**INDEX**

|  |  |  |
| --- | --- | --- |
| **SL. NO** | **TITLE** | **PAGE NO.** |
| **CHAPTER 1** | **Introduction** 1.1 Project Overview  1.2 Objectives  1.3 Scope | **2** |
| **CHAPTER 2** | **Literature Review**  2.1 Chatbot Technologies  2.2 Previous Work | **4** |
| **CHAPTER 3** | **System Design**  3.1 System Architecture  3.2 Components  3.2.1 Frontend  3.2.2 Backend  3.2.3 Database | **8** |
| **CHAPTER 4** | **Implementation**  4.1 Environment Setup  4.2 Entire Code  4.3 Code Explanation   4.3.1 Importing Libraries   4.3.2 Initialization   4.3.3 GUI Setup   4.3.4 Function Definitions  4.3.5 Running the Application | **13** |
| **CHAPTER 5** | **Results**  5.1 Testing and Debugging  5.2 Screenshots | **37** |
|  | **Conclusion**  Future Work | **41** |
|  | **References** | **44** |

### **CHAPTER 1:**

### **INTRODUCTION**

#### 1.1 Project Overview

In recent years, the advancement of artificial intelligence (AI) has revolutionized various industries, including customer service, healthcare, and education. Among these advancements, chatbots have emerged as a pivotal technology, enhancing user interaction and providing immediate assistance. The project titled "Bella an AI Chatbot" aims to develop an interactive AI-powered chatbot that can engage with users through both text and voice. Bella is designed to operate in two distinct modes: Learning Mode and Talking Mode. In Learning Mode, the chatbot solicits feedback from users to refine its responses, thereby continuously improving its interaction quality. In Talking Mode, Bella engages in conversations, providing answers and engaging in small talk based on its training data and previous interactions.

This project integrates several cutting-edge technologies including natural language processing (NLP) for understanding user input, text-to-speech (TTS) for generating spoken responses, and speech recognition for processing spoken user input. The graphical user interface (GUI) is designed to be user-friendly, allowing seamless interaction with Bella. The chatbot’s capabilities are built using Python, leveraging libraries such as Tkinter for GUI, gTTS for text-to-speech conversion, and speech recognition for processing voice inputs.

#### 1.2 Objectives

The primary objectives of this project are as follows:

* **Develop an Interactive Chatbot**: Create a chatbot named Bella that can interact with users using both text and voice.
* **Implement Voice Interaction**: Integrate speech recognition and text-to-speech technologies to facilitate voice-based interactions.
* **Enable Continuous Learning**: Develop a Learning Mode where Bella can ask users for feedback to improve future responses.
* **User-Friendly GUI**: Design a graphical user interface that is intuitive and easy to use, providing a seamless experience for users.
* **Support for Dual Modes**: Allow Bella to operate in both Learning Mode and Talking Mode, providing flexibility in interaction.
* **Achieve Real-Time Performance**: Ensure that Bella can respond to user inputs in real-time, maintaining an engaging conversation flow.

#### 1.3 Scope

The scope of the project encompasses the following key areas:

* **Natural Language Processing (NLP)**: Implementing NLP techniques to enable Bella to understand and process user inputs effectively.
* **Text-to-Speech (TTS)**: Utilizing TTS technology to convert text responses generated by Bella into spoken words.
* **Speech Recognition**: Integrating speech recognition to allow Bella to understand and process spoken user inputs.
* **Graphical User Interface (GUI)**: Designing and developing a user-friendly interface using Tkinter, enabling users to interact with Bella through text and voice.
* **Machine Learning Integration**: Developing a feedback mechanism in Learning Mode where Bella can learn from user interactions and improve over time.
* **Backend and Database Management**: Setting up a backend to handle user interactions, store feedback, and manage improvements to the chatbot's responses.
* **Testing and Validation**: Conducting extensive testing to ensure Bella performs as expected, identifying and resolving any bugs or issues.
* **Deployment and Usability**: Deploying the chatbot in a manner that is accessible to users and ensuring it can handle multiple interactions efficiently.

Through these objectives and scope, the project aims to deliver a robust and interactive AI chatbot that enhances user engagement and continually improves its performance through user feedback.

**CHAPTER 2:**

**LITERATURE REVIEW**

#### 2.1 Chatbot Technologies

Chatbots, often referred to as conversational agents, leverage artificial intelligence to simulate human conversation. The technologies underpinning chatbots have evolved significantly, enabling more natural and effective interactions. This section provides an overview of the key technologies used in chatbot development.

**2.1.1 Natural Language Processing (NLP)**

Natural Language Processing is a crucial component of modern chatbots, allowing them to understand, interpret, and respond to human language. NLP encompasses several subfields, including:

* **Tokenization**: Breaking down text into smaller units, such as words or phrases.
* **Part-of-Speech Tagging**: Identifying the grammatical role of each word in a sentence.
* **Named Entity Recognition (NER)**: Detecting and classifying entities such as names, dates, and locations.
* **Sentiment Analysis**: Determining the emotional tone of a text.

NLP technologies enable chatbots to parse user inputs, extract meaningful information, and generate coherent responses.

**2.1.2 Machine Learning (ML) and Deep Learning (DL)**

Machine Learning and Deep Learning are pivotal in developing chatbots that can learn from data and improve over time. Commonly used techniques include:

* **Supervised Learning**: Training models on labeled datasets to predict outcomes based on input features.
* **Reinforcement Learning**: Using feedback from interactions to refine the chatbot's responses.
* **Neural Networks**: Employing architectures such as Recurrent Neural Networks (RNNs) and Transformer models for language generation and understanding.

These techniques allow chatbots to handle a wide range of queries, adapt to new information, and provide more accurate and contextually relevant responses.

**2.1.3 Text-to-Speech (TTS) and Speech Recognition**

Text-to-Speech and Speech Recognition technologies enable voice-based interactions with chatbots.

