

**AN AI ASSISTED ACCESSIBILITY
FOR VISUALLY IMPAIRED
PEOPLE**

A DESIGN PROJECT REPORT

Submitted by

THEJEAL SRI.K

SUJAINITHA.G

DHIVYA.A. D

SATHVIKA.A

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, NewDelhi)

SAMAYAPURAM-621112

JUNE 2024

**K. RAMAKRISHNAN COLLEGE OF
TECHNOLOGY(AUTONOMOUS)
SAMAYAPURAM–621112
BONAFIDE CERTIFICATE**

Certified that this design project report titled “AN AI ASSISTED ACCESSIBILITY FOR VISUALLY IMPAIRED PEOPLE” is the bonafide work of **THEJEAL SRI.K (REG NO: 811721243057) SUJAINITHA.G (REGNO: 811721243055) DHIVYA.A. D (REG NO: 811721243014) SATHVIKA.A(REG NO:811721243049)** who carried out the project under my supervision.

SIGNATURE

Dr.T.Avudaiappan, M.E., Ph.D.,
HEAD OF THE DEPARTMENT

Associate Professor
Department of Artificial Intelligence
K.Ramakrishnan College of Technology
(Autonomous)
Samayapuram – 621112.

SIGNATURE

Mr.R. Roshan Joshua, M.E.,
SUPERVISOR

Assistant Professor
Department of Artificial Intelligence
K.Ramakrishnan College of Technology
(Autonomous)
Samayapuram – 621112.

Submitted for the viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We jointly declare that the project report on “**AN AI ASSISTED ACCESSIBILITY FOR VISUALLY IMPAIRED PEOPLE**” is the result of original work done by us and best of our knowledge, similar work has not been submitted to “**ANNA UNIVERSITY CHENNAI**” for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This design project report is submitted on the partial fulfilment of the requirement of the award of Degree of **BACHELOR OF TECHNOLOGY**.

SIGNATURE

THEJEAL SRI.K

SUJAINITHA.G

DHIVYA.A. D

SATHVIKA.A

PLACE : SAMAYAPURAM

DATE :

ACKNOWLEDGEMENT

It is with great pride that we express our gratitude and in - debt to our institution “**K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)**”, for providing us with the opportunity to do this project.

We are glad to credit honorable chairman **Dr. K. RAMAKRISHNAN, B.E.**, for having provided for the facilities during the course of our study in college.

We would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA., Ph.D.**, for forwarding to our project and offering adequate duration in completing our project.

We would like to thank our principal **Dr. N. VASUDEVAN, M.E., Ph.D.**, who gave opportunity to frame the project the full satisfaction.

We whole heartily thanks to **Dr. T. AVUDAIAPPAN, M.E., Ph.D.**, HEAD OF THE DEPARTMENT, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

I express my deep and sincere gratitude to my project guide **Mr. R. ROSHAN JOSHUA M.E.**, ASSISTANT PROFESSOR, **ARTIFICIAL INTELLIGENCE** for his incalculable suggestions, creativity, assistance and patience which motivated me to carry out the project successfully.

I render my sincere thanks to my project coordinator **Mrs. G. NALINA KEERTHANA M.E.**, ASSISTANT PROFESSOR, **ARTIFICIAL INTELLIGENCE** other faculties and non-teaching staff members for providing valuable information during the course. I wish to express my special thanks to the officials & Lab Technicians of our departments who rendered their help during the period of the work progress.

ABSTRACT

An advanced mobile application is designed to revolutionize accessibility for visually impaired individuals by integrating cutting-edge deep learning techniques, including Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. The app responds to voice commands, capturing screen content and processing it through CNNs to extract visual information, which LSTM networks use to generate automatic captions converted into real-time auditory feedback, empowering users to navigate digital interfaces confidently. Extensive optimization ensures high accuracy and reliability across diverse environments, showcasing deep learning's pivotal role in enhancing accessibility and setting new standards in assistive technology. Traditional assistive technologies often lack the real-time and intuitive capabilities needed for seamless interaction, highlighting the significance of this project in addressing such challenges. The methodology integrates CNNs and LSTMs within a responsive app framework, providing efficient assistance across various scenarios. Through rigorous testing, the app demonstrates superior performance, allowing users to access content and interact with applications independently. Deep learning-driven approach ensures adaptability and robustness, contributing to ongoing advancements in assistive technologies for visually impaired individuals.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF FIGURES	ix
	LIST OF ABBREVIATIONS	x
1	INTRODUCTION	1
1.1	BACKGROUND	1
1.2	PROBLEM STATEMENT	2
1.3	AIM & OBJECTIVE	2
	1.3.1 Aim	2
	1.3.2 Objective	2
2	LITERATURE SURVEY	3
2.1	AUTOMATIC IMAGE AND VIDEO CAPTION GENERATION WITH DEEP LEARNING: A CONCISE REVIEW AN ALGORITHMIC OVERLAP.	3
2.2	SWITCHING TEXT-BASED IMAGE ENCODERS FOR CAPTIONING IMAGES WITH TEXT.	4
2.3	A STUDY OF CONVNEXT ARCHITECTURES FOR ENHANCED IMAGE CAPTIONING.	5
2.4	CROSS LINGUAL VOICE CONVERSION WITH CONTROLLABLE SPEAKER INDIVIDUALITY USING VARIATIONAL AUTOENCODER AND STAR GENERATIVE ADVERBIAL NETWORK.	6

2.5	IMAGE CAPTIONING MODEL USING PARTS OF SPEECH GUIDANCE MODULE FOR DESCRIPTION WITH DIVERSE VOCABULARY.	7
3	SYSTEM ANALYSIS	8
3.1	EXISTING SYSTEM	8
3.1.1	Drawbacks	10
3.2	PROPOSED SYSTEM	11
3.2.1	Advantages	13
4	SYSTEM SPECIFICATION	14
4.1	HARDWARE SYSTEM SPECIFICATION	14
4.2	SOFTWARE SYSTEM SPECIFICATION	14
4.3	SOFTWARE DESCRIPTION	14
4.3.1	Library	14
4.3.2	Developing Environment	15
5	ARCHITECTURAL DESIGN	17
5.1	SYSTEM DESIGN	17
5.2	DATA FLOW DIAGRAM	18
5.3	USE CASE DIAGRAM	19
5.4	ACTIVITY DIAGRAM	20

5.5	SEQUENCE DIAGRAM	21
6	MODULE DESCRIPTION	22
6.1	MODULES	22
6.1.1	Voice Command	22
6.1.2	Image Capturing	23
6.1.3	Textual Annotation	23
6.1.4	Verbal Narration	24
7	CONCLUSION AND FUTURE ENHANCEMENT	25
7.1	CONCLUSION	25
7.2	FUTURE ENHANCEMENT	25
	APPENDIX 1 SOURCE CODE	27
	APPENDIX 2 SCREENSHOTS	35
	REFERENCES	38

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
3.1	Design Phases of Algorithm	11
5.1	Architectural Design	17
5.2	Data Flow Diagram	18
5.3	Use Case Diagram	19
5.4	Activity Diagram	20
5.5	Sequence Diagram	21
6.1	Phases of Proposed System	22
A.2.1	App Activation in Power Shell	35
A.2.2	Voice Recognition	35
A.2.3	Epoch Training	36
A.2.4	Bleu Scores	36
A.2.5	Image Caption Generation	37

LIST OF ABBREVIATIONS

ML	Machine Learning
RNN	Recurrent Neural Network
CNN	Convolutional Neural Networks
Bi-LSTM	Bidirectional Long Short-Term Memory
LSTM	Long Short-Term Memory
DBN	Deep Belief Networks

CHAPTER 1

INTRODUCTION

In the realm of assistive technologies, the integration of advanced deep learning techniques like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks has become pivotal. This project focuses on developing a mobile application that responds to voice commands, utilizing CNNs and LSTMs to provide real-time assistance to visually impaired individuals. By capturing screen content and generating automatic captions converted into auditory feedback, the app aims to enhance accessibility and independence. Through rigorous optimization and testing, this project aims to set new standards in assistive technology, showcasing the transformative potential of deep learning in revolutionizing accessibility for the visually impaired.

1.1 BACKGROUND

The integration of deep learning, particularly Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, has revolutionized assistive technologies for visually impaired individuals. Hand gesture recognition, a subset of computer vision, has emerged as a natural and intuitive mode of interaction in various domains, including virtual reality and robotics. CNNs excel in image recognition tasks, while LSTMs are adept at generating natural language captions, making them ideal for developing assistive apps. Ensemble learning techniques further enhance accuracy and robustness in hand gesture recognition systems. This background sets the foundation for developing an app that responds to voice commands, captures screen content, generates automatic captions, and converts them into voice format, enhancing accessibility and independence for the visually impaired.

1.2 PROBLEM STATEMENT

The challenge in developing an app for assisting visually impaired individuals lies in creating a seamless and accurate system that responds to voice commands, captures screen content, generates automatic captions, and converts them into real-time auditory feedback. Achieving high accuracy and reliability in automatic caption generation and voice conversion, especially in diverse environmental conditions, presents a significant challenge. Current solutions often face difficulties in accurately interpreting screen content and generating natural-sounding voice feedback. Leveraging advanced deep learning models like Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks is crucial to address these challenges, ensuring efficient and effective assistance for visually impaired users in navigating digital interfaces.

1.3 AIM AND OBJECTIVE

1.3.1 AIM

Develop an AI-driven accessibility app with CNNs and LSTM for real-time, accurate responses, adaptable to various environments, and optimized for efficiency.

1.3.2 OBJECTIVE

Optimize gesture recognition using CNNs and ensemble learning for accuracy, robustness, and real-time performance. Ensure reliability, adaptability, and scalability for practical applications like human-computer interaction.

CHAPTER 2

LITERATURE SURVEY

2.1 AUTOMATIC IMAGE AND VIDEO CAPTION GENERATION WITH DEEP LEARNING: A CONCISE REVIEW AND ALGORITHMIC OVERLAP.

Author

Soheyla Amirian, Khaled Rasheed, Thiab R. Taha, Hamid R. Arabnia

Year of Publication: 2020

Algorithm Used

CNNs for feature extraction, RNNs/LSTM/GRU for sequence modeling and text generation.

