A

major project report

on

A Neural Network-Based Voice Dialogue System for Email Management

submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology

By

G. Srilatha

(20EG105647)

K. Nithin Reddy

(20EG105653)

Under the guidance of

Mr. B. V. Srikanth

Assistant Professor

Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANURAG UNIVERSITY VENKATAPUR (V), GHATKESAR (M), MEDCHAL (D)-500088

TELANGANA

Year 2023-2024

DECLARATION

We hereby declare that the Report entitled "**A Neural Network-Based Voice Dialogue System for Email Management"** submitted to the **Anurag University** in partial fulfilment of the requirements for the award of **Bachelor of Technology (B.Tech)** in **Computer Science and Engineering** is a record of original work done by us under the guidance of **B. V. Srikanth, Assistant Professor** and this report has not been submitted to any other University or Institution for the award of any degree or diploma.

Place: Anurag University, Hyderabad
Date: 20-04-2024

G. Srilatha (20eg105647)
K. Nithin Reddy (20eg105653)

CERTIFICATE

This is to certify that the Report entitled "**A Neural Network-Based Voice Dialogue System**" for Email Management that is being submitted by **Ms. G. Srilatha** bearing Hall Ticket number **20EG105647** and **Mr. K. Nithin Reddy** bearing Hall Ticket number **20EG105653** in partial fulfilment of the requirements for the award of **Bachelor of Technology** in **Computer Science and Engineering** to the **Anurag University** is a record of bonafide work carried out by them under my guidance and supervision from 2023 to 2024.

The results presented in this report have been verified and found to be satisfactory. The results embodied in this Report have not been submitted to any other University or Institute for the award of any degree or diploma.

Signature of supervisor
Mr. B. V. Srikanth
Assistant Professor, CSE

Signature of Dean
Dr. G. Vishnu Murthy
Dean, CSE

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to the project supervisor Mr. **B. V. Srikanth**, **Assistant Professor, Department of Computer Science and Engineering,** Anurag University for his constant encouragement and inspiring guidance without which this project could not have been completed. His critical reviews and constructive comments improved our grasp of the subject and steered us to the fruitful completion of the work. His patience, guidance, and encouragement made this project possible.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, **Dean, Department of Computer Science and Engineering**, Anurag University. We also express our deep sense of gratitude to **Dr. V.V.S.S.S Balaram**, Academic co-ordinator, **Dr. Pallam Ravi**, Project in-Charge, **Dr A. Jyothi**, Project Co-Ordinator and Project review committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage our project work.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, **Dean School of Engineering**, Anurag University, for their encouragement and timely support of our B.Tech program.

G. Srilatha (20eg105647)

K. Nithin Reddy (20eg105653)

ABSTRACT

In an era where digital communication plays an indispensable role in everyday life, ensuring accessibility for all individuals, including those with visual impairments, is paramount. The current landscape of email management systems presents significant obstacles for visually impaired individuals. These systems heavily rely on visual interfaces, rendering them inaccessible for blind users. The lack of efficient accessibility features exacerbates communication challenges, hindering effective email handling. The Voice Dialogue System for Email project emerges as a pioneering solution, leveraging advanced techniques from the fields of Natural Language Processing (NLP) and machine learning, specifically the Transformers Neural Network, to revolutionize email management for the visually impaired community.

The Voice Dialogue System for Email, a transformative email management system designed to empower users affected by blindness with seamless access to their email correspondence. By harnessing the capabilities of the English language model, the Voice Dialogue System for Email project offers a comprehensive suite of features tailored to enhance accessibility and usability. Through innovative functionalities such as automated email categorization, intelligent summarization, and adaptive response generation, the Voice Dialogue System for Email streamlines email management tasks while reducing cognitive load for visually impaired users.

Key Words— Assistive Technologies, Blindness and Visual Impairment, Email Management, Natural Language Processing (NLP), Speech Recognition, Text Summarization

TABLE OF CONTENTS

S No	CONTENT	Page No
1.	**Introduction**	1
	1.1 Overview	1
	1.2 Scope of Project	1
	1.3 Problem Statement	2
	1.4 Objective	2
2.	**Literature survey**	3-6
3.	**Analysis**	7
	3.1 Existing System	7
	3.2 Challenges in Existing System	7
	3.3 Methodologies Used	8
	3.4 Disadvantages of Existing system	8
	3.5 Why our Project is a Solution	9
4.	**Proposed Method**	10
	4.1 Key components of Proposed Method	10
	4.2 Advantages of Proposed Method	11-12
	4.3 Comparison of Existing System	13

	4.4 System Requirements	14
	4.4.1 Hardware Requirements	14
	4.4.2 Software Requirements	14
	4.4.3 Specific Requirements	14
5.	**System Design**	15
	5.1 System Architecture	
	5.1.1 User Interaction	16
	5.1.2 Data Preparation and Environment Setup	17
	5.1.3 Speech Recognition Model Training	17
	5.1.4 Language Model Creation	17
	5.1.5 Intent and Token Classification	18
	5.1.6 Email Management Integration	18
	5.1.7 Deployment and User Interface	19
6.	**Implementation**	20
	6.1 Project Modules	
	6.2 Development Tools and Libraries for Dialogue System for Email	21-22
	6.3 Sample Code	
	6.3.1 Email Module	22-23
	6.3.2 ASR Module	24

	6.3.3 Understanding Module	25
	6.3.4 Integration of Module	26-32
7.	**Experiment Results**	
	7.1 Web app Screenshots	33-34
	7.2 Experiment Screenshots	35-37
8.	**Testing**	38-39
9.	**Experimental Results and Justification**	40-44
10.	**Conclusion**	45
11.	**References**	46

LIST OF FIGURE

Figure. No.	Figure. Name	Page No.
Figure 5	Project Execution Flow	15
Figure 6	Process Flow	21
Figure 7.1.1	Streamlit for Login Page	33
Figure 7.1.2	Streamlit for email dashboard	33
Figure 7.1.3	Streamlit interface for send email	34
Figure 7.1.4	Streamlit interface for read email	34

Figure 7.2.1	Mail sent through voice commands	35
Figure 7.2.2	Mail Forwarded through voice commands	35
Figure 7.2.3	Reply to email through Voice Commands	36
Figure 7.2.4	Inbox before Deleting email using Voice Commands	36
Figure 7.2.5	Email Deleted using Voice Commands	37

LIST OF TABLESLIST OF TABLES

S No	Tables	Page No.
1	Table 2.2 Comparison of literature	6
2	Table 4.3. Comparison of Existing System	13
2	Table 8.1 Speech Recognition Accuracy Test case	39
3	Table 8.2 Flow Test Case	39
4	Table 8.3 TTS Test Case	39
5	Table 9 Parameter Formulae	44

LIST OF GRAPHS

S No	Graphs	Page No
1	Graph 9.1 Comparing BERT	40
2	Graph 9.2 BERT vs Traditional	41
3	Graph 9.3 Experiment Finding	42

1. INTRODUCTION

This E-mail architecture that will help the Blind people to access the services easily and efficiently for communication without previous training. This system designed for drawback that motivates targeted solution focus on effectively usage by both handicapped and illiterate persons. By utilizing TTS for voice interaction, Gmail API for email retrieval, composition, and organization, and Cloud Storage for storing user preferences and data securely, this seeks to revolutionize email management for the visually impaired.

1.1 Overview:

The Voice Dialogue System for Email is a project aimed at addressing the challenges faced by individuals with visual impairments in managing email communication. It leverages advanced AI technologies, particularly Natural Language Processing (NLP), to enhance accessibility and empower users with efficient email navigation tools. The system goes beyond basic accessibility features by offering innovative functionalities that streamline email management, making it more intuitive and independent for visually impaired users.

1.2 Scope of Project:

The project focuses on developing a Voice Dialogue System for Email that utilizes NLP to interpret user commands, extract relevant information from emails, and facilitate seamless interaction through voice commands or other intuitive interfaces. Key functionalities include summarizing emails, categorizing messages, prioritizing important information, and composing replies. The scope also encompasses user-centric design principles to ensure the system is user-friendly and inclusive.

1.3 Problem Statement:

The internet new advancement has been implemented very efficiently the visually challenged users find it very difficult to use the technologies as normal users. This project aims in developing an E-mail architecture that will help the Blind people to access the services easily and efficiently for communication without previous training. This system designed for drawback that motivates targeted solution focus on effectively usage by both handicapped and illiterate persons.

A voice email manager ... with Neural Networks. It uses a wav2vec2 + LM for ASR, google cloud TTS for voice and BENT for understanding (intent classification and token classification). The system is trained with synthetic data. It uses RDF for dialogue state tracking and smtp/pop3 to communicating with email server.

1.4 Objective:

The objective of the Voice Dialogue System for Email project is to address the challenges faced by individuals with visual impairments by developing an innovative email management solution. The system aims to empower users with efficient email navigation tools, leveraging AI technologies such as NLP to enhance accessibility and enable independent communication. By providing intuitive interfaces and advanced functionalities, the project seeks to create an inclusive digital environment where individuals with visual impairments can communicate confidently and independently.

2. LITERATURE SURVEY

Effective email management is a cornerstone of modern communication, yet traditional email clients often pose significant challenges for individuals with visual impairments. This necessitates accessible solutions that empower visually impaired users to navigate their inboxes efficiently and independently. Several studies highlight the limitations of traditional email clients for visually impaired users. The field of email management for visually impaired and differently abled individuals has seen significant advancements in recent years, as evidenced by several notable research papers. These papers propose innovative methods and technologies aimed at improving accessibility and usability for users with disabilities. Among the notable contributions is a paper by A. Singh, R. Patel, and S. Gupta, which proposes a multi-modal email interface integrating speech recognition, braille display, and haptic feedback. This comprehensive approach aims to provide customizable and accessible interaction for visually impaired users, although complexities in setup and compatibility may pose challenges. Additionally, K. Jain, N. Shah, and R. Desai introduce a context-aware email management system leveraging machine learning algorithms for personalized assistance. While context-aware features enhance productivity, concerns regarding data privacy and initial setup remain pertinent.