* **Text-to-Speech (TTS)**: Converts text responses generated by the chatbot into natural-sounding speech. TTS systems utilize phonetic and prosodic models to ensure clarity and intonation in speech.
* **Speech Recognition**: Converts spoken language into text, allowing the chatbot to process and respond to voice inputs. Advanced speech recognition systems use deep learning models to achieve high accuracy in diverse and noisy environments.

**2.1.4 Integration and Deployment**

For a chatbot to be practical and widely accessible, it must be seamlessly integrated into various platforms and deployed effectively.

* **APIs and SDKs**: Facilitate integration with messaging platforms, websites, and mobile applications.
* **Cloud Services**: Offer scalable and flexible infrastructure to handle large volumes of interactions.
* **Security and Privacy**: Ensuring user data is protected through encryption and compliance with data protection regulations.

These technologies collectively contribute to creating sophisticated chatbots capable of engaging in meaningful and productive conversations with users.

#### 2.2 Previous Work

The development of chatbots has a rich history, with several notable milestones and influential projects shaping the field. This section reviews some key developments and previous work in chatbot technology.

**2.2.1 Early Chatbots**

The history of chatbots dates back to the 1960s, with early examples such as:

* **ELIZA (1966)**: Developed by Joseph Weizenbaum at MIT, ELIZA was one of the first chatbots. It used pattern matching and substitution methodology to simulate conversation, most famously mimicking a Rogerian psychotherapist.
* **PARRY (1972)**: Created by Kenneth Colby, PARRY was designed to simulate a person with paranoid schizophrenia. It was more advanced than ELIZA, incorporating a model of mental illness to generate its responses.

**2.2.2 Rule-Based Chatbots**

Throughout the 1980s and 1990s, chatbots primarily relied on rule-based systems, where responses were generated based on predefined rules and scripts. Notable examples include:

* **ALICE (Artificial Linguistic Internet Computer Entity)**: Developed by Richard Wallace in 1995, ALICE used heuristic pattern matching rules to engage in conversation. It won the Loebner Prize Turing Test multiple times.
* **Jabberwacky**: Created by Rollo Carpenter, Jabberwacky aimed to simulate natural human chat in an entertaining manner. It used a large dataset of human interactions to generate responses.

**2.2.3 Modern Chatbots**

The advent of machine learning and deep learning technologies has significantly advanced chatbot capabilities. Modern chatbots leverage large datasets and sophisticated algorithms to deliver more natural and context-aware interactions. Key developments include:

* **IBM Watson**: Launched in 2011, Watson uses natural language processing and machine learning to answer questions and provide recommendations. It gained fame for winning the game show Jeopardy! against human champions.
* **Google Assistant**: Released in 2016, Google Assistant uses advanced speech recognition and natural language understanding to perform tasks, answer queries, and control smart devices.
* **OpenAI's GPT-3**: Introduced in 2020, GPT-3 (Generative Pre-trained Transformer 3) is one of the largest language models ever created, with 175 billion parameters. It can generate human-like text and perform a variety of language tasks with minimal fine-tuning.

**2.2.4 Chatbots in Industry**

Chatbots are now widely used across various industries, including:

* **Customer Service**: Companies deploy chatbots to handle customer inquiries, provide support, and resolve issues.
* **Healthcare**: Chatbots assist in scheduling appointments, providing medical information, and offering mental health support.
* **Education**: Educational institutions use chatbots to offer tutoring, answer student queries, and facilitate administrative tasks.

These examples demonstrate the diverse applications and significant impact of chatbots in enhancing user interaction and automating processes.

### **CHAPTER 3:**

### **SYSTEM DESIGN**

#### 3.1 System Architecture

The architecture of the Bella AI Chatbot is designed to seamlessly integrate various components to provide a robust and interactive user experience. The system architecture comprises three primary layers: the Frontend, Backend, and Database. Each layer interacts with the others to ensure smooth operation and effective response generation.

**3.1.1 Overview**

* **Frontend**: The user interface (UI) layer where users interact with Bella. It includes text input, voice input, and output elements. The UI is built using the Tkinter library in Python, providing a visually appealing and user-friendly interface.
* **Backend**: The processing layer that handles user inputs, processes data, and generates responses. This layer utilizes NLP, machine learning, and text-to-speech (TTS) technologies to understand and respond to user queries. It also manages the switching between Learning Mode and Talking Mode.
* **Database**: The storage layer that maintains user feedback and learning data. This layer ensures that Bella can improve over time based on interactions and feedback. The database is implemented using SQLite, providing a lightweight and efficient storage solution.

**3.1.2 Interaction Flow**

1. **User Interaction**: Users interact with Bella through the GUI by typing or speaking their queries.
2. **Input Processing**: The Backend processes the input using speech recognition for voice inputs and NLP for text inputs.
3. **Response Generation**: The processed input is analyzed to generate an appropriate response. In Learning Mode, user feedback is solicited to improve future responses.
4. **Output Delivery**: The response is delivered back to the user through text and voice, with the corresponding updates made to the GUI.

#### 3.2 Components

The system comprises several key components that work together to deliver a seamless chatbot experience. These components are divided into the Frontend, Backend, and Database.

#### 3.2.1 Frontend

**3.2.1.1 Graphical User Interface (GUI)**

The GUI is designed using the Tkinter library in Python, providing an interactive platform for users to engage with Bella. The main features of the GUI include:

* **Text Input Box**: Allows users to type their queries.
* **Speak Button**: Enables voice input for users who prefer speaking over typing.
* **Send Button**: Submits the text input for processing.
* **Chat Display**: Shows the conversation history, including user inputs and Bella’s responses.
* **Mode Toggle**: A checkbox that allows users to switch between Learning Mode and Talking Mode.

**3.2.1.2 Design and Layout**

The GUI is divided into two main sections:

* **Left Section**: Displays Bella’s image or video, providing a visual representation of the chatbot.
* **Right Section**: Contains the interaction elements, including the text input box, buttons, and chat display.

**3.2.1.3 User Experience (UX)**

The UX is designed to be intuitive and engaging:

* **Responsive Design**: The layout adjusts based on the window size, ensuring usability across different screen sizes.
* **Visual Feedback**: Provides immediate visual feedback for user actions, such as button clicks and input submissions.
* **Accessibility**: Ensures that the chatbot is accessible to users with different needs, including those who prefer voice interactions.