Abstract

Deep Learning methodologies for image and video captioning generate automatic descriptions for visually impaired users, create metadata for search engines, and enhance robot vision systems. These techniques offer significant benefits for various applications, including indexing, accessibility, and task-specific automation.

This article provides a concise review of deep learning-based captioning methods.

Merit

Accurate captions generation.

Demerit

Requires substantial computational resources and data.

2.2 SWITCHING TEXT-BASED IMAGE ENCODERS FOR CAPTIONING IMAGES WITH TEXT.

Author

Arisa Ueda, Wei Yang, Komei Sugiura

YEAR of Publication: 2023

Algorithm Used

Multimodal transformer with four image-related modalities, enriched using pre-trained Contrastive Language-Image Pre-training (CLIP) models and two additional attention models.

Abstract

This study addresses the TextCaps task, which involves generating textual descriptions of images by integrating optical character recognition (OCR) with traditional image captioning. To enhance performance, we utilize multiple modalities, enriching image and OCR features with pre-trained CLIP models, and employ two additional attention models in a transformer architecture to strengthen image representation.

Merit

Enhanced TextCaps performance through multimodal transformer integration and pre-trained model enrichment.

Demerit

Increased complexity and computational requirements for integration and training.

2.3 A STUDY OF CONVNEXT ARCHITECTURES FOR ENHANCED IMAGE CAPTIONING.

Author

Leo Ramos, Edmundo Casas, Cristian Romero.

Year of Publication: 2024

Algorithm Used

ConvNeXt model with LSTM and visual attention for image captioning.

Abstract

This study evaluates the ConvNeXt model for image captioning, integrating it with an LSTM and visual attention module. Using the MS COCO 2014 dataset, we tested various ConvNeXt versions, learning rates, and the effect of teacher-forcing. ConvNeXt showed notable performance improvements, outperforming benchmarks by 43.04% (soft-attention) and 39.04% (hard-attention) in BLEU-4 scores, and surpassing vision transformers and data-efficient image transformers by 4.57% and 0.93%, respectively.

Merit

Significant performance enhancements in accuracy and loss metrics.

Demerit

Potential complexity in integrating different components and computational demands.

2.4 CROSS LINGUAL VOICE CONVERSION WITH CONTROLLABLE SPEAKER INDIVIDUALITY USING VARIATIONAL AUTOENCODER AND STAR GENERATIVE ADVERSIAL NETWORK.

Author

Tuan Vu Ho, Masato Akagi

Year of Publication: 2021

Algorithm Used

Non-parallel cross-lingual voice conversion (CLVC) model with VAE and Star GAN.

Abstract

This paper introduces a non-parallel cross-lingual voice conversion (CLVC) model with VAE and Star GAN for voice mimicry and speaker individuality control. It also includes an F0 injection method to improve F0 modeling in cross-lingual settings.

Adversarial training mitigates over-smoothing issues, showcasing effectiveness in both objective and subjective evaluations.

Merit

Enables voice mimicry with controlled speaker individuality and improved F0 modeling.

Demerit

Complexity in integrating VAE and Star GAN, requiring careful parameter tuning.

2.5 IMAGE CAPTIONING MODEL USING PARTS OF SPEECH GUIDANCE MODULE FOR DESCRIPTION WITH DIVERSE VOCABULARY. Author

Ju-Won Bae, Soo-Hwan Lee, Won-Yeol Kim, Ju-Won Bae

Year of Publication: 2022

Algorithm Used

Part-Of-Speech (POS) guidance module and multimodal-based image captioning model incorporating Bi-LSTM and a multimodal layer.

Abstract

This paper introduces a multimodal-based image captioning model with a Part-Of-Speech (POS) guidance module to enhance lexical diversity in deep learning (DL) captioning. The POS module controls image and sequence information based on predicted POS guidance for richer expression. By integrating POS and Bi-LSTM output via a multimodal layer, the model predicts captions while considering grammatical structure.

Merit

Enhanced lexical diversity in image captions.

Demerit

Potential complexity in integrating POS guidance and multimodal features, requiring careful tuning and increased computational resources during training.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The presented system is designed to analyze and summarize text content, focusing on images to generate descriptive captions. It employs advanced deep learning techniques like bidirectional LSTM (BiLSTM) for understanding textual information and a deep belief network (DBN) for summarizing the text effectively. The integration of BiLSTM and DBN enables accurate information retrieval and concise text summarization. However, it's important to note that this system specifically caters to processing visual data and doesn't include functionalities for voice commands or audio output. The novelty of this approach lies in its ability to efficiently generate image descriptions based on the content within the images, showcasing advancements in deep learning for text summarization and image captioning tasks.

ALGORITHM USED

- **BIDIRECTIONAL LONG SHORT-TERM MEMORY (Bi-LSTM)** Bidirectional Long Short-Term Memory (BiLSTM) plays a pivotal role in the effectiveness of this system for automated information retrieval and text summarization. One key advantage of BiLSTM is its ability to process input sequences in both forward and backward directions simultaneously. This bidirectional processing captures a more comprehensive understanding of the textual content by considering the context from preceding and succeeding words or sentences. As a result, BiLSTM can grasp the nuances and dependencies within the text, leading to more accurate information extraction and summary generation.

By leveraging BiLSTM, the system gains a deeper insight into the semantic structure of the text, enabling it to identify important information and extract meaningful content for summarization. The bidirectional nature of BiLSTM allows the model to capture long-range dependencies and contextual cues that may be crucial for generating concise and informative summaries. This enhances the overall quality and relevance of the generated summaries, making them more useful for users seeking condensed yet comprehensive insights from large volumes of text data.

Furthermore, BiLSTM capability to handle bidirectional sequences is particularly beneficial in scenarios where context plays a vital role in understanding the meaning of the text. Whether it's analyzing complex documents or summarizing articles with intricate information, BiLSTM's bidirectional processing ensures that the system can effectively navigate through the text and produce accurate and coherent summaries. Thus, the incorporation of BiLSTM in this system significantly contributes to its ability to perform robust information retrieval and text summarization tasks with enhanced accuracy and contextual understanding.

poses.

- **DEEP BELIEF NETWORK(DBN)**

The Deep Belief Network (DBN) is a critical component in the architecture of this system, particularly for its prowess in text summarization. DBN's strength lies in its ability to learn intricate patterns and hierarchical representations within textual data. This hierarchical learning approach allows DBN to extract complex structures and relationships embedded within the text, enabling it to generate concise and meaningful summaries. One of the key advantages of DBN is its capacity to discern multiple levels of abstract features from the input text. This hierarchical learning process enables DBN to identify salient information and prioritize essential content for summarization accurately.

3.1.1 Drawbacks

The existing assistive technology faces limitations in real-time responsiveness due to manual processes, lacks advanced deep learning integration, and struggles with adaptability across diverse digital interfaces. Additionally, it encounters challenges in seamless navigation and accurate interpretation of complex content. As technology evolves, meeting the dynamic needs of visually impaired users remains a crucial area for improvement.

3.2 PROPOSED SYSTEM

This project introduces an innovative mobile app for visually impaired users, leveraging CNNs and LSTMs for real-time screen content analysis and automatic caption generation. The app responds to voice commands, providing auditory feedback for confident digital interaction. Extensive optimization ensures high accuracy, setting new standards in assistive technology for seamless accessibility. Overall, this deep learning-driven system empowers independent navigation of digital interfaces with inclusivity and reliability.

ALGORITHM USED

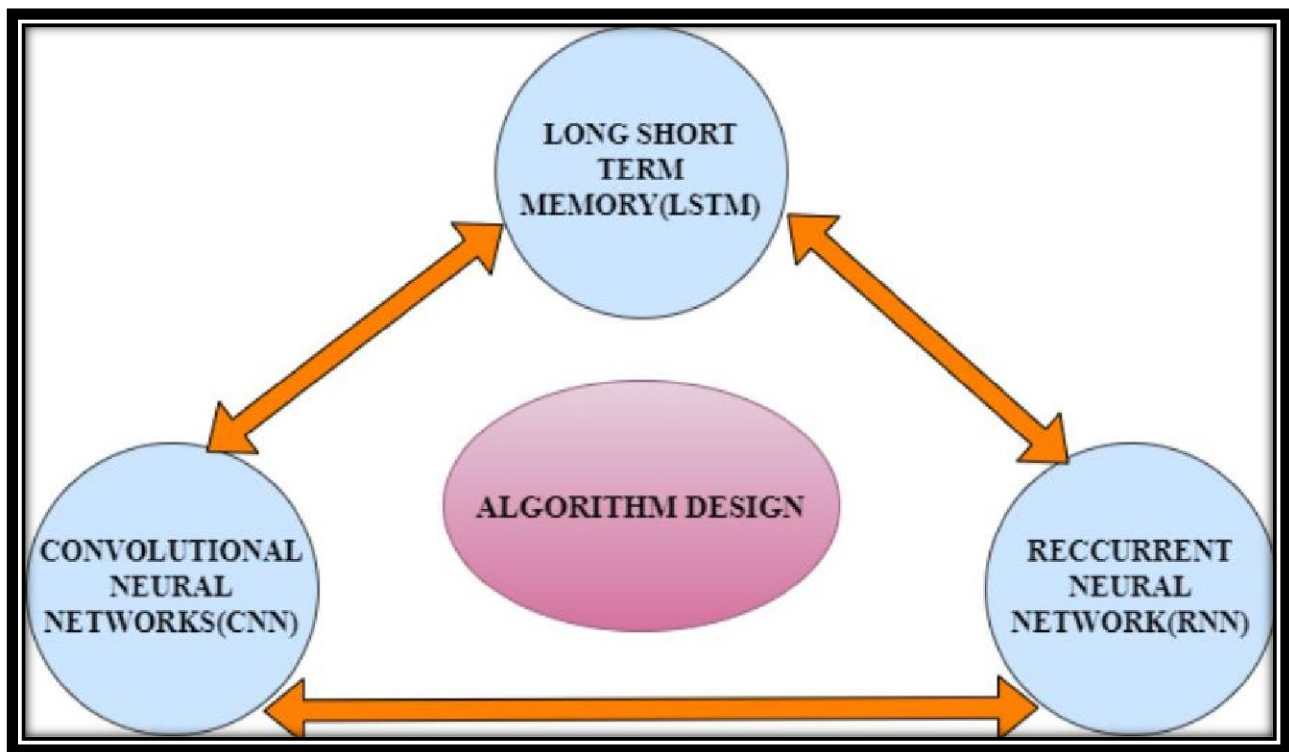


Figure No.3.2.1. Algorithm Phase

LSTM ALGORITHM

Long Short-Term Memory (LSTM) is crucial in the proposed system for its ability to retain and process sequential information effectively. By capturing dependencies over time, LSTM enhances the accuracy of automatic caption generation and real-time auditory feedback. Its capability to handle long-range dependencies ensures coherent and contextually relevant output, improving user experience. Overall, LSTM plays a pivotal role in enabling intuitive and efficient digital interaction for visually impaired individuals in diverse environments.

