTTS as the most effective models for Speech-to-Text (STT) and Text-to-Speech (TTS) functionalities, respectively. Whisper was chosen for its exceptional accuracy and robust performance, especially in handling various accents and background noise, outperforming alternatives like SpeechBrain and Vosk. gTTS was selected for its ability to generate clear and natural-sounding speech output efficiently.

Integrating both STT and TTS models into a single API provides a centralized and flexible interface, streamlining the transition between speech and text processing tasks. This design not only simplifies the system's operation but also enhances its maintainability and scalability. Future upgrades or replacements of the STT or TTS engines can be easily managed within the API without requiring significant changes to the core program. This approach ensures continuous improvement in performance and functionality, making the system adaptable to evolving technological advancements and user needs.

6.2 Future Work

Future research should first evaluate the performance of the API's integration with the AI4MDE Chatbot after its launch. This assessment will provide valuable insights into how well the Audio API functions in real-world scenarios and its effectiveness in enhancing the chatbot's capabilities. Following this, exploring the integration of OpenAI's TTS model is recommended to potentially enhance the quality of the generated speech outputs.

Additionally, the exploration of new Speech-to-Text (STT) models, such as aiOla, should be considered. Given aiOla's reported 50% faster performance compared to Whisper, its effectiveness and suitability in various contexts warrant thorough investigation. And lastly, expanding the model to support multiple languages and automatically transcribing requirements to English would significantly improve the efficiency of the requirements elicitation process through speech. These steps will help refine the technology and enhance its applicability in diverse scenarios.

6.3 Threats to Validity

The validity of the research findings may be influenced by the experimental setup, which involved running models through Python in PyCharm on a local computer and using m4a and mp3 file formats. Variations in file types or computing environments could impact model performance,

suggesting that the results might differ with alternative configurations or formats. Future studies should consider testing the models under different conditions to validate their robustness.
The implementation of SpeechBrain involved using its local GitHub repository, which may not represent the most efficient approach. The complexity of SpeechBrain's implementation may have affected its speed and performance. Investigating alternative methods for implementing SpeechBrain could provide insights into improving its efficiency and overall functionality.

7 Conclusion

The integration of Large Language Models (LLMs) and speech functions into requirements elicitation software offers a promising avenue for enhancing both the efficiency and user experience of such tools. This conclusion provides an overview of how these technologies can be seamlessly integrated into the software, the perceived benefits and challenges, and suggestions for future research to further refine these innovations.

7.1 Integration Speech Functionalities

Integrating Speech Functionalities into requirements elicitation software involves employing models that can convert spoken language into text (speech-to-text, STT) and generate spoken responses from text (text-to-speech, TTS). This integration enhances user interaction by enabling more natural and intuitive communication. In the study, Whisper emerged as the most effective Speech-to-Text model, noted for its accuracy and ability to manage various accents and background noise. For the Text-to-speech functionality, gTTS was selected due to its superior implementation results and natural-sounding speech output. All of these functionalities are centralized within a unified Audio API, streamlining the process and providing a seamless interface for managing both STT and TTS operations.

7.2 Research Objectives and Findings

To address the research objectives, several aspects of integrating AI-driven speech functionalities into requirements elicitation software were explored. The first objective, *"What are the best practices for designing an API within the requirements elicitation software to communicate with AI-based speech functionalities?"* was addressed in Chapter 4.1. It was found that creating a centralized Audio API to integrate Speech-to-Text (STT) and Text-to-Speech (TTS) functionalities streamlined access and simplified development. This approach reduces complexity, allows for future flexibility by making it easier to update or replace underlying engines without significant changes to the core system, gives the possibilities to add triggering words for both the STT and TTS functionality, and can be integrated with all other AI4MDE APIs.

The second objective, *"What are the advantages and limitations of continuous flow versus batch processing for speech functionalities?"* is discussed in Chapter 4.1. where is said that continuous flow provides real-time transcription, which can enhance immediate interaction and allows for quick adjustments to spoken input. However, it can face challenges such as potential interruptions, processing delays, and user distraction if they focus on reading their own transcription. On the other hand, batch processing, which displays the transcription only after the user has finished recording, is efficient for managing larger volumes of text and avoids the need for constant updates. However, it may delay feedback and reduce interactivity, potentially frustrating users if they are unable to make immediate corrections or adjustments after recording a lengthy response.

For the third objective, *"How can AI-Driven Speech Functionalities effectively be used in a requirement elicitation process where chat-like interaction needs to be combined with complex tabular and/or visual information?"* was addressed in Chapter 4.2. The API was designed to send specific questions marked with a "Qx" format to the gTTS engine for text-to-speech conversion, maintaining a chat-like interaction while handling complex content efficiently. This approach ensures that the system remains adaptable and effective in integrating speech with detailed information, and it selectively provides responses that are most beneficial for the user to hear.

Finally, the fourth objective, *"How can the accuracy of speech models be effectively measured and which metrics are most relevant for assessing performance?"* was examined in Chapter 5.3. The Whisper model's accuracy was evaluated by comparing its transcription output to the actual words that have been said using metrics such as Word Error Rate (WER) and sentence-level accuracy. Discrepancies related to accents and recording conditions were analyzed to identify performance limitations.