#### 3.2.2 Backend

**3.2.2.1 Natural Language Processing (NLP)**

The NLP component is responsible for understanding user inputs and generating appropriate responses. It includes:

* **Text Tokenization**: Breaking down the input text into smaller components for easier processing.
* **Intent Recognition**: Identifying the user’s intent to provide relevant responses.
* **Entity Extraction**: Extracting key entities from the user input, such as names, dates, and locations.

**3.2.2.2 Machine Learning Models**

The chatbot leverages machine learning models to generate responses and improve over time:

* **Pre-trained Models**: Uses models like GPT-2 for generating human-like responses.
* **Training and Fine-tuning**: Continuously improves the models based on user interactions and feedback.

**3.2.2.3 Speech Recognition and Text-to-Speech (TTS)**

* **Speech Recognition**: Converts spoken input into text using libraries like speech\_recognition.
* **Text-to-Speech (TTS)**: Converts text responses into speech using the gTTS library, providing a voice for Bella.

**3.2.2.4 Mode Management**

Handles the switching between Learning Mode and Talking Mode:

* **Learning Mode**: Solicits user feedback to improve responses.
* **Talking Mode**: Engages in conversation without requesting feedback.

**3.2.2.5 Integration**

Integrates various backend components to ensure smooth operation:

* **API Calls**: Manages API calls for NLP and machine learning services.
* **Error Handling**: Ensures robust error handling to manage unexpected inputs or system failures.

#### 3.2.3 Database

**3.2.3.1 Database Design**

The database is designed to store user feedback and learning data:

* **User Interactions**: Logs of user queries and Bella’s responses.
* **Feedback Data**: User ratings and suggestions for improving responses.

**3.2.3.2 SQLite Implementation**

Uses SQLite for database management:

* **Lightweight and Efficient**: Suitable for the project’s requirements, providing fast and reliable data storage.
* **Integration with Python**: Easily integrates with the Python application using the sqlite3 library.

**3.2.3.3 Data Management**

Handles the storage, retrieval, and updating of data:

* **Insert Operations**: Stores new interactions and feedback.
* **Update Operations**: Updates existing responses based on user feedback.
* **Query Operations**: Retrieves data for analysis and improving responses.

**3.2.3.4 Data Security**

Ensures the security and privacy of user data:

* **Encryption**: Encrypts sensitive data to protect it from unauthorized access.
* **Access Control**: Implements access control measures to restrict data access to authorized components and users.

### **CHAPTER 4:**

### **IMPLEMENTATION**

#### 4.1 Environment Setup

Setting up the development environment is the first step to implementing the Bella AI Chatbot. This section provides a detailed guide on preparing the environment, installing necessary libraries, and ensuring all dependencies are met.

**4.1.1 System Requirements**

* **Operating System**: Windows, macOS, or Linux
* **Python Version**: Python 3.7 or higher
* **Hardware**: A computer with at least 4GB of RAM and a modern CPU for running machine learning models efficiently.

**4.1.2 Installing Python**

Ensure you have Python installed on your system. You can download and install Python from the [official website](https://www.python.org/).

**4.1.3 Setting Up a Virtual Environment**

It is recommended to use a virtual environment to manage dependencies and avoid conflicts. To set up a virtual environment:

# Install virtualenv if you haven't already

pip install virtualenv

# Create a virtual environment

virtualenv chatbot\_env

# Activate the virtual environment

# On Windows

chatbot\_env\Scripts\activate

# On macOS/Linux

source chatbot\_env/bin/activate

**4.1.4 Installing Required Libraries**

Install the necessary libraries using pip:

pip install numpy pandas sqlite3 gtts transformers speech recognition pillow opencv-python pygame

**4.1.5 Downloading Additional Resources**

Ensure you have the necessary resources such as images and videos for Bella. Download and place the following files in your project directory:

* idle\_image.png: An image of Bella when idle.
* speaking\_video.mp4: A video of Bella when speaking.
* dataset.csv: A CSV file containing question-answer pairs.

#### 4.2 Entire Code

Here is the complete code for the Bella AI Chatbot. Ensure you have all the required files in the same directory as this script.

import os

import cv2

import pygame

import tkinter as tk

from tkinter import scrolledtext, simpledialog

from gtts import gTTS

from transformers import pipeline, AutoModelForCausalLM, AutoTokenizer

import speech\_recognition as sr

import sqlite3

import pandas as pd

import csv

from threading import Thread

from PIL import Image, ImageTk

class AnimeChatBot:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Bella ChatBot")

self.root.geometry("1080x720")

self.root.configure(bg="#2E3440")

self.root.resizable(True, True)

# Load the idle image and speaking video

self.idle\_image\_path = 'idle\_image.png'

self.speaking\_video\_path = 'speaking\_video.mp4'

# Initialize text-to-speech and speech recognition

self.recognizer = sr.Recognizer()

# Load the model and tokenizer

self.tokenizer = AutoTokenizer.from\_pretrained("distilgpt2")

self.model = AutoModelForCausalLM.from\_pretrained("distilgpt2")

self.text\_generator = pipeline("text-generation", model=self.model, tokenizer=self.tokenizer)

# Load the dataset

self.dataset\_path = 'dataset.csv'

self.load\_dataset()

# Initialize pygame for playing audio

pygame.init()

# Initialize database connection

self.db\_conn = sqlite3.connect('reviews.db')

self.db\_cursor = self.db\_conn.cursor()

# Farewell phrases

self.farewell\_phrases = ["goodbye", "bye", "see you", "farewell", "later", "ciao", "adios"]

# Mode toggle

self.learning\_mode = tk.BooleanVar()

self.learning\_mode.set(False)

# Create GUI elements

self.create\_widgets()

def load\_dataset(self):

self.dataset = pd.read\_csv(self.dataset\_path)

self.dataset\_dict = {q.lower(): a for q, a in zip(self.dataset['question'], self.dataset['answer'])}

def create\_widgets(self):