RNN ALGORITHM

Recurrent Neural Networks (RNNs) are vital in the proposed system for their sequential data processing capabilities. They enable the model to capture temporal dependencies, enhancing the accuracy of automatic caption generation and real-time auditory feedback. RNNs' ability to retain context from previous inputs ensures coherent and meaningful output, contributing to a seamless user experience. Overall, RNNs play a key role in facilitating intuitive and efficient digital interaction for visually impaired users across various scenarios.

CNN ALGORITHM

The Convolutional Neural Network algorithm is pivotal in this project for assisting visually impaired individuals due to its unparalleled ability in accurate feature extraction from images. By automatically learning hierarchical representations of visual data, CNNs capture intricate details crucial for precise recognition of hand gestures and screen content, ensuring robust performance across diverse environmental conditions. Their computational efficiency enables quick processing of image data, facilitating real-time applications like automatic screen capture and caption generation triggered by voice commands. Additionally, CNNs adapt well to new gestures or unseen data

patterns, making the system adaptable to evolving user needs and expanding gesture classes without extensive manual annotation of training data. Leveraging CNNs in this project enhances the overall user experience, providing a seamless and accurate assistive technology solution.

3.2.1 Advantages

This advanced assistive technology integrates deep learning techniques (such as CNNs and LSTMs) for precise screen analysis and real-time caption generation. It ensures high accuracy, adapts seamlessly to various digital interfaces, and enhances user experience through intuitive navigation and immediate auditory feedback. Additionally, it promotes inclusivity by empowering visually impaired users and drives ongoing innovation in accessibility technology.

CHAPTER 4

SYSTEM SPECIFICATION

4.1 HARDWARE SYSTEM SPECIFICATION

- **Computer** - minimum of 4GB RAM & dual-core processor.
- **Stable internet connection.**
- **Storage.**

4.2 SOFTWARE SYSTEM SPECIFICATION

- **Python programming language** - Python 3.x installed on the computer/server.
- **Operating system** - Windows, Linux, or macOS.
- **Python libraries** such as – NumPy, pandas, tensor flow, Keras, NLTK.
- **HTML/CSS or JavaScript** - for UI design.

4.3 SOFTWARE DESCRIPTION

Innovative mobile app for visually impaired users. Triggered by voice commands, captures and captions screen content. Utilizes CNNs and LSTM networks for image-to-text and audio conversion. Enhances accessibility and inclusivity in digital interactions. The app's intuitive interface and real-time feedback contribute to a seamless user experience, empowering visually impaired individuals in their digital interactions.

4.3.1 Library

To develop the AI assisted app for visually impaired people, the following libraries are commonly used:

- **NumPy:** NumPy is a fundamental library for numerical computations in Python. It provides efficient numerical operations and arrays, which are essential for processing and manipulating image data.
- **TensorFlow:** TensorFlow are deep learning (DL) framework commonly used for training and deploying machine learning models. They offer a range of tools and functions for building and training neural networks, including models for gesture recognition.
- **Pandas:** Pandas is vital for organizing screen content, managing captions, and optimizing app functionalities for visually impaired users. It ensures data accuracy and advances assistive technology standards.
- **Keras:** Keras is fundamental for developing deep learning models, particularly CNNs and LSTMs for image-to-text conversion and natural language processing in the visually impaired app. It enables the creation of advanced models crucial for accurate captioning and real-time assistance.
- **Pickle:** Pickle is crucial for serializing and deserializing Python objects, allowing efficient storage and retrieval of model parameters and processed data in the visually impaired app. It ensures data persistence and seamless model deployment.
- **NLTK:** NLTK (Natural Language Toolkit) is essential for text processing tasks, including tokenization and part-of-speech tagging, enhancing the app's natural language processing capabilities. It enables advanced linguistic analysis crucial for accurate captioning and user interaction.

4.3.2 Developing environment

To develop the AI assisted app for visually impaired people, you would typically setup the following environment:

- **Python:** Python is the primary programming language used for developing the system. Ensure that Python is installed on your system.
- **Integrated Development Environment (IDE):** Choose an IDE for Python development, such as PyCharm, Visual Studio Code, or Jupiter Notebook. These IDEs provide features like code editing, debugging, and project management, enhancing the development process.
- **Install Required Libraries:** Use the Python package manager, pip, to install the necessary libraries such as NumPy, TensorFlow. You can install them using the command line interface or directly within your IDE.
- **File Structure:** Organize your project files and folders. Typically, you would have directories for storing face images, trained models, configuration files.
- **Database Integration:** Integrate a database system to store attendance records, user information, and any other necessary data. Set up the database connection and create the required tables and schemas.
- **User Interface Design:** Design and develop the user interface using HTML, CSS, and JavaScript.
- **Testing and Deployment:** Test your application thoroughly, checking for any bugs or issues. Once the testing phase is complete, you can deploy the application to a webserver or cloud platform for online access.
- **Database Integration:** Integrate a database system to store attendance records, user information, and any other necessary data., and JavaScript.
- **Testing and Deployment:** Test your application thoroughly, checking for any bugs or issues. Once the testing phase is complete, you can deploy the application to a web server or corm for online access.

CHAPTER

ARCHITECTURAL DESIGN

5.1 SYSTEM DESIGN

A system architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

The Voice Command Triggered Image Captioning System enhances accessibility for the visually impaired. It employs deep learning to process voice commands, capture screen content, and generate real-time audio descriptions. This system facilitates effortless interaction with digital interfaces.