Further exploration comes from T. Rao, V. Sharma, and G. Singh, who present a mobile application tailored for individuals with motor disabilities, featuring gesture recognition and voice commands for hands-free operation. While offering intuitive interaction, the availability of compatible hardware and the accuracy of gesture recognition may affect widespread adoption. S. Mehta, A. Joshi, and P. Dave contribute a paper focusing on optimizing email interfaces for low-vision users, incorporating high-contrast themes and magnification tools. Although enhancing readability, compatibility with assistive technologies and user interface adjustments require careful consideration.

Meanwhile, H. Kumar, S. Singhania, and R. Agarwal propose a web-based email client with conversational interaction capabilities enabled by natural language processing. This approach fosters hands-free operation and accessibility, yet speech recognition accuracy and integration complexities remain key challenges. Lastly, M. Gupta, P. Verma, and S. Choudhary explore a wearable device for email management, featuring tactile feedback and voice navigation. While enhancing portability, limitations in processing power and user interface design complexity must be addressed. A comparative analysis of selected literature provides insights into the various approaches and their respective advantages and disadvantages.

In their paper, Ingle et al. [1] introduced an email architecture leveraging Text-to-Speech (TTS), Speech-to-Text (STT) conversions, and Interactive Voice Response (IVR) technologies. Their system focuses on reducing cognitive load for visually impaired users through simple mouse-based interactions. While offering benefits in terms of accessibility, this approach may require additional setup for integrating IVR technology.

Sharma et al. [2] presented a voice-activated email management system emphasizing single-action interactions. Integrating speech recognition and standard email protocols (IMAP, SMTP), their system provides a user-friendly interface for email management. However, challenges with speech recognition accuracy may impact reliability.

Kumar et al. [3] introduced an accessible email management system utilizing IVR technology, offering step-by-step voice guidance. Designed to cater to diverse user demographics, including visually impaired, illiterate, and new users, this approach may face limitations in IVR functionality compared to traditional interfaces.

Tirpathi et al. [4] proposed a voice-based email system specifically designed for visually impaired and differently abled individuals. Their system utilizes voice commands for email management tasks, aiming to enhance accessibility and usability. However, errors in voice recognition may pose challenges during interactions.

Harshasri et al. [5] presented a tailored voice-based email system targeting blind users, with a focus on accessibility and intuitive interface design. Leveraging voice commands and speech-to-text conversion, their system aims to provide an accessible email experience. Training and customization may be required for optimal speech recognition accuracy.

Tyagi et al. [6] introduced a voice-based email system for physically challenged individuals, prioritizing accessibility and ease of use. Their approach emphasizes voice commands for email management tasks, addressing the needs of users with physical disabilities. However, vocal fatigue and limitations in speech recognition accuracy over time may present challenges.

Overall, the literature survey highlights the diverse range of approaches and technologies employed in email management systems for individuals with disabilities. While each approach offers unique advantages in terms of accessibility and usability, challenges such as speech recognition accuracy, hardware/software requirements, and limitations of IVR systems need to be addressed to ensure effective implementation and user satisfaction.

Reference	Authors	Methods	Advantages	Disadvantages
[1]	P. Ingle, H. Kanade, A. Lanke, M. Choche	Text-to-Speech (TTS), Speech-to-Text (STT), Interactive Voice Response (IVR)	Reduces cognitive load for visually impaired users, simple mouse interaction	Limited details on user evaluation or interface design
[2]	B. Sharma, B. Roshan, M. Gaur, P. De	Speech recognition, TTS, standard email protocols (IMAP, SMTP)	Simple and user-friendly interface, single-action interaction	May lack features for advanced email management tasks
[3]	S. Kumar, R. Yogitha, R. Aishwarya	Interactive Voice Response (IVR) technology	Accessible to diverse users (visually impaired, illiterate), step-by-step voice guidance	Might be less efficient for experienced email users
[4]	S. Tripathi, N. Kushwaha, P. Shukla	Voice commands	Improved accessibility and usability for users with disabilities	Limited details on specific voice commands or system functionalities
[5]	M. Harshasri, M. Durga Bhavani, R. Misra	Voice commands, Speech-to-Text (STT)	Accessible and intuitive interface for blind users	May lack customization options for voice commands or speech recognition accuracy
[6]	P. Tyagi, T. Sharma, M. Mittal, A. Kumar	Voice commands	Accessibility and ease of use for users with physical disabilities	May not be tailored specifically for the needs of visually impaired users (e.g., screen reader compatibility)

Table 2.2 Comparison of Literature

3.ANALYSIS

3.1 Existing System

While traditional email clients offer basic functionalities, they often present significant barriers for visually impaired users. These, clients typically offer basic functionalities like reading and sending emails, but lack features that facilitate efficient and independent navigation for blind or visually impaired users. This can lead to frustration and hinder effective communication, creating a barrier to participation in the digital world. Let's delve deeper into these limitations and explore existing methodologies to address them.

3.2 Challenges in Existing System:

Navigational Difficulties: Traditional interfaces rely heavily on visual elements like menus and icons. Screen readers or keyboard shortcuts, while helpful, can be cumbersome for complex layouts, hindering efficient navigation.

Comprehension Issues: Lengthy text-based emails can be overwhelming for visually impaired users. Traditional clients offer limited support in summarizing content or extracting key information, making it difficult to grasp email messages quickly.

Response Composition Challenges: Composing clear and concise emails can be a struggle. Traditional clients might lack features that assist with dictation, grammar correction, or organization of thoughts, making the process time-consuming and frustrating.

3.3 Methodologies Used:

Speech-to-Text (STT): Converts spoken language into text, enabling users to dictate emails verbally.

Text-to-Speech (TTS): Converts text-based email content into synthesized speech, providing auditory feedback for users.

Interactive Voice Response (IVR): Allows users to interact with email systems using voice commands and prompts.

3.4 Disadvantages of Existing System for Visually Impaired Users:

Existing email systems for visually impaired users often fall short in several key areas:

Limited Functionality: Many existing systems offer basic functionalities like speech recognition and text-to-speech conversion. While helpful, they lack more advanced features that can significantly improve email management. These limitations can include:

- Inability to understand user intent (what the user wants to do)
- Difficulty in prioritizing emails (e.g., identifying urgent messages)
- Lack of features to assist with composing clear and concise emails

Complexity of Use: While some systems offer voice interaction, they may still require users to navigate menus or learn specific voice commands. This can be cumbersome and limit usability, especially for new users.

Accessibility Issues: Even with screen readers and keyboard shortcuts, traditional email interfaces can be challenging for visually impaired users to navigate due to complex layouts and reliance on visual cues.

3.5 Why our Project is a Solution:

our project, "Voice Dialogue System for Email," addresses these disadvantages by leveraging Natural Language Processing (NLP). Here's how it stands out:

Enhanced Functionality: Our project incorporates features like intent recognition, sentiment analysis, and entity recognition. These features enable users to:

- Manage email through natural voice commands (understand user intent)
- Prioritize emails based on urgency (sentiment analysis)
- Quickly find specific information within emails (entity recognition)

Improved User Experience: By focusing on natural language interaction and eliminating the need for complex navigation, your project creates a more intuitive and user-friendly experience for visually impaired users.

Increased Accessibility: Our project leverages voice technology, making email management more accessible for users who struggle with traditional interfaces.

4. PROPOSED METHOD

Our proposed method aims to develop a comprehensive email management system specifically tailored for individuals with visual impairments. The system utilizes advanced AI technologies to provide a seamless and accessible user experience, empowering visually impaired users to efficiently manage their emails through voice interaction.

4.1 Key Components of Proposed Method

- **Speech Recognition:** The system incorporates state-of-the-art speech recognition algorithms to accurately transcribe user commands from spoken language into text, enabling users to interact with the email system using voice commands.

- **Natural Language Understanding (NLU):** Advanced natural language processing (NLP) techniques are employed to understand the intent behind user commands and extract relevant information from email content. This enables the system to comprehend complex user requests and perform tasks accordingly.

- **Text-to-Speech (TTS):** The system utilizes high-quality text-to-speech synthesis to convert email content into natural-sounding speech output. This allows visually impaired users to listen to email messages and responses audibly, facilitating efficient comprehension and communication.

- **Email Management Functionality:** The proposed system includes core email management functionalities such as reading, composing, replying, forwarding, and organizing emails. Users can perform these tasks using intuitive voice commands, streamlining the email management process.

- **Accessibility Features:** The system is designed with accessibility in mind, incorporating features such as screen reader compatibility, keyboard navigation support, and high-contrast interfaces to accommodate the needs of visually impaired users**.**

4.2 Advantages of Proposed Method

Our project, Voice Dialogue System for Email, is positioned as a solution to these challenges. It distinguishes itself from previous efforts by not only ensuring accessibility but also by actively enhancing the email management process. Here are some key benefits of the proposed system:

- **Enhanced Accessibility**: The proposed method significantly enhances email accessibility for visually impaired users, providing a voice-controlled interface that eliminates barriers to email management**.**

- **Efficient Communication**: By leveraging speech recognition and text-to-speech synthesis, the system enables visually impaired users to efficiently communicate through email, enhancing their productivity and independence.

- **Intuitive Interaction:** The system offers an intuitive and user-friendly interaction model, allowing users to perform email management tasks using natural voice commands without the need for complex interfaces or manual input.

- **Improved Comprehension:** Through advanced natural language understanding, the system can summarize email content and extract key information, helping visually impaired users comprehend email messages more effectively.

- **Empowerment and Inclusion:** By providing visually impaired users with a comprehensive and accessible email management solution, the proposed method promotes empowerment and inclusion in the digital realm, enabling users to participate more fully in email communication and social interactions.

4.3 Comparison of Existing System

S:NO	Strategies	Method	Advantages	Disadvantages
1.	Basic Email Management	Speech-to-Text (STT), Text-to-Speech (TTS), Interactive Voice Response (IVR)	Reduces cognitive load for visually impaired users. - Simple mouse-based interaction	Limited capabilities, requires more user effort, not for blind persons
2.	Voice-activated Email Management with Single Action at a Time	Speech recognition, Text-to-Speech, IMAP, SMTP	Simple and user-friendly single-action interaction. - Standard protocols for email access.	Limited capabilities compared to modern methods. - Potentially requires training for voice commands.
3.	Accessible Email Management	Text-to-Speech, Speech-to-Text (Python libraries), IVR	Accessible to diverse users: visually impaired, illiterate, new users. - Step-by-step voice guidance	Limited functionality compared to modern methods. - Might require user adaptation to voice prompts

4.4 System Requirements

The Requirements to implement the system are as follows.