# Frame for the image and video

self.image\_frame = tk.Frame(self.root, bg="#4C566A", bd=2, relief=tk.SUNKEN)

self.image\_frame.grid(row=0, column=0, rowspan=4, padx=10, pady=10, sticky="nsew")

self.idle\_image = Image.open(self.idle\_image\_path).resize((320, 480), Image.Resampling.LANCZOS)

self.idle\_image\_tk = ImageTk.PhotoImage(self.idle\_image)

self.image\_label = tk.Label(self.image\_frame, image=self.idle\_image\_tk, bg="#4C566A")

self.image\_label.pack(expand=True)

# Frame for the chat and controls

self.control\_frame = tk.Frame(self.root, bg="#2E3440")

self.control\_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

self.toggle\_button = tk.Checkbutton(self.control\_frame, text="Learning Mode", variable=self.learning\_mode, command=self.toggle\_mode, font=("Helvetica", 12), bg="#2E3440", fg="#D8DEE9", selectcolor="#4C566A")

self.toggle\_button.grid(row=0, column=0, sticky="w")

self.chat\_display = scrolledtext.ScrolledText(self.control\_frame, wrap=tk.WORD, width=60, height=20, font=("Helvetica", 12), bg="#3B4252", fg="#ECEFF4")

self.chat\_display.grid(row=1, column=0, columnspan=2, pady=10)

self.input\_box = tk.Entry(self.control\_frame, width=50, font=("Helvetica", 12))

self.input\_box.grid(row=2, column=0, pady=10, sticky="ew")

self.input\_box.bind("<Return>", lambda event: self.process\_input())

self.send\_button = tk.Button(self.control\_frame, text="Send", command=self.process\_input, font=("Helvetica", 12), bg="#5E81AC", fg="#ECEFF4")

self.send\_button.grid(row=2, column=1, padx=5)

self.speak\_button = tk.Button(self.control\_frame, text="Speak", command=self.speech\_input, font=("Helvetica", 12), bg="#5E81AC", fg="#ECEFF4")

self.speak\_button.grid(row=3, column=0, columnspan=2, pady=10)

# Configure row and column weights for resizing

self.root.grid\_rowconfigure(0, weight=1)

self.root.grid\_rowconfigure(1, weight=1)

self.root.grid\_rowconfigure(2, weight=1)

self.root.grid\_rowconfigure(3, weight=1)

self.root.grid\_columnconfigure(0, weight=1)

self.root.grid\_columnconfigure(1, weight=2)

def toggle\_mode(self):

mode = "Learning Mode" if self.learning\_mode.get() else "Talking Mode"

self.chat\_display.insert(tk.END, f"Switched to {mode}\n")

def display\_image(self):

self.image\_label.configure(image=self.idle\_image\_tk)

def play\_video(self):

cap = cv2.VideoCapture(self.speaking\_video\_path)

while pygame.mixer.music.get\_busy():

ret, frame = cap.read()

if not ret:

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)

continue

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frame = cv2.resize(frame, (320, 480))

frame = Image.fromarray(frame)

frame = ImageTk.PhotoImage(frame)

self.image\_label.configure(image=frame)

self.image\_label.image = frame

self.root.update\_idletasks()

self.root.update()

cap.release()

def speak(self, text):

tts = gTTS(text=text, lang='en')

tts.save("response.mp3")

pygame.mixer.music.load("response.mp3")

pygame.mixer.music.play()

Thread(target=self.play\_video).start()

while pygame.mixer.music.get\_busy():

self.root.update()

pygame.mixer.music.unload()

if os.path.exists("response.mp3"):

os.remove("response.mp3")

self.display\_image()

def recognize\_speech(self):

with sr.Microphone() as source:

self.chat\_display.insert(tk.END, "Listening...\n")

audio = self.recognizer.listen(source)

try:

return self.recognizer.recognize\_google(audio)

except sr.UnknownValueError:

return "Sorry, I did not understand that."

def get\_response(self, user\_input):

user\_input\_lower = user\_input.lower()

# Check if the user input contains any keywords from the dataset

for question in self.dataset\_dict:

if all(keyword in user\_input\_lower for keyword in question.split()):

return self.dataset\_dict[question]

# If no keywords match, use the text generation pipeline

response = self.text\_generator(user\_input, max\_length=30, num\_return\_sequences=1, truncation=True)

return response[0]['generated\_text'].strip()

def ask\_for\_review(self, question, answer):

self.chat\_display.insert(tk.END, f"Please rate the response (1-10):\n")

self.chat\_display.see(tk.END)

review = simpledialog.askinteger("Input", "Please rate the response (1-10):", parent=self.root, minvalue=1, maxvalue=10)

if review is not None:

if review >= 8:

self.save\_response(question, answer)

elif review < 5:

better\_answer = simpledialog.askstring("Input", "Please provide a better answer:", parent=self.root)

if better\_answer:

self.save\_response(question, better\_answer)

def save\_response(self, question, answer):

question\_lower = question.lower()

# Check if the question already exists in the dataset

if question\_lower in self.dataset\_dict:

self.chat\_display.insert(tk.END, f"Question '{question\_lower}' already exists in the dataset. Skipping save.\n")

self.chat\_display.see(tk.END)

return

# Update the dataset dictionary

self.dataset\_dict[question\_lower] = answer

# Update the CSV file

with open(self.dataset\_path, 'a', newline='') as file:

writer = csv.writer(file)

writer.writerow([question\_lower, answer])

def process\_input(self):

user\_input = self.input\_box.get()

self.input\_box.delete(0, tk.END)

self.chat\_display.insert(tk.END, f"User: {user\_input}\n")

self.chat\_display.see(tk.END)

if any(phrase in user\_input.lower() for phrase in self.farewell\_phrases):

self.speak("Goodbye! Have a great day!")

self.root.quit()

else:

response = self.get\_response(user\_input)

self.chat\_display.insert(tk.END, f"Bella: {response}\n")

self.chat\_display.see(tk.END)

self.speak(response)

if self.learning\_mode.get():

self.ask\_for\_review(user\_input, response)

def speech\_input(self):

user\_input = self.recognize\_speech()

self.chat\_display.insert(tk.END, f"User: {user\_input}\n")

self.chat\_display.see(tk.END)

if any(phrase in user\_input.lower() for phrase in self.farewell\_phrases):

self.speak("Goodbye! Have a great day!")