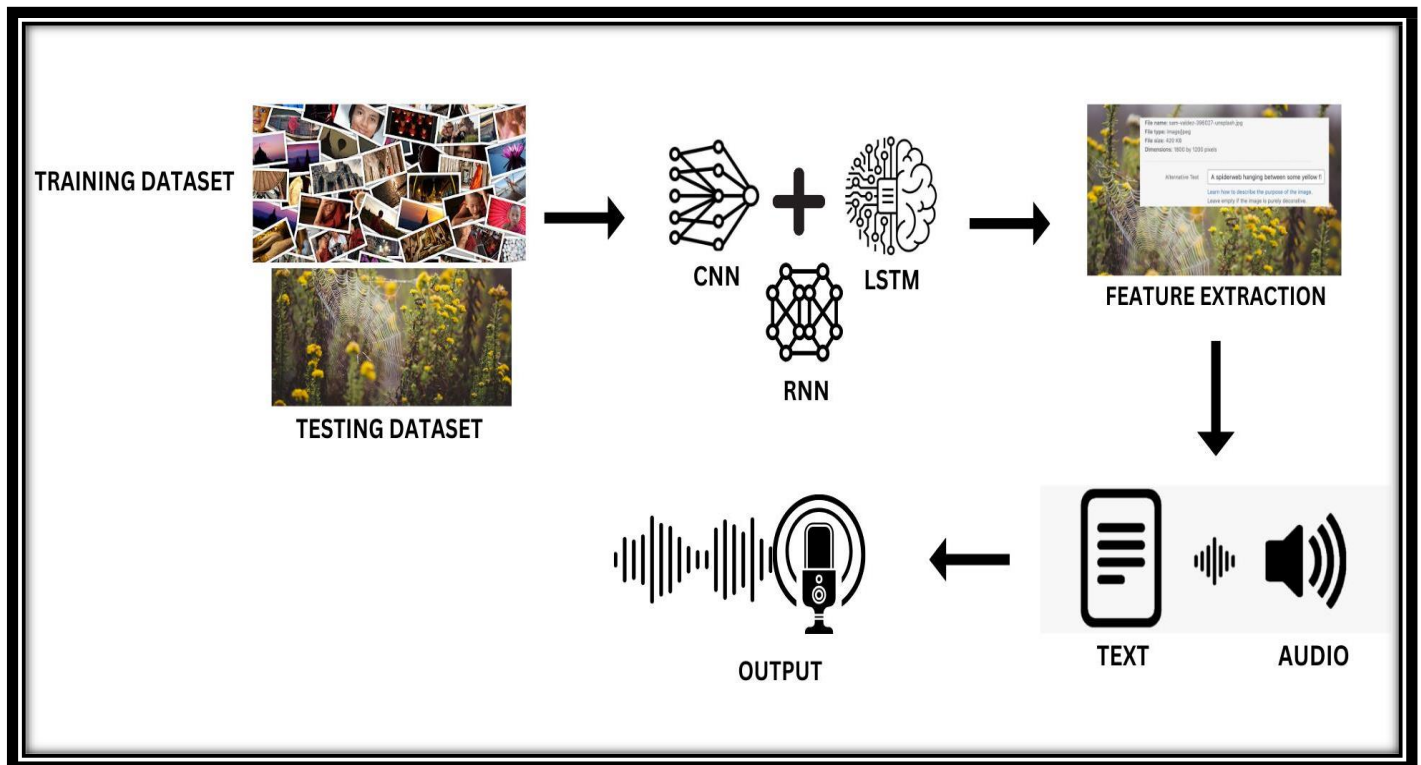


Figure No.5.1. Architectural Diagram

5.2 DATA FLOW DIAGRAM

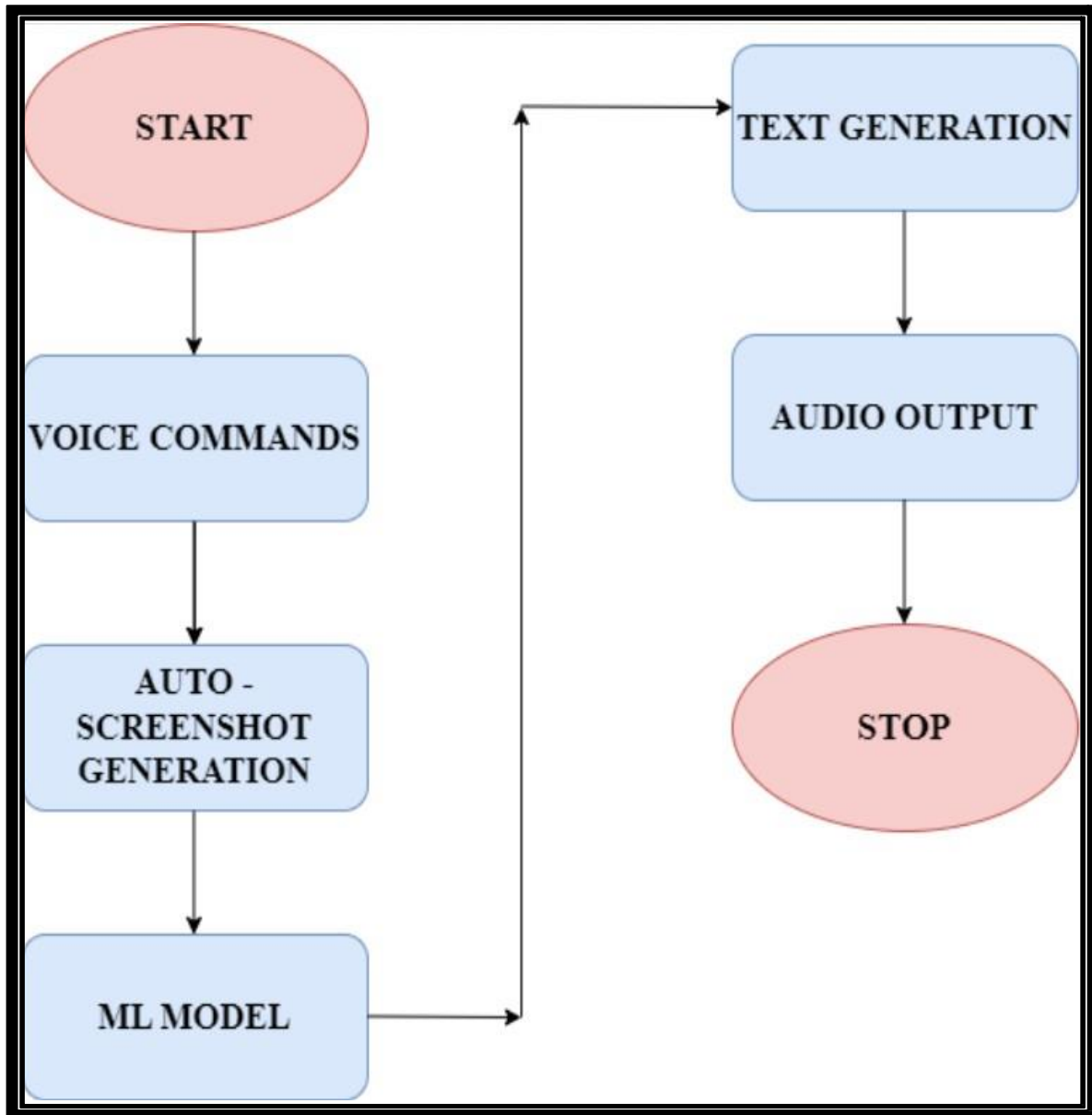


Figure No.5.2.Data Flow Diagram

5.3 USE CASE DIAGRAM

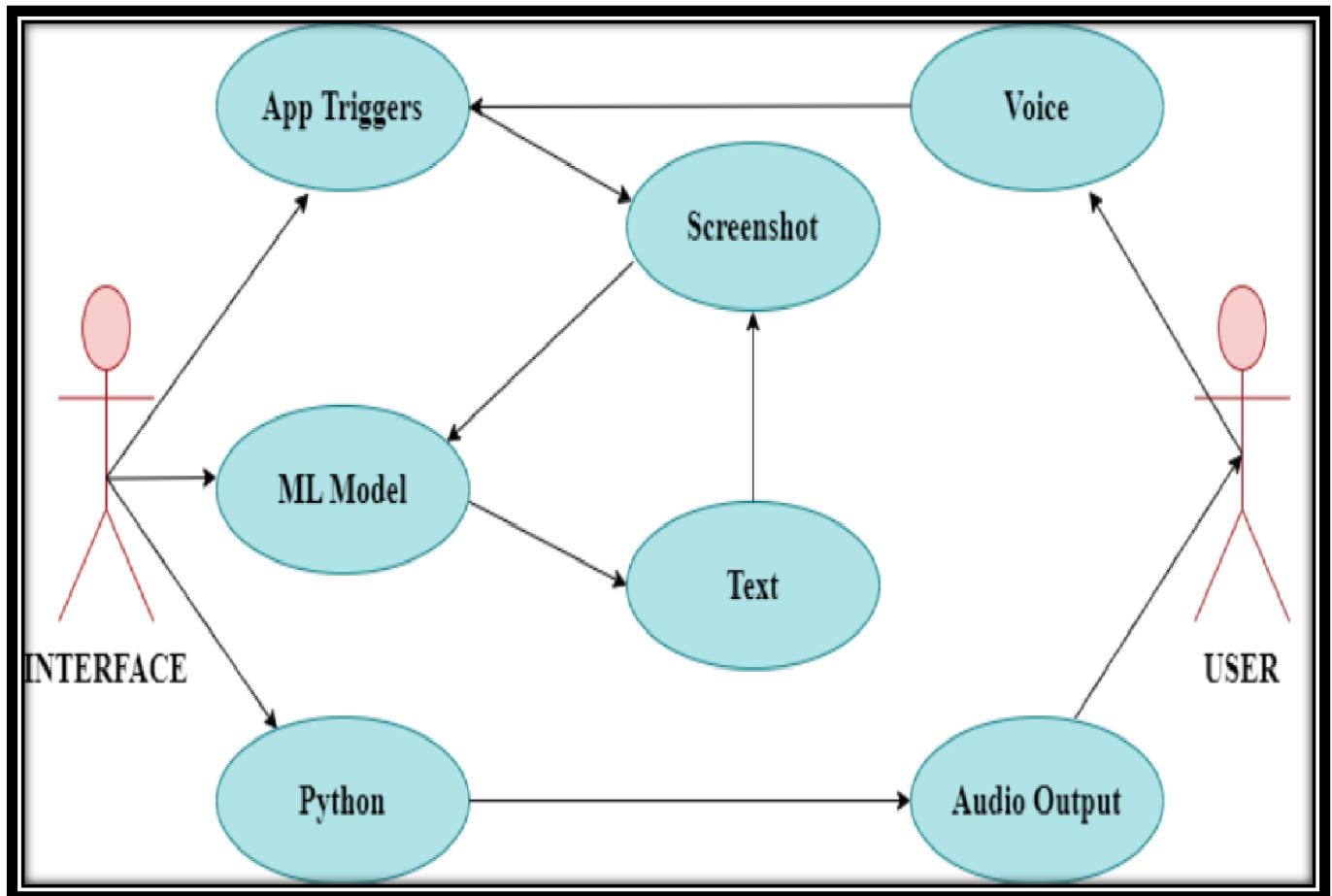


Figure No.5.3. Use Case Diagram

5.4 ACTIVITY DIAGRAM

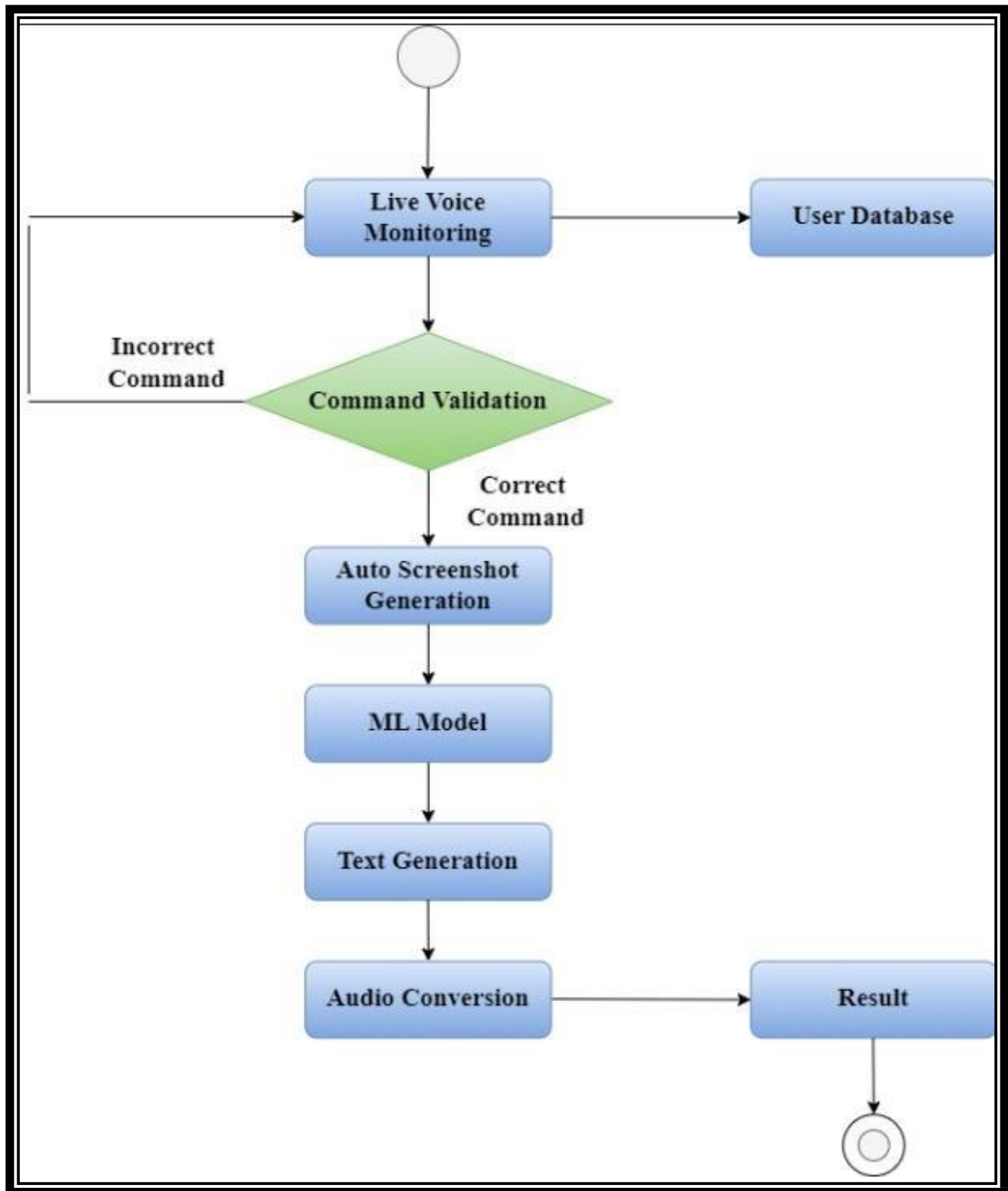


Figure No.5.4. Activity Diagram

5.5 SEQUENCE DIAGRAM

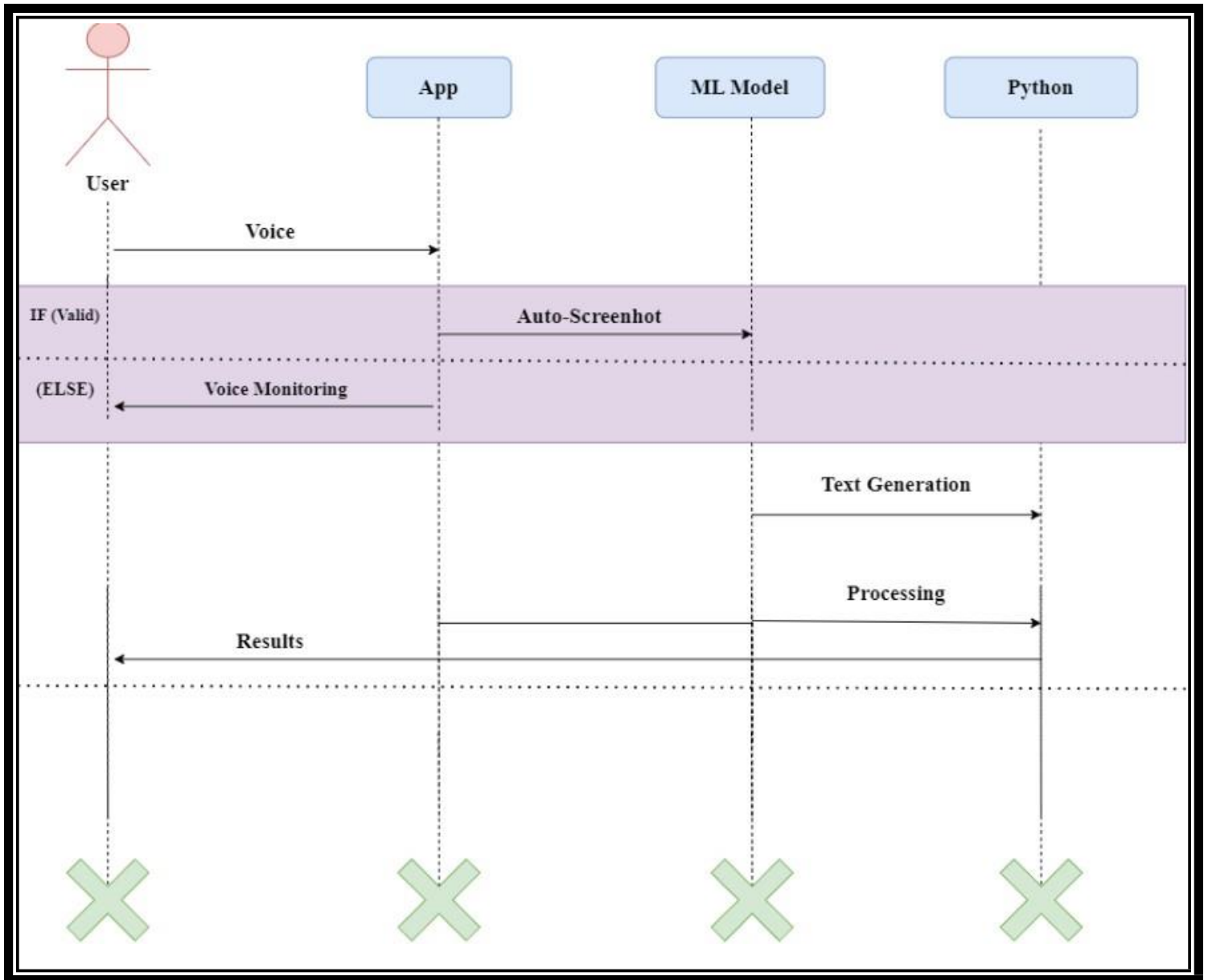


Figure No.5.5. Sequence Diagram

CHAPTER 6

MODULE DESCRIPTION

6.1 MODULES

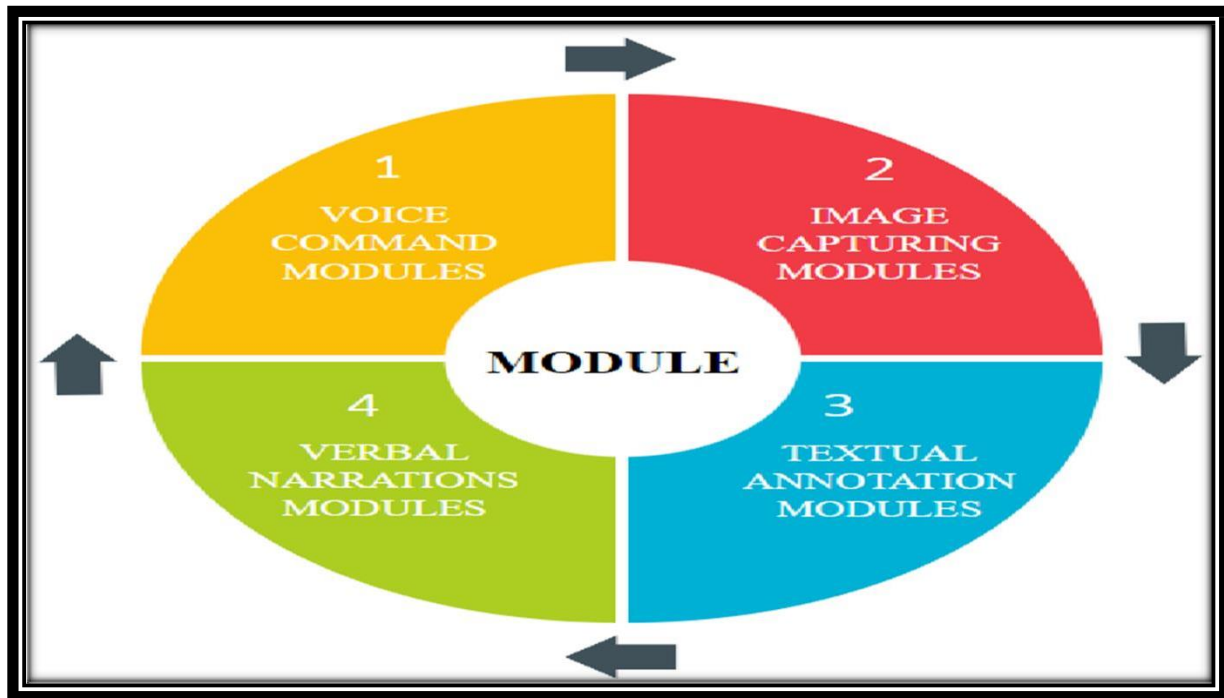


Figure No.6.1. Phases of Proposed System

6.1.1 Voice command

The Voice Command Module utilizes speech recognition for user commands, enhancing accessibility and user experience through seamless interaction. This module can be implemented using HTML, CSS and PYTHON.

- **Voice Command Processing:** Recognizes user commands, executes actions, provides feedback, and adapts to user preferences for personalized interactions.
- **Screenshot Automation:** Automatically captures screen, integrates voice commands, provides feedback, and allows users to customize screenshot settings for their needs.

- **Accessibility and Optimization:** Enhances accessibility, promotes inclusivity, optimizes performance for efficiency, and continuously updates features based on user feedback.

6.1.2 Image capturing

The Image Captioning Module generates descriptive captions for captured images, leveraging advanced deep learning models for accurate and detailed descriptions. This module can be implemented using HTML, CSS ad PYTHON.

- **Real Time Capture:** Captures screen content in real-time, ensuring accuracy and immediacy in providing visual information. Additionally, it supports multiple capture modes for flexible usage based on user preferences.
- **Integration With Voice Commands:** Seamlessly integrates with voice commands for hands-free operation, enhancing accessibility and user convenience. Moreover, it offers customizable voice command settings for personalized interactions.

6.1.3 Textual annotation

The textual annotation module provides real-time, customizable annotations to enhance user comprehension and engagement with textual content. It utilizes CNN techniques and machine learning algorithms to perform these tasks.

- **Text Annotation:** Annotates text in real-time, providing immediate feedback and enhancing user comprehension of textual content. The module dynamically adjusts annotations based on context and user preferences, ensuring accuracy and relevance.