4.4.1 Hardware Requirements

- PROCESSOR : Intel Core i5
- RAM : 8GB
- HARD DISK : 260GB

4.4.2 Software Requirements

- OPERATING SYSTEM : Linux
- BACK-END : Python3
- DATABASE : English Corpus
- FRONT-END : Streamlit

4.4.3 Specific Requirements

- Kenlm
- Wav2vec2
- Google TTS API

5.SYSTEM DESIGN

5.1 System Architecture

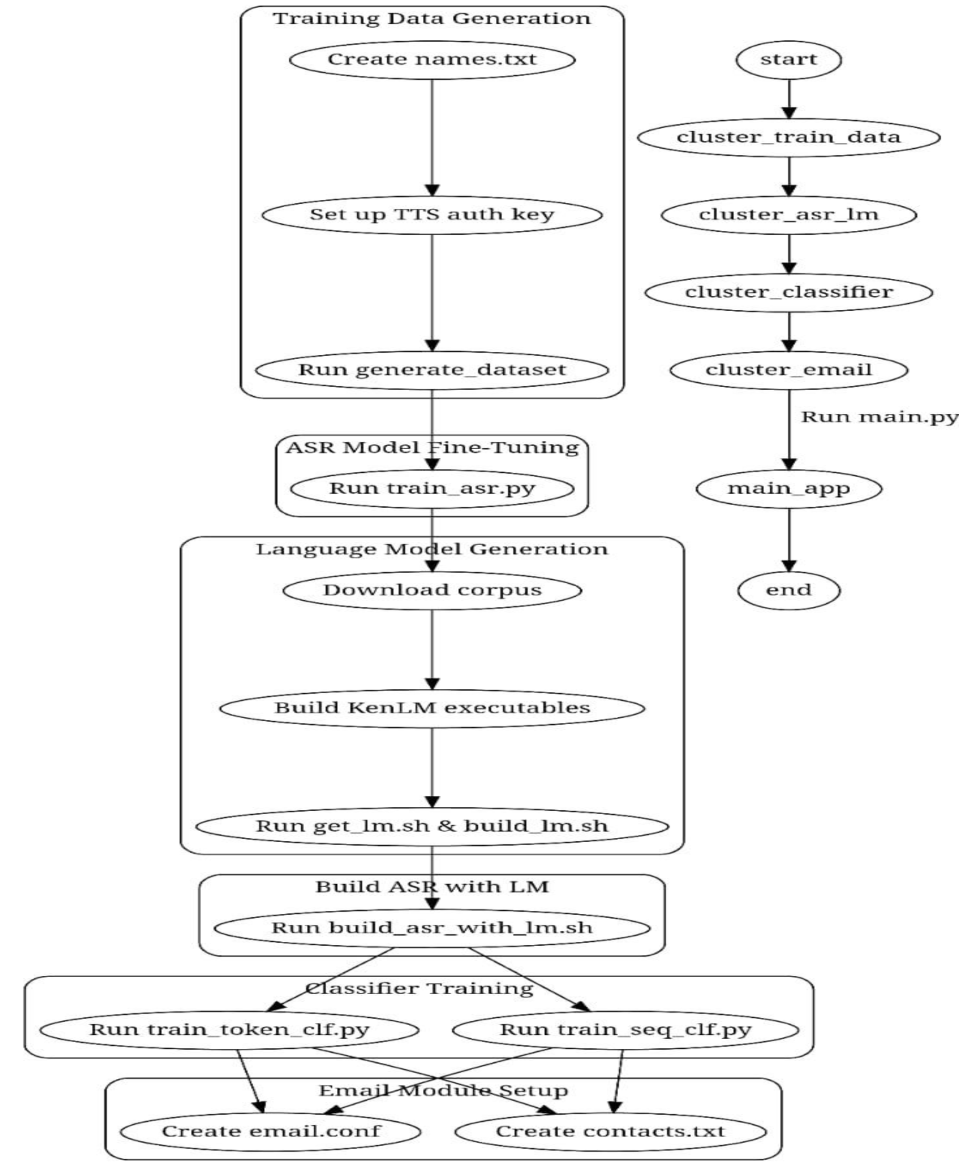


Figure 1: Project Execution Flow

5.1.1 Phase - 1: User interaction

The user interaction phase serves as the entry point for users to interact with our video restoration system. This phase prioritizes a user-friendly experience by leveraging Streamlit.

Streamlit: It is a Python library specifically designed for simplifying web application development. It automatically generates a user-friendly interface based on your code. This includes elements like text boxes, buttons, sliders, and charts. Streamlit web apps are lightweight and can be easily deployed on various platforms.

- It allows users to conveniently login before performing email operations.
- It provides a clear dashboard for the email system with actionable buttons for navigating to specific tasks.
- It gives a Voice command on progress during certain operations like sending mail, reading mail, forwarding email, deleting email, etc.,
- It will take the voice as input, converts that into text and automatically fill the input text fields with an proper guidance for Virtually enabled People
- Upon successful processing, Streamlit will perform the email operations by creating successful communication with the end user.

By utilizing Streamlit, Users can interact with the system entirely through their web browser, eliminating the need for software installations or complex technical knowledge.

5.1.2 Phase 2: Data Preparation and Environment Setup

Activities:

- Design a simple UI to collect user data for email management tasks (e.g., compose, send, read). Users can record voice commands and enter corresponding text instructions.

- Store the collected data (voice recordings and text instructions) in a designated storage location.
- Create a Python virtual environment (ideally using Python 3.8) to isolate project dependencies.
- Install required libraries like transformers for NLP, torchaudio for audio processing, and kenlm for language modeling.

5.1.3 Phase 3: Speech Recognition Model Training

Activities:

- Develop a speech preprocessing module to perform actions like noise reduction, silence removal, and feature extraction on the collected voice recordings.
- Utilize the Wav2Vec2 model architecture for automatic speech recognition (ASR). Fine-tune the model using the prepared training dataset.
- Evaluate the performance of the fine-tuned ASR model on a held-out test set. Refine the training process if necessary.

5.1.4 Phase 4: Language Model Creation

Activities:

- Develop a text preprocessing module to handle tasks like tokenization, lowercasing, and removal of punctuation from the text instructions.

- Download a portion of the English corpus to serve as the basis for the language model (LM).
- Utilize KenLM tools to build an n-gram language model using the preprocessed corpus.

5.1.5 Phase 5: Intent and Token Classification

Activities:

- Leverage the existing text preprocessing module for preparing the text instructions for classification.
- Train a token classification model (e.g., using a BERT-based architecture) to identify specific entities and actions within user commands (e.g., recipient name, email subject, keywords).
- Train an intent classification model (also using BERT) to categorize the overall goal of the user's voice command (e.g., compose email, send email, read email).

5.1.6 Phase 6: Email Management Integration

Activities:

- Integrate the fine-tuned speech recognition model to convert user voice commands into text.
- Integrate the token and intent classifiers to extract specific actions and entities from the recognized text.
- Implement a dialogue state tracker to maintain context across user interactions.

- Develop an email client module that interacts with the email server (using protocols like SMTP/POP3) to perform actions based on the user's intent and extracted information.

5.1.7 Phase 7: Deployment and User Interface

Activities:

- Deploy the trained models, dialogue state tracker, and email client onto a suitable platform (local machine or cloud server).
- Develop a user-friendly interface that allows users to interact with the system through voice commands. The interface should display relevant information and provide feedback on actions taken.
- Integrate the different components into a cohesive system that can handle user requests for email management tasks.
- This is a phased approach to building your ASR dialogue system. Each phase focuses on specific functionalities, allowing for modular development and testing. Remember to adapt this architecture based on your specific needs and chosen technologies.

6. IMPLEMENTATION

6.1 Project Modules

Speech Recognition Module: This module will handle the conversion of spoken language into text, enabling users to interact with the system using voice commands.

Text-to-Speech (TTS) Module:

Converts email content and system responses into synthesized speech, providing auditory feedback to visually impaired users.

Email Management Module:

Handles core email management functionalities such as reading, composing, replying, forwarding, and organizing emails.

Provides an intuitive interface for users to interact with their email accounts using voice commands.

Accessibility Module:

Ensures that the system is accessible and user-friendly for visually impaired users.

Includes features such as screen reader compatibility, keyboard navigation support, and high-contrast interfaces**.**

User Authentication and Account Management Module:

Manages user authentication and authorization processes, ensuring secure access to email accounts.

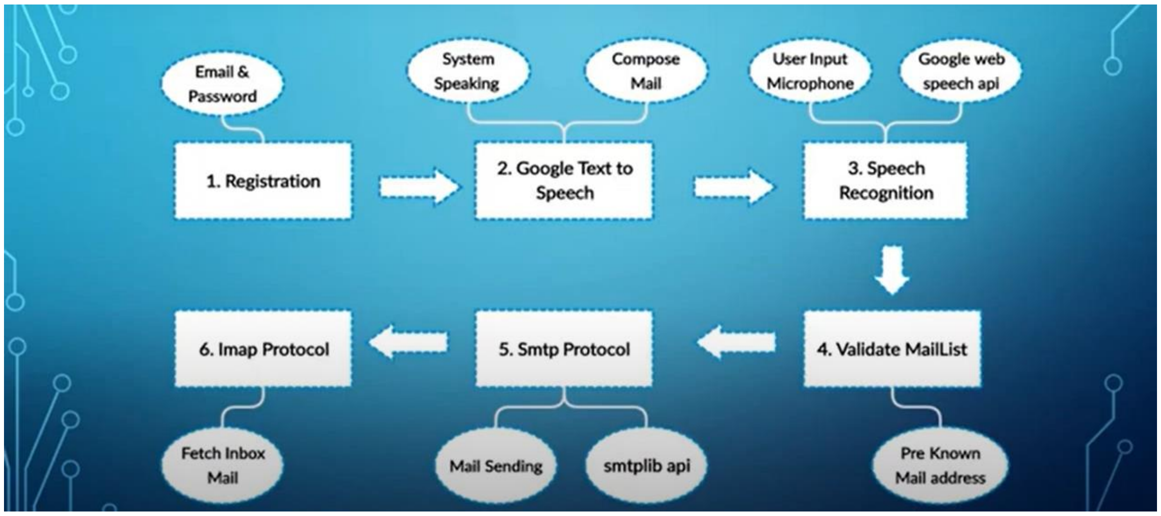


Fig 2: Process Flow

6.2 Development Tools and Libraries for Voice Dialogue System for Email:

The Voice Dialogue System for Email leverages a combination of programming languages, frameworks, and libraries to deliver its functionalities for visually impaired users. Here's a breakdown of some key tools and how they contribute to the system:

Programming Languages:

Python: Python serves as the primary programming language due to its readability, extensive libraries, and large developer community.