self.root.quit()

else:

response = self.get\_response(user\_input)

self.chat\_display.insert(tk.END, f"Bella: {response}\n")

self.chat\_display.see(tk.END)

self.speak(response)

if self.learning\_mode.get():

self.ask\_for\_review(user\_input, response)

def run(self):

self.display\_image()

self.root.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

bot = AnimeChatBot(root)

bot.run()

### **4.3 Code Explanation**

#### 4.3.1 Importing Libraries

The first step in the implementation is to import the necessary libraries. Each library serves a specific purpose:

import os

import cv2

import pygame

import tkinter as tk

from tkinter import scrolledtext, simpledialog

from gtts import gTTS

from transformers import pipeline, AutoModelForCausalLM, AutoTokenizer

import speech\_recognition as sr

import sqlite3

import pandas as pd

import csv

from threading import Thread

from PIL import Image, ImageTk

* **os**: Provides functions for interacting with the operating system, such as file handling.
* **cv2**: OpenCV library for handling video processing and capturing video frames.
* **pygame**: A library for playing audio files.
* **tkinter**: A standard GUI library in Python to create graphical interfaces.
* **scrolledtext** and **simpledialog**: Tkinter submodules for text display and dialog boxes.
* **gTTS**: Google Text-to-Speech library for converting text into speech.
* **transformers**: A library from Hugging Face for NLP and machine learning tasks, such as text generation.
* **speech\_recognition**: Library for recognizing and converting speech to text.
* **sqlite3**: A library to interact with SQLite databases for storing user interactions and feedback.
* **pandas**: Data manipulation library used for reading and processing CSV files.
* **csv**: Provides functions to read and write CSV files.
* **threading**: A module to run multiple threads (tasks) concurrently.
* **PIL (Pillow)**: Python Imaging Library to handle image processing and display.

#### 4.3.2 Initialization

In this section, the AnimeChatBot class is initialized with essential components and configurations.

class AnimeChatBot:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Bella ChatBot")

self.root.geometry("1080x720")

self.root.configure(bg="#2E3440")

self.root.resizable(True, True)

* **root**: The main window of the Tkinter application.
* **title**: Sets the window title.
* **geometry**: Defines the size of the window.
* **configure**: Sets the background color of the window.
* **resizable**: Allows the window to be resizable.

# Load the idle image and speaking video

self.idle\_image\_path = 'idle\_image.png'

self.speaking\_video\_path = 'speaking\_video.mp4'

# Initialize text-to-speech and speech recognition

self.recognizer = sr.Recognizer()

# Load the model and tokenizer

self.tokenizer = AutoTokenizer.from\_pretrained("distilgpt2")

self.model = AutoModelForCausalLM.from\_pretrained("distilgpt2")

self.text\_generator = pipeline("text-generation", model=self.model, tokenizer=self.tokenizer)

# Load the dataset

self.dataset\_path = 'dataset.csv'

self.load\_dataset()

# Initialize pygame for playing audio

pygame.init()

# Initialize database connection

self.db\_conn = sqlite3.connect('reviews.db')

self.db\_cursor = self.db\_conn.cursor()

# Farewell phrases

self.farewell\_phrases = ["goodbye", "bye", "see you", "farewell", "later", "ciao", "adios"]

# Mode toggle

self.learning\_mode = tk.BooleanVar()

self.learning\_mode.set(False)

# Create GUI elements

self.create\_widgets()

* **idle\_image\_path** and **speaking\_video\_path**: File paths for the idle image and speaking video.
* **recognizer**: Initializes the speech recognizer.
* **tokenizer** and **model**: Load the tokenizer and model from the transformers library.
* **text\_generator**: Creates a text generation pipeline using the loaded model and tokenizer.
* **dataset\_path**: File path for the dataset CSV file.
* **load\_dataset**: Method to load the dataset into memory.
* **pygame.init()**: Initializes pygame for audio playback.
* **db\_conn** and **db\_cursor**: Initialize the SQLite database connection and cursor.
* **farewell\_phrases**: A list of phrases that signify the end of a conversation.
* **learning\_mode**: A Tkinter BooleanVar to toggle between Learning Mode and Talking Mode.
* **create\_widgets**: Method to create the GUI elements.

#### 4.3.3 GUI Setup

The GUI elements are created and placed within the main window.

def create\_widgets(self):

# Frame for the image and video

self.image\_frame = tk.Frame(self.root, bg="#4C566A", bd=2, relief=tk.SUNKEN)

self.image\_frame.grid(row=0, column=0, rowspan=4, padx=10, pady=10, sticky="nsew")

self.idle\_image = Image.open(self.idle\_image\_path).resize((320, 480), Image.Resampling.LANCZOS)

self.idle\_image\_tk = ImageTk.PhotoImage(self.idle\_image)

self.image\_label = tk.Label(self.image\_frame, image=self.idle\_image\_tk, bg="#4C566A")

self.image\_label.pack(expand=True)