- **Customizable Interface:** Offers a personalized interface for users to tailor annotations, ensuring flexibility and control over the annotation process. Users can customize annotation styles, colors, and placement for a personalized experience, enhancing usability and engagement.

6.1.4 Verbal narration

The Verbal Narrations Module delivers real-time audio descriptions of screen content, providing essential auditory feedback for visually impaired users. It seamlessly integrates with voice commands for hands-free operation, enhancing accessibility and user convenience. Additionally, the module offers customizable narration settings, allowing users to personalize the audio experience for improved usability and engagement.

- The Verbal Narrations Module delivers essential audio descriptions, ensuring accessibility for visually impaired users during screen interactions.
- Integrating with voice commands enables hands-free operation, enhancing convenience and user experience.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

Our project signifies a major leap in assistive technology, particularly benefiting visually impaired individuals. By combining advanced deep learning models with voice command capabilities, real-time image capture, and dynamic text annotation, the app delivers a holistic solution for enhancing accessibility and independence in digital interactions.

The seamless integration of these modules ensures a user-friendly experience, facilitating effortless navigation of digital interfaces through voice commands, capturing and captioning screen content, and providing auditory feedback for improved comprehension.

The proposed approach demonstrates promise for practical applications like human-digital interaction showcasing robustness achieved through CNNs and LSTM.

Our Future work should focus on refining techniques, exploring scalability, and addressing emerging issues to enhance the practicality for visually impaired people.

FUTURE ENHANCEMENT

The future scope includes multilingual support and advanced natural language processing for more accurate captions. Integration of image recognition and collaboration with accessibility experts can further enhance the app's capabilities. Continuous innovation can establish this project as a benchmark in assistive technology, setting new standards for accessibility and inclusivity.

And also use several potential avenues for further development and improvement. Here are some future directions for enhancing this approach:

- Enhanced Security
- Integration with IOT Devices
- Real-time Analytics and Insights
- Mobile Application
- Integration with HR Systems
- Continuous Learning and Adaptation
- Scalability and Cloud Deployment

Furthermore, exploring features like object recognition and scene understanding can enable the app to provide comprehensive context-aware assistance, further enhancing usability and accessibility. Continuous innovation and refinement of the app's functionalities will establish it as a benchmark in assistive technology, setting new standards for accessibility and inclusivity in digital interactions.

APPENDIX 1 SAMPLE CODE

```
from flask import Flask, jsonify
import speech_recognition as sr
from pyautogui import screenshot
import os
import threading
import pyttsx3
import pickle
import numpy as np
from tqdm import tqdm
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from nltk.tokenize import word_tokenize
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
import tensorflow as tf

app = Flask(__name__)

# Variable to track whether the app should listen for voice commands
listening_for_command = False

# Initialize text-to-speech engine
engine = pyttsx3.init()

def speak(text):
    engine.say(text)
    engine.runAndWait()

def listen_for_voice_command():
    global listening_for_command

    recognizer = sr.Recognizer()
    recognizer.energy_threshold = 4000 # Adjust this threshold based on your environment

    print("Available Microphones:")
    for index, name in enumerate(sr.Microphone.list_microphone_names()):
        print(f"{index}: {name}")

    # Set the microphone index to the desired microphone
    microphone_index = 2 # Replace with the index corresponding to your desired microphone

    while True:
        if listening_for_command:
            try:
```

```

with sr.Microphone(device_index=microphone_index) as source:
    print("Say something:")
    audio = recognizer.listen(source, timeout=5) # Adjust the timeout as needed

text = recognizer.recognize_google(audio).lower()
print(f"Recognized: {text}")

# Check if the recognized text contains the trigger phrase
if 'buddy' in text:
    print("Voice command recognized: 'Hey buddy'")
    speak("Yes") # Provide audio feedback
    take_screenshot_and_generate_caption()
except sr.UnknownValueError:
    print("Speech recognition could not understand audio")
except sr.RequestError as e:
    print(f"Error connecting to Google API: {e}")
except Exception as e:
    print(f"Error: {e}")

# New route to activate the app
@app.route('/activate', methods=['POST'])
def activate_app():
    global listening_for_command
    listening_for_command = True
    return jsonify({'status': 'success', 'message': 'App activated.'})

def take_screenshot_and_generate_caption():
    global listening_for_command
    listening_for_command = False # Disable listening temporarily while taking a screenshot

    image_path = os.path.abspath('images/screenshot.png')
    screenshot(image_path)

    print(f"Screenshot captured successfully. Image saved to {image_path}")#

    # Load and preprocess image
    vgg_model = VGG16()
    vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    feature = vgg_model.predict(image, verbose=0)

    # Predict caption
    caption = predict_caption(loaded_model, feature, word_index, max_length, index_word)
    print("Predicted Caption:", caption)

```

```

# Save caption to a notepad file
with open('caption.txt', 'w') as f:
    f.write(caption)

# Convert caption to audio
speak(caption)

listening_for_command = True # Resume listening for voice commands#

Load features and captions
BASE_DIR = "C:\\Users\\Thejeal Sri\\Desktop\\flickr8k"
WORKING_DIR = "C:\\Users\\Thejeal Sri\\Desktop\\working"

with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)

with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:next(f)
    captions_doc = f.read()

# Process lines
mapping = {}
for line in tqdm(captions_doc.split('\n')):
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    image_id = image_id.split('.')[0]
    caption = " ".join(caption) if
    image_id not in mapping:
        mapping[image_id] = []
    mapping[image_id].append(caption)

# Clean captions
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            caption = captions[i].lower()
            caption = caption.replace('[^A-Za-z]', '')
            caption = caption.replace('\s+', ' ')
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word) > 1]) + '
endseq'
            captions[i] = caption

clean(mapping)

# Create a vocabulary

```

```

all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)

# Tokenize captions
vocabulary = set()
for caption in all_captions:
    vocabulary.update(word_tokenize(caption))

# Create word-to-index and index-to-word mappings
word_index = {word: idx + 1 for idx, word in enumerate(sorted(vocabulary))}
index_word = {idx: word for word, idx in word_index.items()}
vocab_size = len(word_index) + 1

# Convert captions to sequences of integers
def captions_to_sequences(mapping, word_index):
    sequences = {}
    for key, captions in mapping.items():
        sequences[key] = []
        for caption in captions:
            seq = [word_index[word] for word in word_tokenize(caption) if word in word_index]
            sequences[key].append(seq)
    return sequences

sequences = captions_to_sequences(mapping, word_index)
max_length = max(len(seq) for seqs in sequences.values() for seq in seqs)

# Load model architecture from JSON
model_architecture_path = r"C:\Users\Thejeal Sri\Desktop\mod\model_architecture.json"
with open(model_architecture_path, 'r') as f:
    model_json = f.read()
loaded_model = tf.keras.models.model_from_json(model_json)

# Load model weights
model_weights_path = r"C:\Users\Thejeal Sri\Desktop\mod\model_weights.h5"
loaded_model.load_weights(model_weights_path)

def predict_caption(loaded_model, image, word_index, max_length, index_word):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [word_index.get(word, 0) for word in word_tokenize(in_text)]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = loaded_model.predict([image, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = index_word.get(yhat)
        if word is None or word == 'endseq':

```



```

        break
    in_text += ' ' + word
return in_text.strip('startseq ').strip('endseq')

if __name__ == '__main__':
    os.makedirs('images', exist_ok=True)
    # Start a separate thread for continuous voice command listening
    voice_command_thread = threading.Thread(target=listen_for_voice_command)
    voice_command_thread.start()
    app.run(debug=True)

#Image captioning model

import os
import pickle
import numpy as np
from tqdm.notebook import tqdm
import tensorflow
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array

from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
from tensorflow.keras.preprocessing.text import Tokenizer
BASE_DIR = "C:\\Users\\Thejeal Sri\\Desktop\\flickr8k"
WORKING_DIR = "C:\\Users\\Thejeal Sri\\Desktop\\working"
features = {}
# extract features from image
directory = os.path.join(BASE_DIR, 'Images')

for img_name in tqdm(os.listdir(directory)):
    # load the image from file
    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, verbose=0)
    # get image ID
    image_id = img_name.split('.')[0]
    # store feature
    features[image_id] = feature

```

```

pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'), 'wb'))
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f: next(f)
    captions_doc = f.read()
mapping = {}
# process lines
for line in tqdm(captions_doc.split("\n")):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
len(mapping)
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + '
endseq'
            captions[i] = caption
all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1

```

```

image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size,
batch_size):
    # loop over images
    X1, X2, y = list(), list(), list()
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence
                seq = tokenizer.texts_to_sequences([caption])[0]
                # split the sequence into X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pairs
                    in_seq, out_seq = seq[:i], seq[i] #
                    pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                    # store the sequences
                    X1.append(features[key][0])
                    X2.append(in_seq)
                    y.append(out_seq)
                if n == batch_size:
                    X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                    yield {"image": X1, "text": X2}, y
                    X1, X2, y = list(), list(), list()
                    n = 0
inputs1 = Input(shape=(4096,), name="image")
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,), name="text")
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)se2
= Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

```

```

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')
epochs = 5
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size,
batch_size)
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
epochs = 5
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size,
batch_size)
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
import pickle
import pickle

# Save model architecture as JSON
model_architecture_path = r"C:\Users\Thejeal Sri\Desktop\mod\model_architecture.json"
with open(model_architecture_path, 'w') as f:
    f.write(model.to_json())

#Save model weights
model_weights_path = r"C:\Users\Thejeal Sri\Desktop\mod\model_weights.pkl"
with open(model_weights_path, 'wb') as f:
    pickle.dump(model.get_weights(), f)