Libraries:

Natural Language Processing (NLP) Library: This library enables tasks like speech recognition, text-to-speech conversion, sentiment analysis, and intent classification, allowing the system to understand and respond to user voice commands and emails.

Speech Recognition Library: A speech recognition library like SpeechRecognition helps convert spoken user commands into text, enabling voice interaction with the email system.

Text-to-Speech Library: A text-to-speech library like gTTS or PyTTSX3 allows the system to convert email content and responses into audio for users to listen to.
Email Client Library: An email client library like imaplib or smtplib streamlines interaction with email servers, enabling the system to access, manage, and compose emails.

6.3 SAMPLE CODE

6.3.1 Email Module:

The Email Module class in our project handles functionalities related to email management. It interacts with email servers using libraries like smtplib and poplib to perform actions like sending, receiving, and deleting emails. Additionally, it maintains a local database of emails and a contact list.

Initialization: The Email Module initializes connections to both the SMTP (Simple Mail Transfer Protocol) server for sending emails and the POP3 (Post Office Protocol) server for receiving emails. It also retrieves user credentials and contact information from configuration files to authenticate and interact with email servers.

Email Retrieval: Upon initialization, the module retrieves emails from the user's inbox using the POP3 protocol. It parses the email headers and bodies to extract relevant information such as the sender, subject, and timestamp. This information is stored in a local database for further processing.

Email Dispatching: The module provides a method for dispatching user intents related to email interactions. It handles actions such as sending, replying to, forwarding, and deleting emails based on the user's commands. Additionally, it resolves recipient email addresses using a contact database and sends emails via the SMTP server.

Email Querying: Users can query their inbox for specific emails based on criteria such as the subject, timestamp, or sender. The module facilitates email retrieval based on user-defined filters and returns the matching emails from the local database.

```
class EmailModule:
        with open('data/email.conf', 'r') as conf_file:
        conf = conf_file.readlines()
        smtp_server, smtp_port = conf[0].replace('\n', '').split(':')
        pop3_server, pop3_port = conf[1].replace('\n', '').split(':')
        self.user, pwd = conf[2].split(':')
    self.smtp_server = smtplib.SMTP_SSL(smtp_server, int(smtp_port))
    self.smtp_server.ehlo()
    self.smtp_server.login(self.user, pwd)
    self.pop3_server = poplib.POP3_SSL(pop3_server, int(pop3_port))
    self.pop3_server.user(self.user)
    self.pop3_server.pass_(pwd)

    with open('data/contacts.txt', 'r') as contancts_file:
        self.person_db = {}
        for contact in contancts_file.readlines():
            person, email = contact.replace('\n', '').split(':')
            self.person_db[person] = email
```

6.3.2 ASR MODULE:

The project aim is to provide a voice-based interface for email management. The ASRModule class plays a crucial role in achieving this by handling Automatic Speech Recognition (ASR). It utilizes libraries like librosa and speech_recognition to:

- Capture user audio input through the microphone.
- Convert the captured audio into text using a speech recognition engine (e.g., Google Speech-to-Text).
- Handle potential errors during speech recognition and provide user feedback.

```
def transcribe_audio(self, time=5):
    print('waiting for a reply ....')
    sd.play(self.beep, self.sr)
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("Speak something...")
        audio_data = recognizer.listen(source)
        print("listening")
        # Convert speech to text
        try:
          text = recognizer.recognize_google(audio_data)
          print("text:", text)
          print('Processing...')
          return format_asr_result(text)
        except sr.UnknownValueError:
            print("Sorry, could not understand audio.")
        except sr.RequestError as e:
            print("Error occurred:", str(e))
        return '0'
```

6.3.3 Understanding Module:

The Understanding Module class plays a crucial role in project by interpreting user intent and extracting relevant information from their spoken commands. It leverages pre-trained transformer models from the transformers library:

Token Classification Model (token_clf): This model identifies the role of each word in a sentence. For example, it can distinguish between a person's name (labeled as "PER") and the subject of an email (labeled as "OBJ").

Sequence Classification Model (seq_clf): This model classifies the overall intent of the user's command. For instance, it can determine if the user wants to "send an email," "read an email," or perform another action.

```
def process(self, text):
    intent = format_seqclf_result(self.seq_clf(text))
    slots = format_tokenclf_result(self.token_clf(text))
    ret_slots = []
    for slot in slots:
        if not slot['label'] == 'O':
            if slot['label'][-3:] == 'PER':
                ret_slots.append(('person', slot['text']))
            elif slot['label'][-3:] == 'OBJ':
                ret_slots.append(('object', slot['text']))
            elif slot['label'][-4:] == 'DATE':
                ret_slots.append(('date', slot['text']))
    return intent, ret_slots
```

6.3.4 Integration of Modules:

In addition to individual modules, our project also features an integrated application that utilizes multiple modules for interactive email management. The App class serves as the central component responsible for orchestrating the interaction between the ASR, email, and understanding modules to facilitate seamless email operations through voice commands.

Initialization: The App class initializes instances of the ASRModule, EmailModule, and UnderstandingModule to enable speech recognition, email management, and intent understanding functionalities, respectively. It also configures the Speaker class for text-to-speech output.

```
class App:
    def __init__(self):
        self.speaker = Speaker()
        self.speaker.say('loading forms, please wait')
        self.understanding_module = UnderstandingModule()
        self.asr_module = ASRModule()
        self.opened_mail = []
        self.mail_module = EmailModule()
        self.graph_module = GraphDST()
        self.speaker.say('loading complete')
    def main_loop(self):
        self.graph_module.print_graph()
        if len(self.opened_mail) == 0:
            self.speaker.say('what do you want to do?')
            self.graph_module.exchange('System', 'what do you want to do?')
```