# Frame for the chat and controls

self.control\_frame = tk.Frame(self.root, bg="#2E3440")

self.control\_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

self.toggle\_button = tk.Checkbutton(self.control\_frame, text="Learning Mode", variable=self.learning\_mode, command=self.toggle\_mode, font=("Helvetica", 12), bg="#2E3440", fg="#D8DEE9", selectcolor="#4C566A")

self.toggle\_button.grid(row=0, column=0, sticky="w")

self.chat\_display = scrolledtext.ScrolledText(self.control\_frame, wrap=tk.WORD, width=60, height=20, font=("Helvetica", 12), bg="#3B4252", fg="#ECEFF4")

self.chat\_display.grid(row=1, column=0, columnspan=2, pady=10)

self.input\_box = tk.Entry(self.control\_frame, width=50, font=("Helvetica", 12))

self.input\_box.grid(row=2, column=0, pady=10, sticky="ew")

self.input\_box.bind("<Return>", lambda event: self.process\_input())

self.send\_button = tk.Button(self.control\_frame, text="Send", command=self.process\_input, font=("Helvetica", 12), bg="#5E81AC", fg="#ECEFF4")

self.send\_button.grid(row=2, column=1, padx=5)

self.speak\_button = tk.Button(self.control\_frame, text="Speak", command=self.speech\_input, font=("Helvetica", 12), bg="#5E81AC", fg="#ECEFF4")

self.speak\_button.grid(row=3, column=0, columnspan=2, pady=10)

self.root.grid\_rowconfigure(0, weight=1)

self.root.grid\_rowconfigure(1, weight=1)

self.root.grid\_rowconfigure(2, weight=1)

self.root.grid\_rowconfigure(3, weight=1)

self.root.grid\_columnconfigure(0, weight=1)

self.root.grid\_columnconfigure(1, weight=2)

* **image\_frame**: A frame for displaying Bella's image or video.
* **idle\_image**: Loads and resizes the idle image.
* **image\_label**: A label to display the idle image within the image\_frame.
* **control\_frame**: A frame for chat controls and display.
* **toggle\_button**: A checkbox to toggle between Learning Mode and Talking Mode.
* **chat\_display**: A scrolled text widget to display the chat history.
* **input\_box**: An entry widget for user text input.
* **send\_button**: A button to send the text input.
* **speak\_button**: A button to enable voice input.

#### 4.3.4 Function Definitions

This section explains the key functions used within the class.

**Loading the Dataset**

def load\_dataset(self):

self.dataset = pd.read\_csv(self.dataset\_path)

self.dataset\_dict = {q.lower(): a for q, a in zip(self.dataset['question'], self.dataset['answer'])}

* **load\_dataset**: Reads the dataset from the CSV file and stores it in a dictionary for quick lookup.

**Toggling Modes**

def toggle\_mode(self):

mode = "Learning Mode" if self.learning\_mode.get() else "Talking Mode"

self.chat\_display.insert(tk.END, f"Switched to {mode}\n")

* **toggle\_mode**: Switches between Learning Mode and Talking Mode, updating the chat display accordingly.

**Displaying the Idle Image**

def display\_image(self):

self.image\_label.configure(image=self.idle\_image\_tk)

* **display\_image**: Configures the image label to display the idle image.

**Playing the Video**

def play\_video(self):

cap = cv2.VideoCapture(self.speaking\_video\_path)

while pygame.mixer.music.get\_busy():

ret, frame = cap.read()

if not ret:

cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)

continue

frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frame = cv2.resize(frame, (320, 480))

frame = Image.fromarray(frame)

frame = ImageTk.PhotoImage(frame)

self.image\_label.configure(image=frame)

self.image\_label.image = frame

self.root.update\_idletasks()

self.root.update()

cap.release()

* **play\_video**: Captures and displays video frames while audio is playing.

**Generating Speech**

def speak(self, text):

tts = gTTS(text=text, lang='en')

tts.save("response.mp3")

pygame.mixer.music.load("response.mp3")

pygame.mixer.music.play()

Thread(target=self.play\_video).start()

while pygame.mixer.music.get\_busy():

self.root.update()

pygame.mixer.music.unload()

if os.path.exists("response.mp3"):

os.remove("response.mp3")

self.display\_image()

* **speak**: Converts text to speech, saves it as an MP3 file, and plays it using pygame.

**Recognizing Speech**

def recognize\_speech(self):

with sr.Microphone() as source:

self.chat\_display.insert(tk.END, "Listening...\n")

audio = self.recognizer.listen(source)

try:

return self.recognizer.recognize\_google(audio)

except sr.UnknownValueError:

return "Sorry, I did not understand that."

* **recognize\_speech**: Uses the speech recognition library to convert spoken input into text.

**Generating a Response**

def get\_response(self, user\_input):

user\_input\_lower = user\_input.lower()

for question in self.dataset\_dict:

if all(keyword in user\_input\_lower for keyword in question.split()):

return self.dataset\_dict[question]

response = self.text\_generator(user\_input, max\_length=30, num\_return\_sequences=1, truncation=True)

return response[0]['generated\_text'].strip()

* **get\_response**: Generates a response using the dataset or the text generation pipeline.

**Asking for Feedback**

def ask\_for\_review(self, question, answer):

self.chat\_display.insert(tk.END, f"Please rate the response (1-10):\n")

self.chat\_display.see(tk.END)

review = simpledialog.askinteger("Input", "Please rate the response (1-10):", parent=self.root, minvalue=1, maxvalue=10)

if review is not None:

if review >= 8:

self.save\_response(question, answer)

elif review < 5:

better\_answer = simpledialog.askstring("Input", "Please provide a better answer:", parent=self.root)

if better\_answer:

self.save\_response(question, better\_answer)

* **ask\_for\_review**: Requests user feedback and updates the dataset if necessary.

**Saving a Response**

def save\_response(self, question, answer):

question\_lower = question.lower()

if question\_lower in self.dataset\_dict:

self.chat\_display.insert(tk.END, f"Question '{question\_lower}' already exists in the dataset. Skipping save.\n")

self.chat\_display.see(tk.END)

return

self.dataset\_dict[question\_lower] = answer

with open(self.dataset\_path, 'a', newline='') as file:

writer = csv.writer(file)

writer.writerow([question\_lower, answer])

* **save\_response**: Saves new question-answer pairs to the dataset.

**Processing User Input**

def process\_input(self):

user\_input = self.input\_box.get()

self.input\_box.delete(0, tk.END)

self.chat\_display.insert(tk.END, f"User: {user\_input}\n")

self.chat\_display.see(tk.END)

if any(phrase in user\_input.lower() for phrase in self.farewell\_phrases):

self.speak("Goodbye! Have a great day!")

self.root.quit()

else:

response = self.get\_response(user\_input)

self.chat\_display.insert(tk.END, f"Bella: {response}\n")

self.chat\_display.see(tk.END)

self.speak(response)

if self.learning\_mode.get():

self.ask\_for\_review(user\_input, response)

* **Process\_input**: Processes text input, generates a response, and optionally asks for feedback.