```

APPENDIX 2 SCREENSHOTS

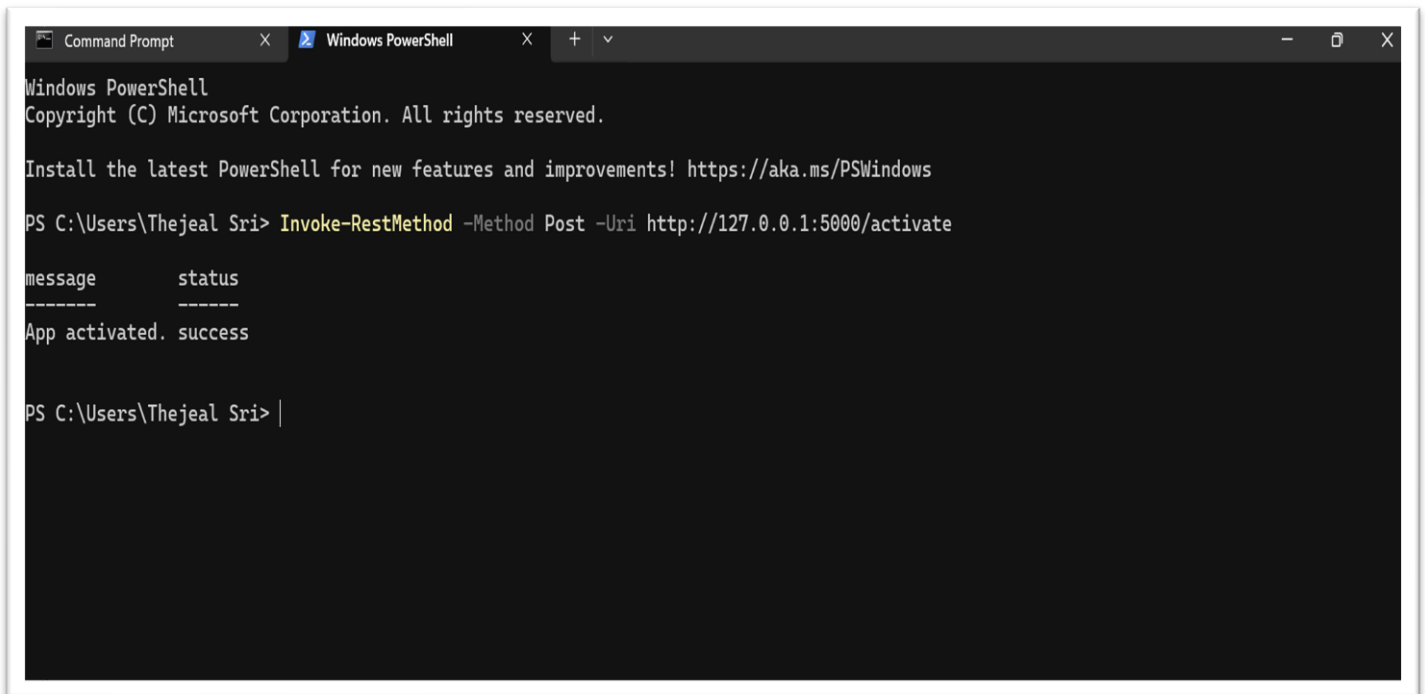


Figure No.A.2.1. App Activation in PowerShell

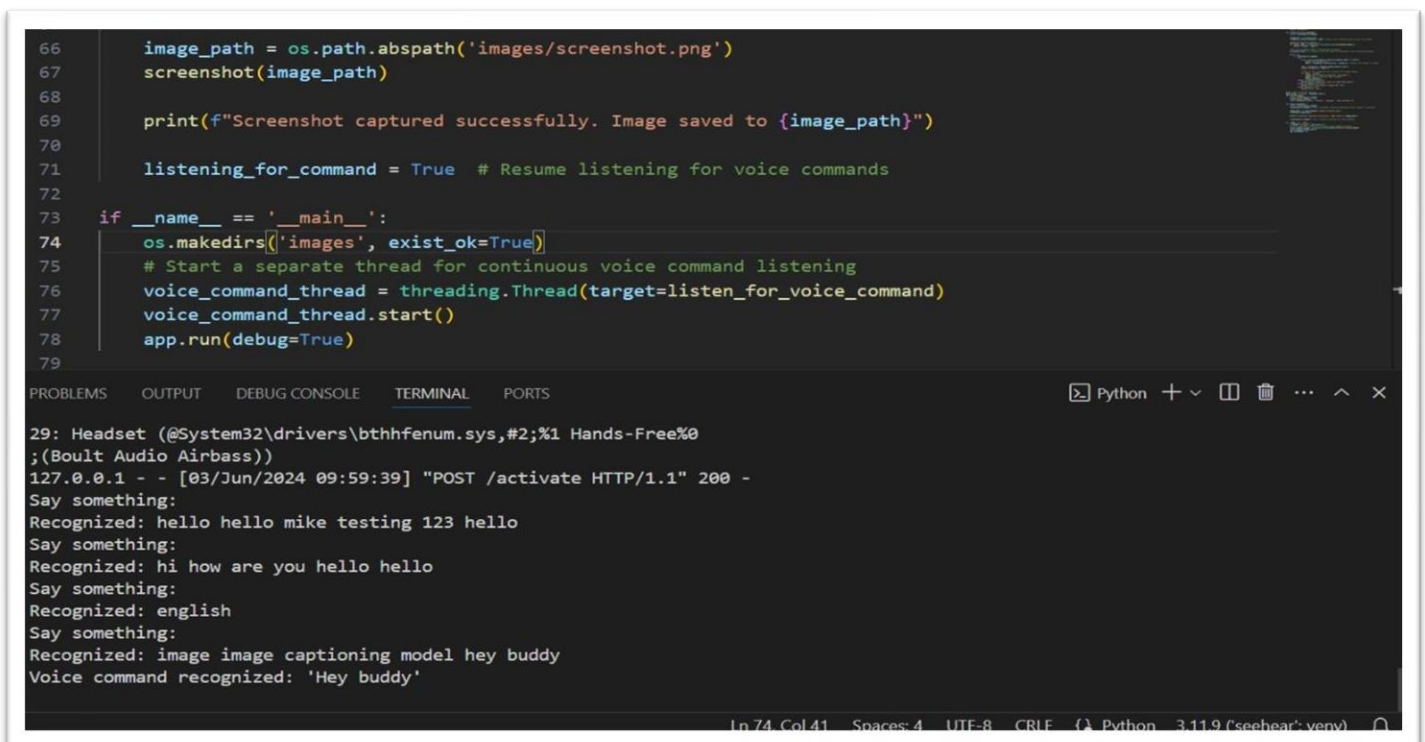
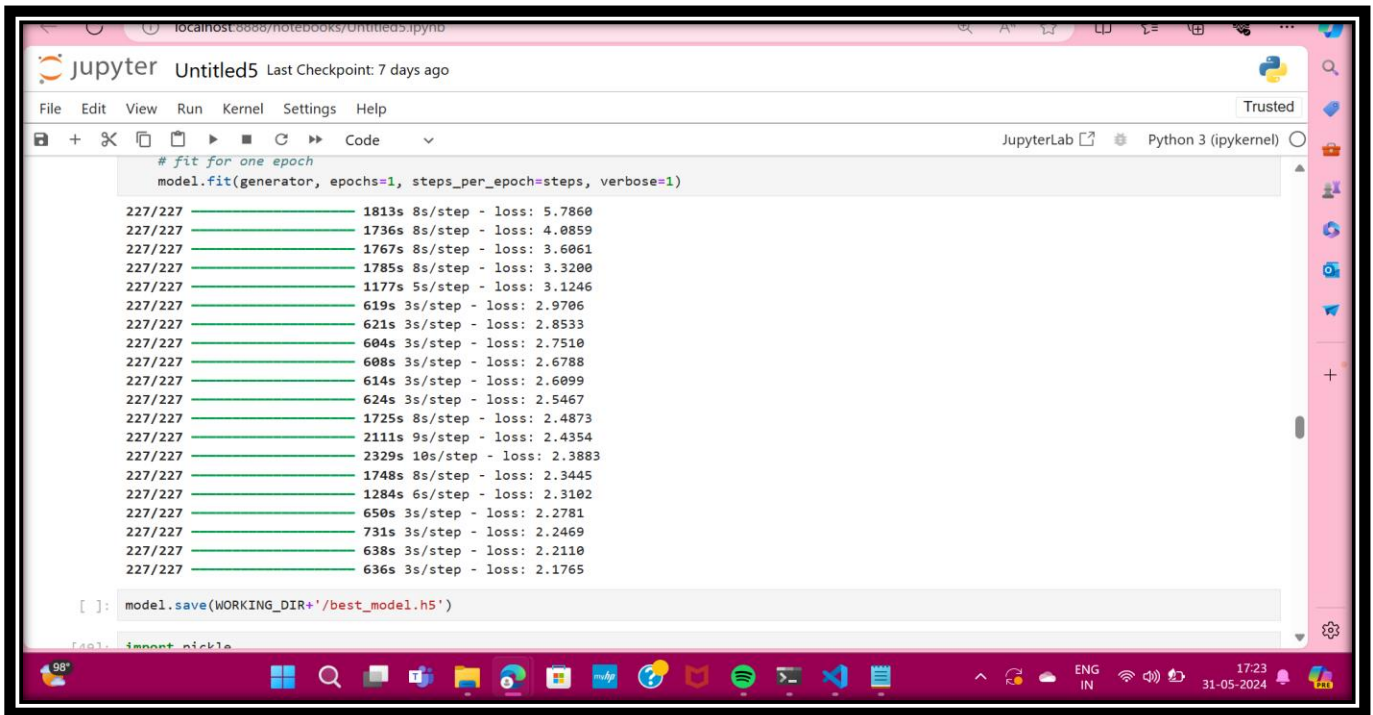