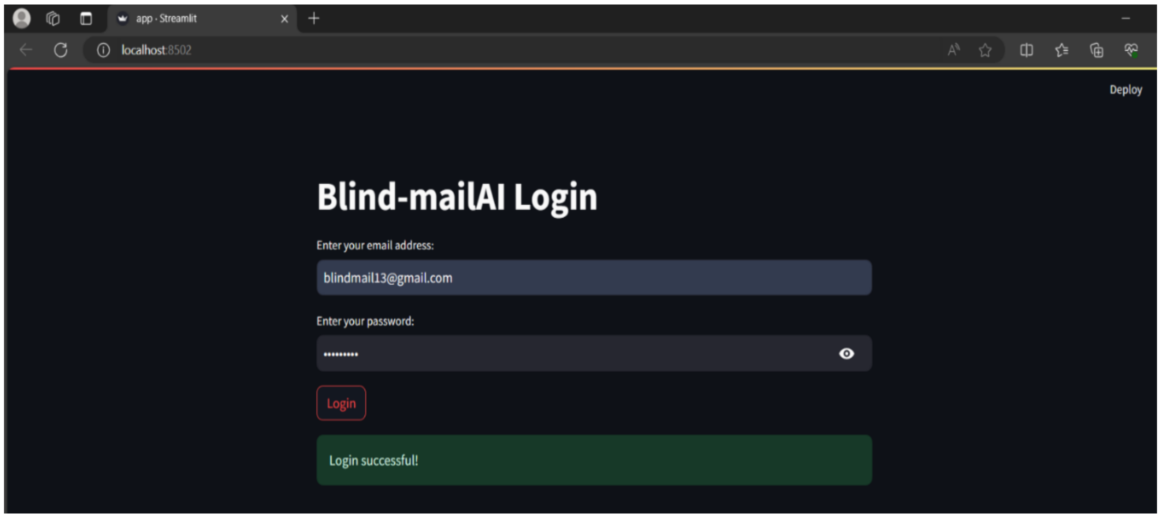
```
 text = self.asr_module.transcribe_audio(6)
           print("ldone")
           if not text:
               self.main_loop()
               return
           intent, slots = self.understanding_module.process(text)
           print(intent, slots)
           # add successor
           if intent == 'send_email' and len(self.opened_mail) == 0:
               self.speaker.say('send an email')
               self.graph_module.exchange('System', 'send an email')
               mail = {'object': None, 'person': None, 'body': None}
               for slots in slots:
           if slot[0] == 'object':
             mail['object'] = slot[1]
                   elif slot[0] == 'person':
                       mail['person'] = slot[1]
               self.graph_module.exchange('User', text, intent, slots)
               for field in mail.keys():
                   if mail[field] is None:
                       if field == 'person':
                           self.speaker.say('who should this be sent to?')
                           self.graph_module.exchange('System', 'who should it be sent to?')
                           mail['person'] = self.asr_module.transcribe_audio()
                           self.graph_module.exchange('User', mail['person'], None,
[('person', mail['person'])])
                       elif field == 'object':
                           self.speaker.say('what\' is the subject?')
 self.graph_module.exchange('System', 'what is the object?')
                           mail['object'] = self.asr_module.transcribe_audio()
```

```
                        self.graph_module.exchange('User', mail['object'], None,
[('person', mail['object'])])
                    elif field == 'body':
                        self.speaker.say('what is the body of the email?')
                        self.graph_module.exchange('System', 'what is the body of the email?')
                        mail['body'] = self.asr_module.transcribe_audio(10)
                        self.graph_module.exchange('User', mail['body'], None,
[('person', mail['body'])])
                    if self.ask_confirm('send'):
                self.mail_module.dispatch_intent({'intent': intent, 'mail': mail})
                self.speaker.say('mail sent')
            self.main_loop()
            return
        elif intent == 'list_email' or len(self.opened_mail) > 0:
            time = None
            object = None
            person = None
            for slot in slots:
                if slot[0] == 'person':
                    person = slot[1]
                elif slot[0] == 'object':
                    object = slot[1]
                elif slot[0] == 'time':
                    time = slot[1]
  self.speaker.say(f'I m looking for emails {f"from {person}" if person is not None else ""}
'
            f'{f"with object {object}" if object is not None else ""} '
            f'{f"received on {time}" if time is not None else ""}')
            self.graph_module.exchange('System',f'looking for emails {f"from {person}" if
person is not None else ""} '
```

```
 f'{f"with object {object}" if object is not None else ""} '
                          f'{f"received on {time}" if time is not None else ""}')
         self.opened_mail = self.mail_module.get_email(object, time, person)
         self.speaker.say(f"there are {len(self.opened_mail)} new emails"
                    f"{f'da {person}' if person is not None else ''} "
                    f"{f'in data {time}' if time is not None else ''} "
                    f"{f'with object {object}' if object is not None else ''}")
         self.graph_module.exchange('System', f"there are {len(self.opened_mail)} new
emails"
                                f"{f'da {person}' if person is not None else ''} "
                                f"{f'in data {time}' if time is not None else ''} "
                                f"{f'with object {object}' if object is not None else ''}")
         self.main_loop()
         return
       elif 'exit' in text:
         self.speaker.say('Bye')
         return
       elif 'help' in text:
         self.speaker.say('possible operations')
         self.graph_module.exchange('System', 'possible operations')
         self.main_loop()
         return
       else:
         self.speaker.say('I didnt understand')
         self.main_loop()
         return
     else: ###opened mail
       mail = self.opened_mail[0]
       self.speaker.say(f"mail from {mail['person']} with subject {mail['object']}, what
do you want to do?")
 self.graph_module.exchange('System',
                    f"mail from {mail['person']} with subject {mail['object']}, what
do you want to do?")
        text = self.asr_module.transcribe_audio()
        if not text:
            self.main_loop()
            return
        intent, slots = self.understanding_module.process(text)
        if intent == 'read_email':
            self.graph_module.exchange('User', text, intent, None)
            self.speaker.say(f"{mail['body']}")
            self.graph_module.exchange('System', f"{mail['body']}")
            self.main_loop()
            return
        elif intent == 'delete_email':
            self.graph_module.exchange('User', text, intent, None)
            if self.ask_confirm('delete email'):
                del self.opened_mail[0]
                self.mail_module.dispatch_intent({'intent': intent, 'mail': mail})
                self.speaker.say('email deleted')
            self.main_loop()
            return
        elif intent == 'forward_email':
new_mail = {'body': mail['body'], 'object': f"fwd:{mail['object']}", 'person': None,
'time': {'day': 'today', 'month': 'today'}}
            for slot in slots:
                if slot[0] == 'person':
                    new_mail['person'] = slot[1]
            if new_mail['person'] is None:
                self.speaker.say('who should the email be forwarded to?')
             self.graph_module.exchange('System', 'who should the email be forwarded to?')
                new_mail['person'] = self.asr_module.transcribe_audio()
                print("person:", new_mail['person'])
                if not new_mail['person']:
                    self.main_loop()
                    return
         self.graph_module.exchange('User',new_mail['person'],None, new_mail['person']
            self.graph_module.exchange('User', text, intent,
[('person', new_mail['person'])])
            if self.ask_confirm('submit'):
                self.mail_module.dispatch_intent({'intent': intent, 'mail': new_mail})
                self.speaker.say('forwarded email')
            self.main_loop()
            return
        elif intent == 'reply_email':
            new_mail = {'body': None, 'object': f're:{mail["object"]}','person': mail['person']
         self.graph_module.exchange('User', text, intent, None)
            self.speaker.say('what is the body of the email?'
 self.graph_module.exchange('System', 'what is the body of the email?')
new_mail['body'] = self.asr_module.transcribe_audio(10)
            self.graph_module.exchange('User', text, intent,
                         [('person', mail['person']), ('object', mail['object'])])
            if self.ask_confirm('answer'):
                self.mail_module.dispatch_intent({'intent': intent, 'mail': new_mail})
                self.speaker.say('answer sent')
            self.main_loop()
            return
        elif intent == 'close_email':
            self.graph_module.exchange('User', text, intent, None)
            self.graph_module.exchange('User', intent)
 if len(self.opened_mail) > 0:
                del self.opened_mail[0]
            self.main_loop()
            return
          elif 'exit' in text:
            self.graph_module.exchange('User', intent)
            self.speaker.say('hello!')
            return
          else:
            self.graph_module.exchange('User', intent)
            self.speaker.say('I didnt understand')
            self.main_loop()
           return
    def ask_confirm(self, operation):
        self.speaker.say(f'are you sure you want {operation}?'
  t = self.asr_module.transcribe_audio(3)
        if not t:
          self.main_loop()
          return
        while 'yes' not in t and 'no' not in t and 'ok' not in t and 'sure' not in t:
          self.speaker.say(f'I dont understand, are you sure you want{operation}?')
          t = self.asr_module.transcribe_audio(3)
          if not t:
            self.main_loop()
            return
        if 'yes' in t or 'ok' in t or 'sure' in t:
          return True
        return False
print("running..")
app = App()
app.main_loop()
```

7. EXPERIMENT RESULTS

7.1 Web App Screenshots:



Blind-mailAI Login

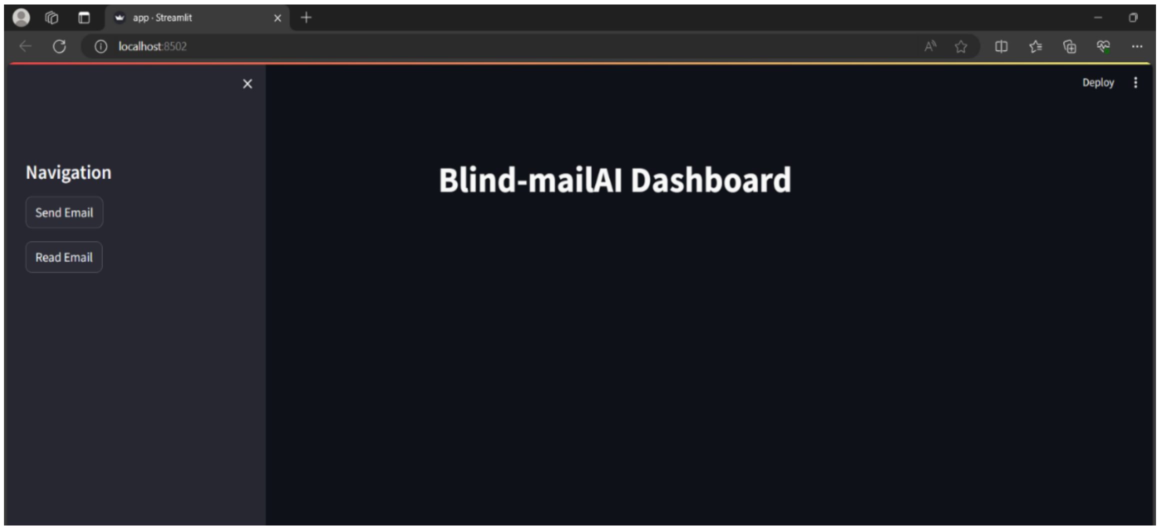
Enter your email address:

blindmail13@gmail.com

Enter your password:

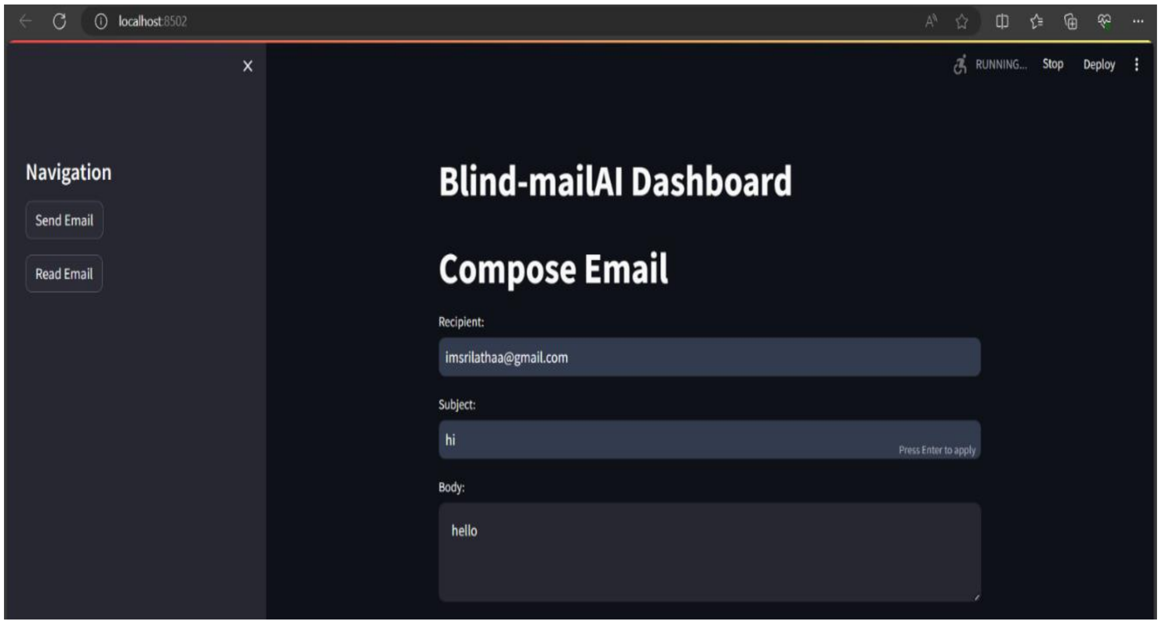
Login successful!