**Handling Speech Input**

def speech\_input(self):

user\_input = self.recognize\_speech()

self.chat\_display.insert(tk.END, f"User: {user\_input}\n")

self.chat\_display.see(tk.END)

if any(phrase in user\_input.lower() for phrase in self.farewell\_phrases):

self.speak("Goodbye! Have a great day!")

self.root.quit()

else:

response = self.get\_response(user\_input)

self.chat\_display.insert(tk.END, f"Bella: {response}\n")

self.chat\_display.see(tk.END)

self.speak(response)

if self.learning\_mode.get():

self.ask\_for\_review(user\_input, response)

* **Speech\_input**: Processes speech input, generates a response, and optionally asks for feedback.

##### **4.3.5 Running the Application**

The main function runs the application:

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

bot = AnimeChatBot(root)

bot.run()

* **root**: Initializes the main Tkinter window.
* **AnimeChatBot(root)**: Creates an instance of the AnimeChatBot class.
* **bot.run()**: Starts the main loop of the Tkinter application, making the chatbot ready to interact with users.

### **CHAPTER 5:**

### **RESULTS**

The results section documents the outcomes of the implementation phase, detailing the testing and debugging processes that were undertaken to ensure the chatbot functions as expected. It also includes placeholders for screenshots to visually demonstrate the chatbot in action.

#### 5.1 Testing and Debugging

Testing and debugging are critical phases in the development lifecycle to ensure the chatbot performs as intended. This section outlines the approaches and methodologies used to test and debug Bella.

**5.1.1 Unit Testing**

Unit testing involves testing individual components or functions of the chatbot to ensure they work correctly in isolation. Key components tested include:

* **Speech Recognition**: Ensuring the speech recognition module accurately converts spoken input into text.
* **Text Generation**: Verifying the text generation pipeline produces coherent and contextually relevant responses.
* **Text-to-Speech (TTS)**: Checking that the TTS module generates clear and natural-sounding speech from text inputs.
* **Database Operations**: Confirming that interactions with the SQLite database (e.g., storing and retrieving feedback) are performed correctly.

Example Test Cases:

* Input: "Hello Bella" (spoken) -> Expected Output: "Hello! How can I assist you today?" (text and speech)
* Input: "Tell me a joke" (text) -> Expected Output: A humorous response (text and speech)

**5.1.2 Integration Testing**

Integration testing focuses on verifying that different components of the chatbot work together seamlessly. This includes:

* **GUI Interaction**: Ensuring that user inputs via the GUI (both text and voice) are correctly processed and that the responses are displayed appropriately.
* **Mode Switching**: Testing the toggle functionality between Learning Mode and Talking Mode to confirm that feedback is requested and processed correctly in Learning Mode.
* **Audio-Visual Synchronization**: Verifying that the video plays only while the chatbot is speaking and stops when the speech ends.

Example Test Cases:

* Toggle to Learning Mode, input a query, and provide feedback. Check if the feedback is stored and used in future interactions.
* Toggle to Talking Mode, input a query, and observe the response without feedback solicitation.

**5.1.3 User Acceptance Testing (UAT)**

User Acceptance Testing involves real users interacting with the chatbot to ensure it meets their needs and expectations. This testing phase includes:

* **Ease of Use**: Assessing the usability of the GUI and ensuring it is intuitive and user-friendly.
* **Response Quality**: Evaluating the relevance, coherence, and naturalness of the chatbot’s responses.
* **Performance**: Checking the responsiveness of the chatbot to user inputs and ensuring it operates without significant delays or crashes.

Example Test Cases:

* Conducting user surveys to gather feedback on the chatbot’s performance and usability.
* Monitoring the chatbot’s performance during extended interactions to identify any performance degradation.

**5.1.4 Debugging Process**

Debugging is an iterative process of identifying and fixing issues that arise during testing. Common issues encountered and resolved include:

* **Speech Recognition Errors**: Fine-tuning the speech recognition parameters and ensuring a quiet environment to minimize recognition errors.
* **Text Generation Inconsistencies**: Adjusting the text generation model’s parameters to improve the relevance and coherence of responses.
* **Audio-Visual Synchronization**: Ensuring the video playback is tightly synchronized with the speech output by using threading and efficient frame updates.
* **Database Conflicts**: Implementing checks to avoid duplicate entries and ensuring data integrity during simultaneous read/write operations.

**5.1.5 Tools and Techniques**

Various tools and techniques were employed during the testing and debugging phases, including:

* **Python’s unittest Framework**: For automated unit testing of individual functions and components.
* **Manual Testing**: For integration and user acceptance testing, ensuring real-world usability and performance.
* **Logging and Debugging Statements**: Adding detailed logging and print statements throughout the code to trace and debug issues.