7.3 Benefits and Challenges

Integrating Speech-to-Text (STT) and Text-to-Speech (TTS) models into requirements elicitation software offers several notable benefits. Primarily, the integration enhances the user experience by enabling speech-based interaction, which makes the elicitation process more engaging and accessible, particularly for users who may find typing inconvenient. Additionally, the use of gTTS for text-to-speech conversion provides clear and rapid speech output, thereby improving the overall efficiency of the system.

However, there are also significant challenges associated with this integration. One major challenge is language limitations, while Whisper performs well with familiar English accents, its accuracy may diminish with less common accents or in noisy environments, and it is limited to understanding only English. Another challenge is related to the extensive text responses generated by large language models (LLMs) such as ChatGPT. Transcribing entire responses into speech is often impractical, this means that only part of the response is spoken aloud, while the rest has to be read by the user.

By addressing these considerations and focusing on the proposed research areas, future work can further refine and enhance the integration of LLMs and speech functions in requirements elicitation software.

References

[ACJ+24] Gobinath A., Manjula Devi C., Suthan Raja S. J., Prakash P., Anandan M., and Srinivasan A. Voice assistant with ai chat integration using openai. In *2024 Third International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*, pages 1–6, 2024.

[AMSW19] Danilo Arruda, Matheus Marinho, Eric Souza, and Fernando Wanderley. A chatbot for goal-oriented requirements modeling. In Beniamino Murgante Elena Stankova Vladimir Korkhov Carmelo Torre Ana Maria A. C. Rocha David Taniar Bernady O. Apduhan Eufemia Tarantino Sanjay Misra, Osvaldo Gervasi, editor, *Computational Science and Its Applications – ICCSA 2019*, volume 11622, pages 533–536. Springer, 2019.

[Ass24] AssemblyAI. Deepspeech for dummies: A tutorial and overview (part 1), 2024.

[Bha23] Niharika Bhandari. An interactive chatbot for software requirements elicitation, January 27 2023. Leiden University.

[BM22] Dimitrios Buhalis and Iuliia Moldavska. Voice assistants in hospitality: using artificial intelligence for customer service. *Journal of Hospitality and Tourism Technology*, 13(3):386–403, 2022.

[Coq24] Coqui. Coqui: Speech-to-text and text-to-speech, 2024.

[Dee24] Deepgram. Best speech-to-text apis, 2024.

[Ede24] EdenAI. Top free speech-to-text tools apis and open source models, 2024.

[Fos24] FossPost. Open source speech recognition, 2024.

[Gla24] Gladia. Best open source speech-to-text models, 2024.

[I+12] Shinya Iizuka et al. Speech recognition technology and applications for improving terminal functionality and service usability. *NTT DOCOMO Technical Journal*, 13(4):79–84, 2012.

[JLC23] J. H. Jeon, S. Lee, and H. Choe. Beyond chatgpt: A conceptual framework and systematic review of speech-recognition chatbots for language learning. *Computers & Education*, 206:104898, December 2023.

[MS19] Alexander Maedche and Rainer Schmidt. Ladderbot: A requirements self-elicitation system. *ResearchGate*, 2019.

[Net24] Spheron Network. A comprehensive look at open source speech-to-text projects 2024, 2024.

[Not24] Notta. Speech-to-text open source, 2024.

[Ope23] OpenAI. Chatgpt can now see, hear, and speak, September 25 2023.

[Ope24] OpenAI. Text-to-speech overview, 2024. Accessed: 2024-08-19.

[PAV$^+$19] Debajyoti Pal, Shashank Arpitha, Rajarshi Vanjara, Mahak Basak, Yadnya Sarode, and Ajita Sinha. User experience with smart voice assistants: The accent perspective. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, 2019.

[PKJ22] PV Krishna Vamsi Prasad, N. Vamsi Krishna, and T. Prem Jacob. AI chatbot using web speech API and Node.js. In *2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*. IEEE, 2022.

[PLS09] Salomon Jakobus Du Preez, Manoj Lall, and Saurabh Sinha. An intelligent web-based voice chat bot. In *IEEE EUROCON 2009*, pages 386–391. IEEE, 2009.

[Rev24] Rev. The 5 best open source speech recognition engines apis, 2024.

[Rie19] Tim Rietz. Designing a conversational requirements elicitation system for end-users. In Sanjay Misra, Osvaldo Gervasi, Beniamino Murgante, Elena Stankova, Vladimir Korkhov, Carmelo Torre, Ana Maria A. C. Rocha, David Taniar, Bernady O. Apduhan, and Eufemia Tarantino, editors, *Computational Science and Its Applications – ICCSA 2019*, pages xx–xx, Saint Petersburg, Russia, 2019. Springer.

[SC20] Geovana R. S. Silva and Edna Dias Canedo. Requirements engineering challenges and techniques in building chatbots. *Department of Computer Science, University of Brasília (UnB)*, 2020.

[Spe24] Speechify. Best text-to-speech apis, 2024.

[WFM22] R. Wolfinger, F. Fotrousi, and W. Maalej. A chatbot for the elicitation of contextual information from user feedback. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*, pages 272–273, 2022.