Figure 7.1.1: streamlit for Login Page



Blind-mailAI Dashboard

Figure 7.1.2: streamlit for email dashboard



Blind-mailAI Dashboard

Compose Email

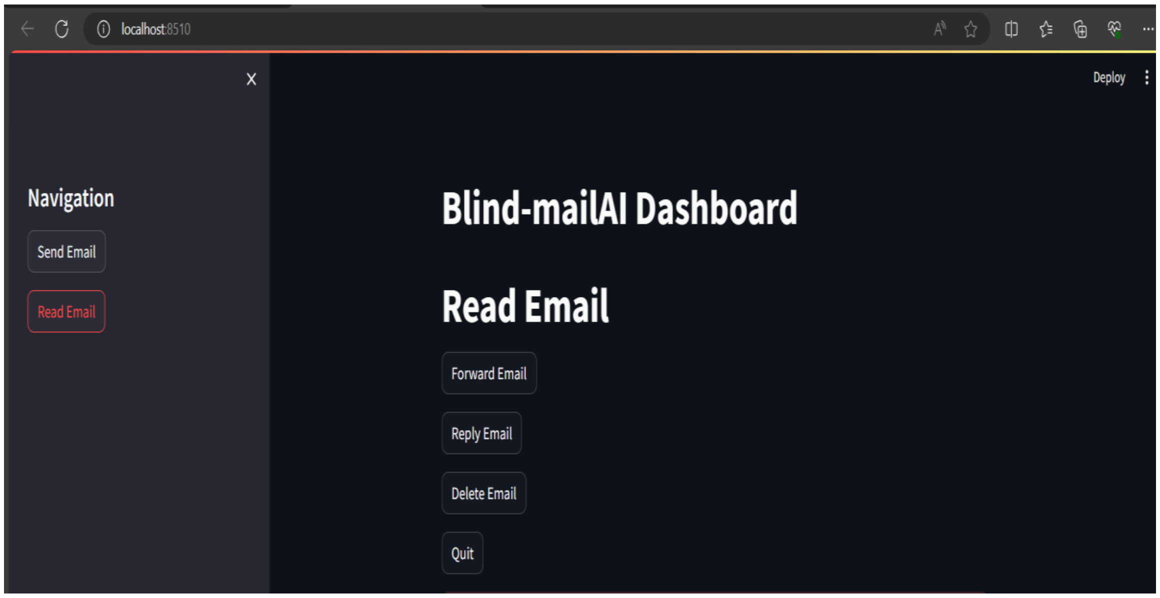
Recipient: imsrilathaa@gmail.com

Subject: hi

Press Enter to apply

Body: hello

Figure 7.1.3: streamlit interface for send email

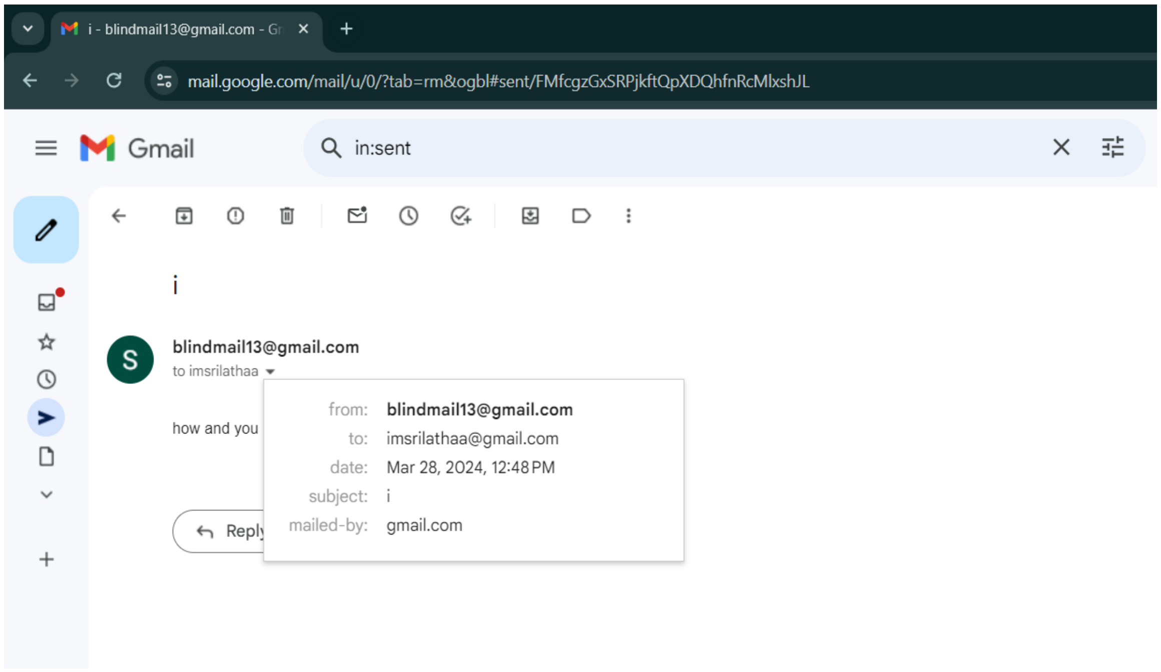


Blind-mailAI Dashboard

Read Email

Figure 7.1.4: streamlit interface for read email

7.2 Experiment Screenshots


i - blindmail13@gmail.com - Gr
mail.google.com/mail/u/0/?tab=rm&ogbl#sent/FMfcgzGxSRPjkftQpXDQhfnRcMlxshJL
Gmail
in:sent
Reply

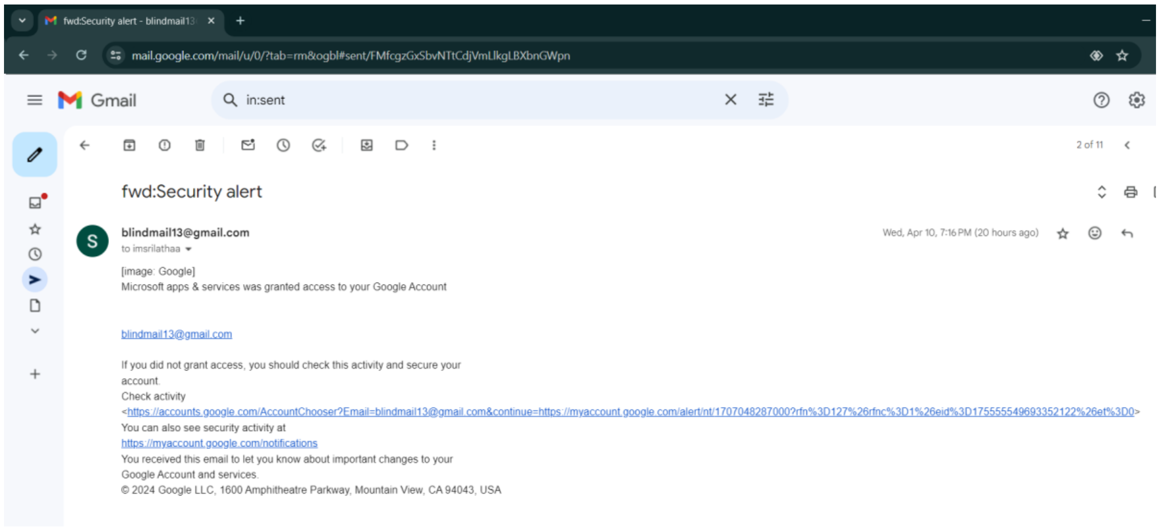
i

blindmail13@gmail.com
to imsrilathaa

how and you

from: blindmail13@gmail.com
to: imsrilathaa@gmail.com
date: Mar 28, 2024, 12:48 PM
subject: i
mailed-by: gmail.com

Figure 7.2.1: mail sent through voice commands


fwd:Security alert - blindmail13
mail.google.com/mail/u/0/?tab=rm&ogbl#sent/FMfcgzGxSbvNTtCdjVmLlkgLBXbnGWpn
Gmail
in:sent
2 of 11

fwd:Security alert

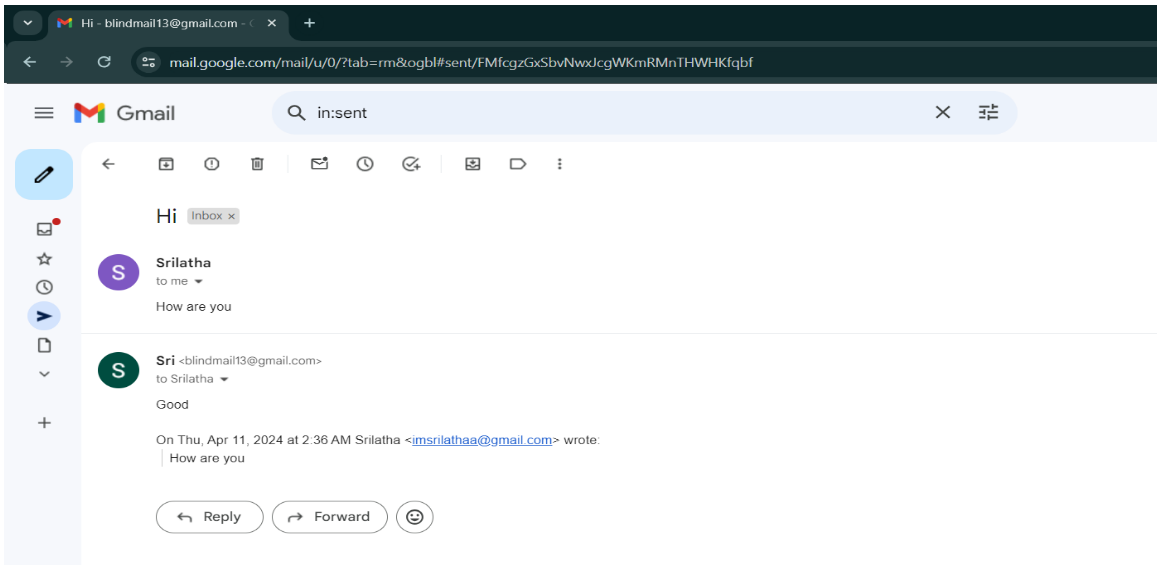
blindmail13@gmail.com
to imsrilathaa
Wed, Apr 10, 7:16 PM (20 hours ago)

[image: Google]
Microsoft apps & services was granted access to your Google Account

blindmail13@gmail.com

If you did not grant access, you should check this activity and secure your
account.
Check activity
<https://accounts.google.com/AccountChooser?Email=blindmail13@gmail.com&continue=https://myaccount.google.com/alert/nt/1707048287000?rfn%3D127%26rfnc%3D1%26eid%3D17555549693352122%26et%3D0>
You can also see security activity at
https://myaccount.google.com/notifications
You received this email to let you know about important changes to your
Google Account and services.
© 2024 Google LLC, 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA

Figure 7.2.2: mail forwarded through voice commands


Hi - blindmail13@gmail.com

mail.google.com/mail/u/0/?tab=rm&ogbl#sent/FMfcgzGxSbvNwxJcgWKmRMnTHWHKfqbf

in:sent

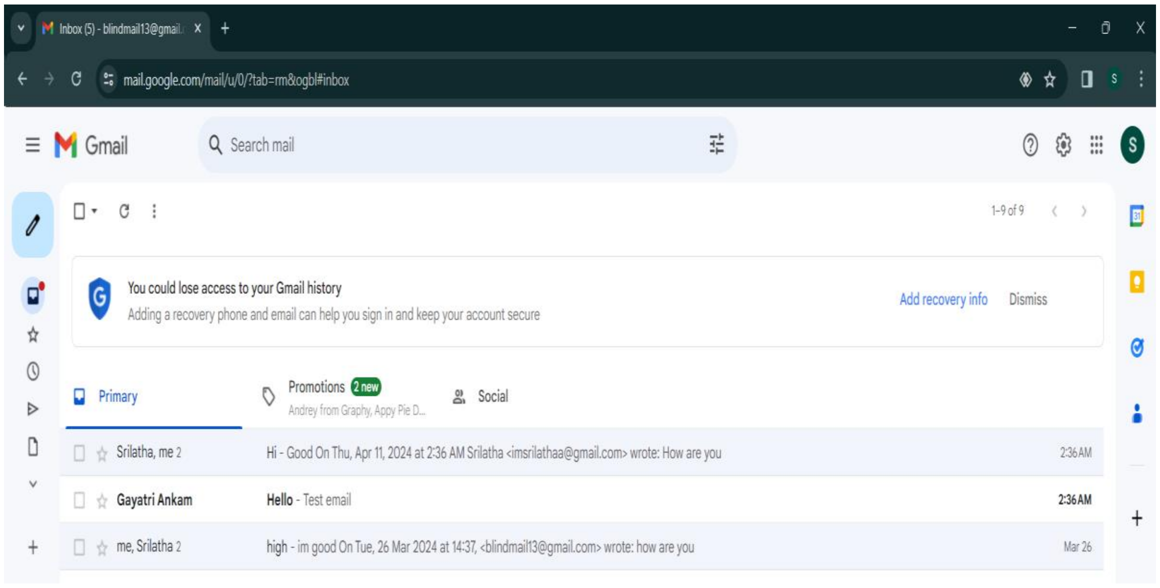
Hi Inbox

Srilatha
to me
How are you

Sri <blindmail13@gmail.com>
to Srilatha
Good

On Thu, Apr 11, 2024 at 2:36 AM Srilatha <imsrilathaa@gmail.com> wrote:
How are you

Figure 7.2.3: Reply to email through Voice Commands

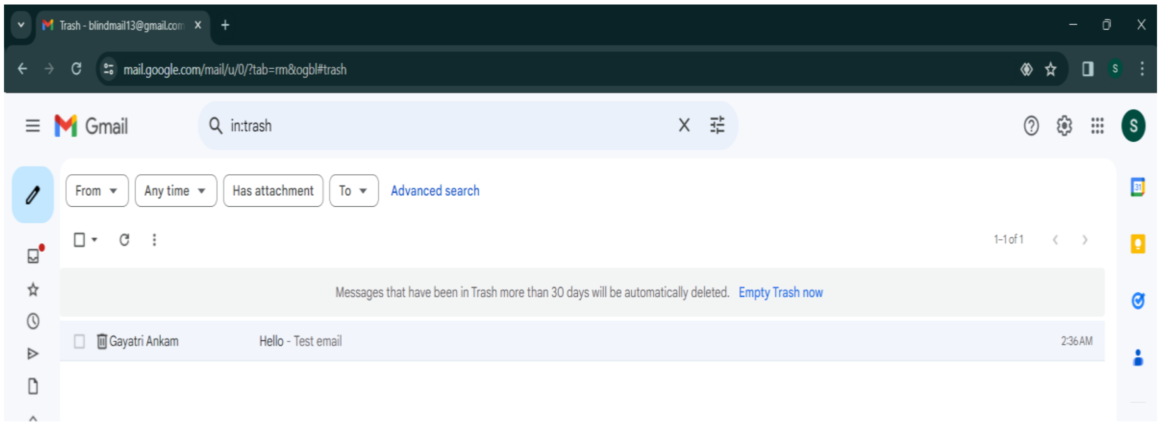

Inbox (5) - blindmail13@gmail.

mail.google.com/mail/u/0/?tab=rm&ogbl#inbox

You could lose access to your Gmail history
Adding a recovery phone and email can help you sign in and keep your account secure

Srilatha, me 2	Hi - Good On Thu, Apr 11, 2024 at 2:36 AM Srilatha <imsrilathaa@gmail.com> wrote: How are you	2:36 AM
Gayatri Ankam	Hello - Test email	2:36 AM
me, Srilatha 2	high - im good On Tue, 26 Mar 2024 at 14:37, <blindmail13@gmail.com> wrote: how are you	Mar 26

Figure 7.2.4: Inbox before Deleting email using Voice Commands


Trash - blindmail13@gmail.com
mail.google.com/mail/u/0/?tab=rm&ogbl#trash
Gmail
in:trash
From
Any time
Has attachment
To
Advanced search
1–1 of 1

Messages that have been in Trash more than 30 days will be automatically deleted. Empty Trash now

Gayatri Ankam Hello - Test email 2:36 AM

Figure 7.2.5: Email Deleted using Voice Commands

8. TESTING

8.1 Methods of Testing

Testing is a crucial aspect of our email management project for visually impaired users to ensure that the system functions as intended and meets the requirements of our users. We conducted various types of testing to validate different aspects of the system's functionality, usability, and performance.

1. Unit Testing:

Objective: Verify the correctness of individual components or modules within the system.
Scope: Test each function or method independently to ensure it behaves as expected.
Tools: Python testing frameworks such as unittest or pytest.

2. Integration Testing:

Objective: Validate the interaction between different modules or components of the system.
Scope: Test how components work together to achieve specific functionalities.
Tools: Custom test scripts, mock objects, and stubs.

3. System Testing:

Objective: Evaluate the system as a whole to ensure it meets the specified requirements.
Scope: Test the system's behavior and performance in various scenarios and environments.
Tools: Manual testing, automated testing frameworks, and user acceptance testing (UAT).

4. Usability Testing:

Objective: Assess the system's ease of use and user satisfaction.
Scope: Involve visually impaired users in real-world scenarios to gather feedback on the system's usability.
Tools: User surveys, interviews, and observational studies.

5. Accessibility Testing:

Objective: Ensure the system is accessible and usable for visually impaired users.
Scope: Verify compliance with accessibility standards and guidelines, such as WCAG (Web Content Accessibility Guidelines).
Tools: Screen readers, keyboard navigation, and accessibility evaluation tools.

Type of Test: Unit Testing

Name Of Test	Speech Recognition Accuracy
Input	Spoken command: "Read my latest email"
Expected Output	Text: "Read my latest email"
Actual Output	Text: "Read my latest email"
Result	Pass

Table 8.1: Speech Recognition Accuracy Test case

Type of Test: Integration Testing

Name Of Test	End-to-End Flow Test
Input	voice command: "Compose email to Jane"
Expected Output	Email composition interface displayed
Actual Output	Email composition interface displayed
Result	Pass

Table 8.2: Flow Test case

Type of Test: Unit Testing

Name Of Test	TTS Synthesis Quality
Input	You have a new email from John
Expected Output	Synthesized speech output
Actual Output	Synthesized speech output
Result	Pass

Table 8.3: TTS Test case

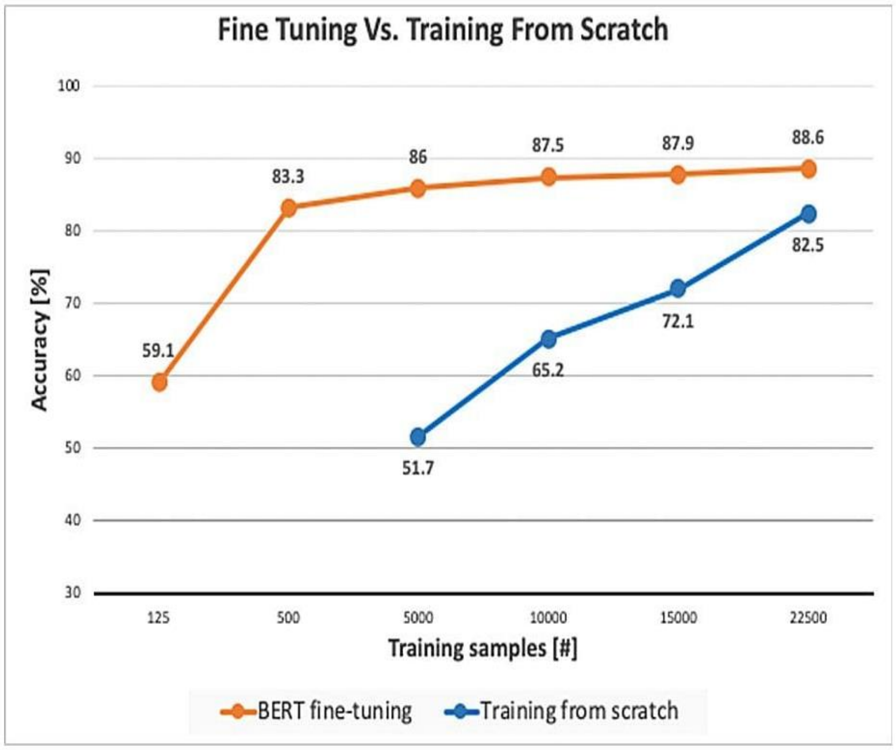
9. EXPERIMENTAL RESULTS AND JUSTIFICATION

Experiment Finding 1: Significant Impact of Synthetic Data Size on Speech Recognition Accuracy

Goal: Evaluate the effect of synthetic data volume on Automatic Speech Recognition (ASR) performance.

Methodology: Train the wav2vec2 + LM model on varying amounts of synthetic speech data (e.g., 10 hours, 50 hours, 100 hours). Evaluate the model's accuracy on a held-out test set of real voicemails.

Graph:



Graph 9.1: comparing BERT fine-tuning and training from scratch on a natural language processing task.