Figure No.A.2.2. Voice Recognition



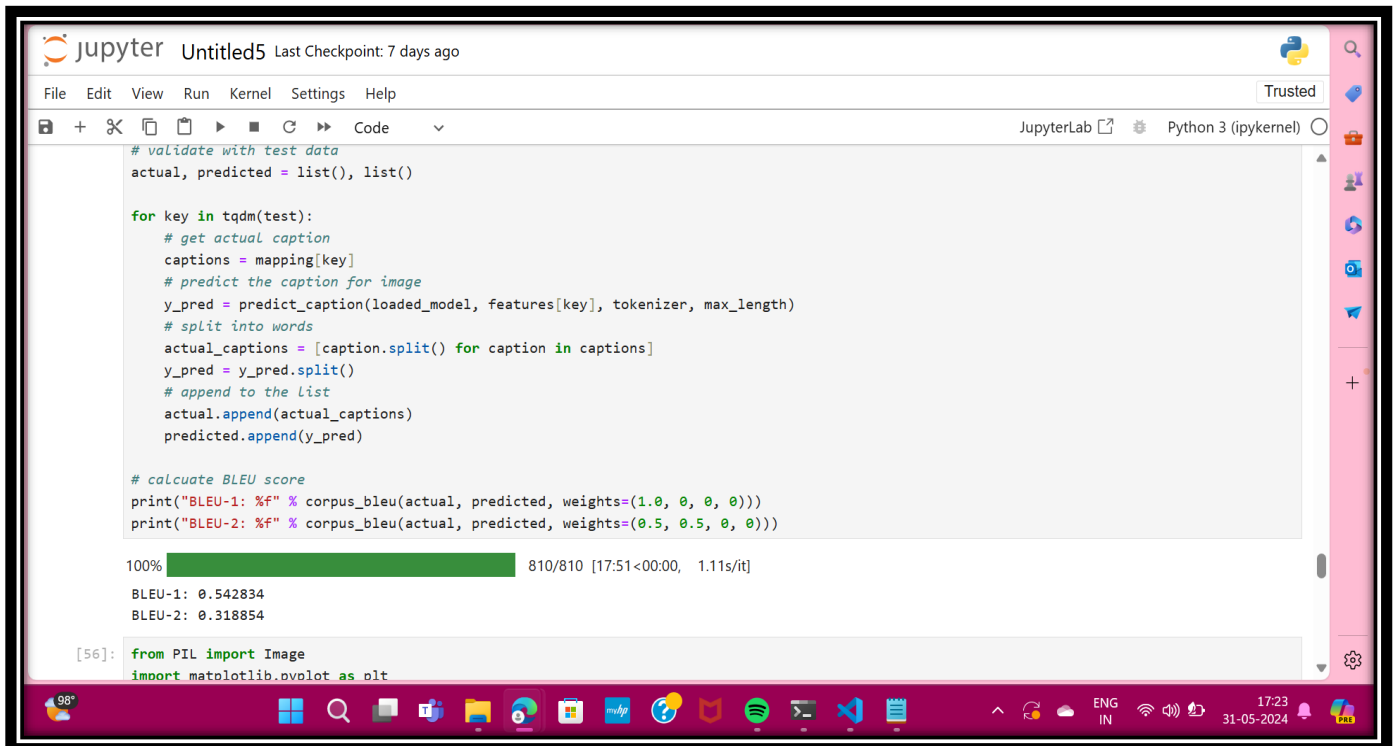
The image shows a JupyterLab interface with a code editor and a terminal. The code editor contains a Python script for training a model. The terminal displays the output of the training process, showing progress bars and loss values for each step.

```
# fit for one epoch
model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)

227/227 ----- 1813s 8s/step - loss: 5.7860
227/227 ----- 1736s 8s/step - loss: 4.0859
227/227 ----- 1767s 8s/step - loss: 3.6061
227/227 ----- 1785s 8s/step - loss: 3.3200
227/227 ----- 1177s 5s/step - loss: 3.1246
227/227 ----- 619s 3s/step - loss: 2.9706
227/227 ----- 621s 3s/step - loss: 2.8533
227/227 ----- 604s 3s/step - loss: 2.7510
227/227 ----- 608s 3s/step - loss: 2.6788
227/227 ----- 614s 3s/step - loss: 2.6099
227/227 ----- 624s 3s/step - loss: 2.5467
227/227 ----- 1725s 8s/step - loss: 2.4873
227/227 ----- 2111s 9s/step - loss: 2.4354
227/227 ----- 2329s 10s/step - loss: 2.3883
227/227 ----- 1748s 8s/step - loss: 2.3445
227/227 ----- 1284s 6s/step - loss: 2.3102
227/227 ----- 650s 3s/step - loss: 2.2781
227/227 ----- 731s 3s/step - loss: 2.2469
227/227 ----- 638s 3s/step - loss: 2.2110
227/227 ----- 636s 3s/step - loss: 2.1765

[ ]: model.save(WORKING_DIR+'best_model.h5')
```

Figure No.A.2.3. Epoch Training



The image shows a JupyterLab interface with a code editor and a terminal. The code editor contains a Python script for calculating BLEU scores. The terminal displays the output of the script, showing the BLEU-1 and BLEU-2 scores.

```
# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(loader_model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

# calculate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))

100% [████████████████████████████████████████] 810/810 [17:51<00:00, 1.11s/it]
BLEU-1: 0.542834
BLEU-2: 0.318854

[56]: from PIL import Image
import matplotlib.pyplot as plt
```

Figure No.A.2.4. Bleu Scores

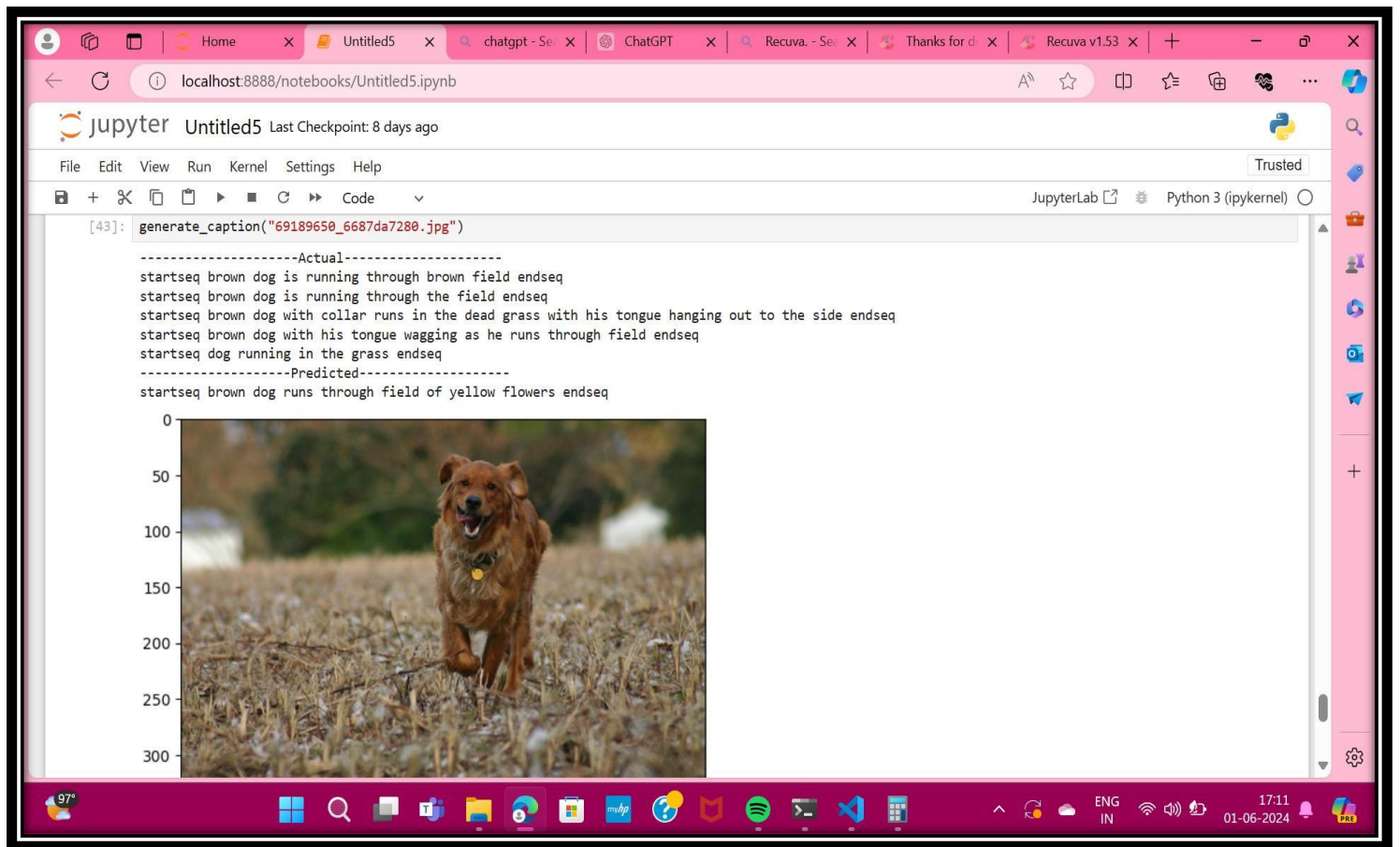


Figure No.A.2.5. Image Caption Generation

REFERENCES

1. Z. Chen, L. Wang, and X. Liu (2020) Automatic speech recognition and generation using recurrent neural networks. *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 3, pp. 456-468.
2. T. Chen, K. Wang, and L. Zhu (2018) Natural language generation with deep learning models. *Journal of Artificial Intelligence Research*, vol. 61, pp. 465-478.
3. A. Gupta, S. Kumar, and R. Singh (2017) Automated text summarization using deep learning techniques. *Information Processing & Management*, vol. 53, no. 5, pp. 789-802.
4. H. Jiang, Y. Xu, and Z. Zhang (2018) Deep learning techniques for natural language understanding and generation. *ACM Transactions on Intelligent Systems and Technology*, vol. 9, no. 1, pp. 345-357.
5. J. Kim, S. Lee, and H. Park (2020) Voice conversion using generative adversarial networks. *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 28, no. 4, pp. 789-802.
6. Y. Li, J. Zhang, and S. Wang (2019) Voice synthesis using deep learning models. *Neural Computing & Applications*, vol. 31, no. 4, pp. 1123-1135.
7. N. Wang, Y. Zhang, and L. Liu (2019) Text generation using recurrent neural networks. *Neural Networks*, vol. 112, pp. 89-102.