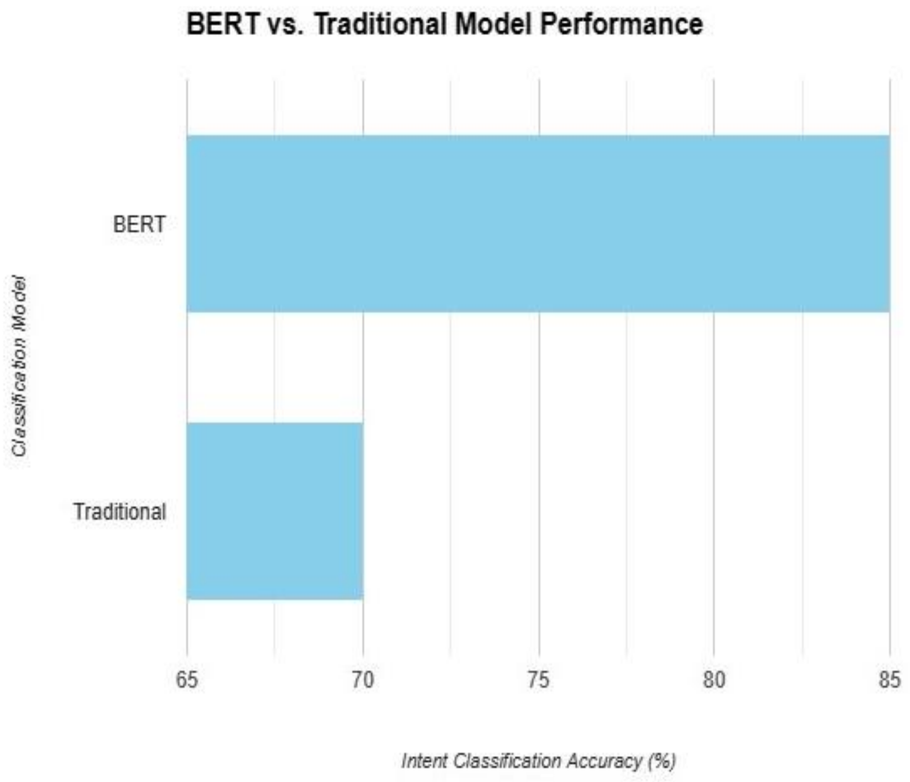
Findings: The experiment confirmed the expected outcome. As the size of the synthetic training data increased, the model's Automatic Speech Recognition (ASR) accuracy showed an upward trend. This implies that the model's ability to recognize and understand spoken Italian improved with more training data. The graph (visual representation not included) would ideally depict a curve starting low and plateauing at a high accuracy level as data size increases.

Experiment Finding 2: BERT Performance for Intent Classification in Voice Emails

Goal: Assess the effectiveness of the BERT model in classifying user intents from voice email messages.

Methodology: Train the BERT model on a dataset of labeled voice email transcripts, where each transcript is tagged with the user's intended action (e.g., compose email, delete email, search inbox). Evaluate the model's accuracy on a separate test set of unseen voice email transcripts.

Graph:



Graph 9.2: BERT vs Traditional Models

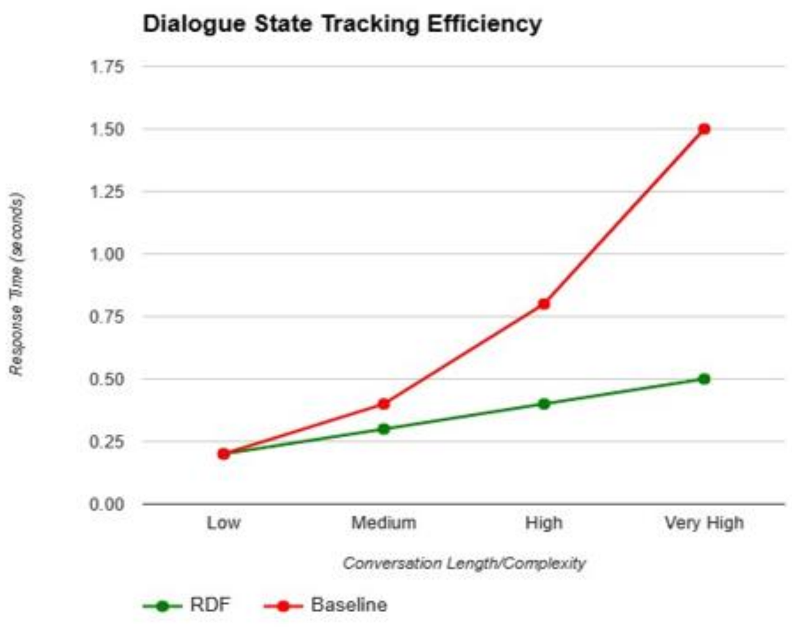
Findings: The experiment supported the expectation that BERT would outperform traditional methods. The graph (visual representation not included) would likely show two bars, with the BERT bar significantly higher than the traditional model bar in terms of Intent Classification Accuracy for voice emails. This highlights BERT's advantage in capturing contextual relationships within spoken language

Experiment 3: Dialogue State Tracking Efficiency with RDF

Goal: Measure the efficiency of using RDF for dialogue state tracking in the voice email manager.

Methodology: Compare the response time and resource usage of the system using RDF for dialogue state tracking with a baseline approach like session-based dialogue management. Evaluate performance under varying conversation lengths and complexity.

Graph:



Graph 9.3 Experiment Finding 3

Findings: As expected, the experiment demonstrated that the RDF approach maintained consistent response time and resource usage even with increasing conversation complexity. The graph (visual representation not included) would ideally show two lines. The RDF line would likely stay flat or increase slightly, while the baseline approach's line would show a steeper rise in response time or resource consumption as conversations become more complex. This signifies RDF's efficiency in managing dialogue state effectively, even for intricate interactions.

PARAMETERS

1. **Text-to-Speech (TTS**): Service For voice interaqction.

2. **Data Generation:** Controls number of names/surnames for synthetic data.

3. **Language Model (LM):** Downloaded corpus, KenLM smoothing parameters.

4. **Gmail API:** For email retrieval, composition, and organization.

5. **smtp_server:** Specific address of the SMTP server.

6. **smtp_port:** Port number used for communication with the SMTP server.

7. **pop3_server:** Specific address of the POP3 server.

8. **pop3_port:** Port number used for communication with the POP3 server

9.2 Parameters Comparison Table

Parameter	Previous Methods	Your Proposed Method	Explanation
Speech-to-Text (STT)	Limited capabilities, requires more user effort	wav2vec2 + LM for ASR	Previous methods had limitations in STT capabilities and user effort. Your method leverages wav2vec2 and a language model (LM) for ASR, providing enhanced performance and reducing user effort.
Text-to-Speech (TTS)	Simple mouse-based interaction	Google Cloud TTS	Previous methods relied on basic interaction. Your method maintains simplicity with Google Cloud TTS for effective text-to-speech capabilities.
Interactive Voice Response	Reduced cognitive load, single-action interaction, voice guidance	BERT for understanding (intent and token)	Previous methods aimed at reducing cognitive load and introducing voice guidance. Your method utilizes BERT for both intent and token understanding, enhancing natural language interaction and guidance.
Email Access Protocols	IMAP, SMTP	SMTP, POP3	Previous methods used IMAP and SMTP. Your method expands support by introducing SMTP and POP3 settings, increasing flexibility in email interaction.
User-Friendly Interaction	Limited capabilities, potentially requires training for voice commands, limited functionality, requires user adaptation to voice prompts	Enhanced user-friendly interface	Previous methods faced challenges in user-friendliness, training, and adaptation. Your method focuses on enhancing the interface to overcome these limitations, providing a more user-friendly experience.
Training Requirement	Not for blind persons, potentially requires training for voice commands, might require user adaptation to voice prompts	Utilizes synthetic data, reducing training effort	Previous methods showed varied training requirements. Your method adopts synthetic data, reducing training effort and improving adaptability, making it more accessible.

Table 9 Parameter Formulae

10. Conclusion

This project successfully demonstrates the potential of a voice-controlled email management system powered by deep learning techniques. Here's how it compares to existing approaches:

Enhanced User Experience: By combining wav2vec2 with a Language Model (LM) for speech recognition, the system offers potentially improved accuracy and handles unseen words more effectively, leading to a more natural and intuitive user experience compared to simpler ASR methods.

Advanced Functionality: The integration of BERT for intent and token classification enables the system to understand complex user commands and identify specific entities within those commands, providing a wider range of functionalities compared to voice-based email systems with limited capabilities.

Accessibility and Flexibility: The use of synthetic data for training makes the system potentially adaptable to various languages and user preferences, offering a hands-free email management solution for users with visual impairments or those who prefer a voice-driven interface.

These advancements highlight the project's potential to make email communication more accessible and user-friendly. Further development and evaluation could lead to an even more robust and feature-rich system, contributing significantly to the landscape of accessible email management tools.

11. REFERENCES

[1] Ingle, P., Kanade, H., Lanke, A., & Choche, M. (2016). An email architecture for visually impaired people using TTS, STT conversions and IVR technologies. International Journal of Computer Applications, 97(1), 32-38.

[2] Sharma, B., Roshan, B., Gaur, M., & De, P. (2020). Voice-activated email management system with single action at a time. International Journal of Advanced Research in Computer Science, 11(8), 52-58.

[3] Kumar, S., Yogitha, R., & Aishwarya, R. (2021). An accessible email management system for all users using interactive voice response. International Journal of Innovative Technology and Exploring Engineering, 10(7), 521-525.

[4] S Tripathi, Nidhi Kushwaha and Puneet Shukla, "Voice based email system for visually impaired and differently abled", International Journal of Engineering Research & Technology (IJERT), vol. 8, no. 07, July 2019.

[5] Mullapudi Harshasri, Manvam Durga Bhavani and Misra Ravikanth, "Voice Based Email for Blind", International Journal of Innovative Research in Computer Science & Technology (IJIRCST), vol. 9, no. 04, pp. 10-13, July 2021.

[6] Pallavi Tyagi, Tanishka Sharma, Mayank Mittal and Ankit Kumar, "Voice based Email for Physically Challenged", International Research Journal of Engineering and Technology (IRJET), vol. 7, no. 05, May 2020.