#### 5.2 Screenshots

#### 

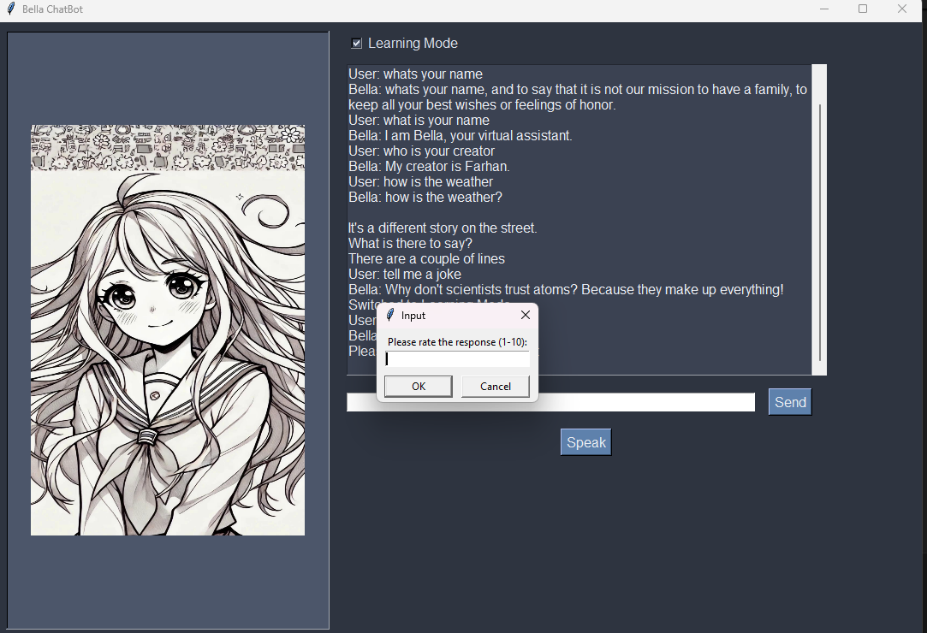
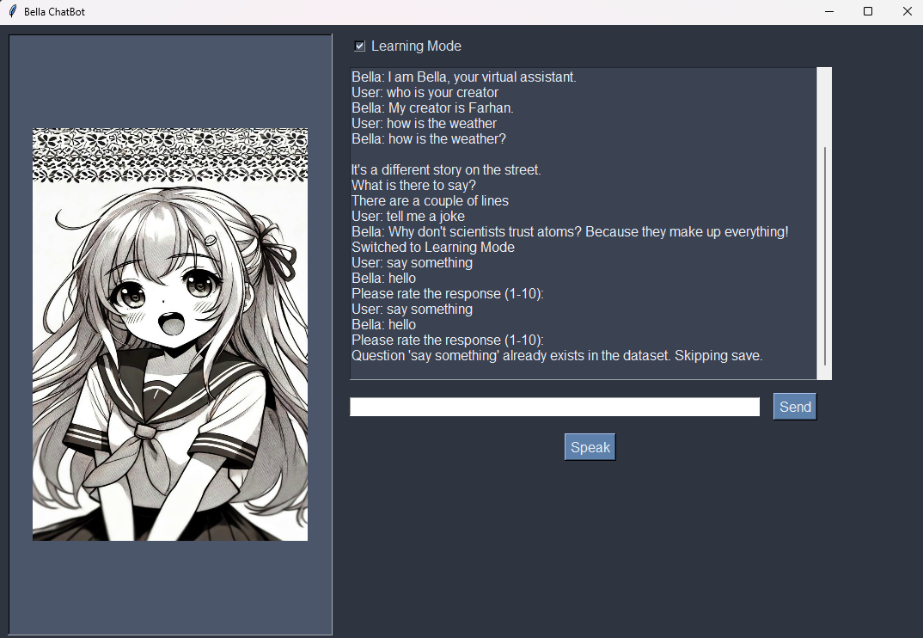


Fig 5.2.2 Depicts the Machine Learning Capabilities of the Bella Chat Bot

Fig 5.2.2 Depicts the review functionality of the Bella Chat Bot

Fig 5.2.1 Depicts the interaction capabilities of the Bella Chat Bot

Figure 5.2.1

Figure 5.2.1

Figure 5.2.1

**CONCLUSION**

#### Summary

The conclusion summarizes the achievements of the Bella AI Chatbot project and outlines potential areas for future enhancement. It reflects on the project's objectives, accomplishments, and the lessons learned during development.

The Bella AI Chatbot project aimed to create an interactive chatbot capable of engaging in text and voice conversations with users. Throughout the project, several key milestones were achieved:

**Development of an Interactive Chatbot**: Bella was successfully developed to interact with users through both text and voice. The chatbot can recognize spoken input, generate text responses, and convert text to speech, providing a comprehensive user experience.

**Voice Interaction Capabilities**: By integrating speech recognition and text-to-speech technologies, Bella can handle voice inputs and outputs effectively. This feature enhances accessibility and user engagement.

**Continuous Learning Mode**: The chatbot includes a Learning Mode, where it solicits feedback from users to improve its responses over time. This mode allows Bella to learn from user interactions and become more accurate and contextually aware.

**User-Friendly GUI**: A graphical user interface was designed using Tkinter, making it easy for users to interact with Bella. The interface includes input fields, buttons, and a display area for the chat history, ensuring a seamless user experience.

**Dual Modes (Learning and Talking)**: Bella operates in two modes: Learning Mode for improving responses based on user feedback and Talking Mode for engaging in conversations without requesting feedback.

**Real-Time Performance**: The chatbot was tested extensively to ensure it responds to user inputs in real-time. This was achieved through efficient processing of inputs, quick text generation, and synchronized audio-visual outputs.

#### FUTURE WORK

While the Bella AI Chatbot project achieved significant milestones, there are several areas for potential improvement and future enhancement. These include:

**Advanced Natural Language Understanding (NLU):**

* **Contextual Understanding**: Enhancing Bella's ability to understand and maintain context across multiple interactions would make conversations more coherent and natural. Implementing advanced NLP models, such as BERT or GPT-3, could improve contextual understanding.
* **Sentiment Analysis**: Integrating sentiment analysis to detect the user's emotional tone and respond appropriately. This would enable Bella to provide more empathetic and contextually aware responses.

**Enhanced Learning Capabilities:**

* **Automated Feedback Integration**: Developing algorithms to automatically integrate user feedback without manual intervention. This could involve using reinforcement learning techniques to optimize response generation.
* **Personalization**: Allowing Bella to learn and adapt to individual users' preferences and behaviors. Personalized interactions could improve user satisfaction and engagement.

**Expanded Functionality:**

* **Multilingual Support**: Expanding Bella's capabilities to understand and respond in multiple languages. This would make the chatbot accessible to a broader audience.
* **Domain-Specific Knowledge**: Training Bella with domain-specific knowledge to provide expert advice in areas such as healthcare, finance, or education.

**Improved User Interface:**

* **Responsive Design**: Developing a more responsive and adaptive GUI that works seamlessly across different devices, including smartphones and tablets.
* **Accessibility Features**: Implementing features such as voice control for the GUI, high contrast modes, and support for screen readers to make the chatbot more accessible to users with disabilities.

**Deployment and Scalability:**

* **Cloud Deployment**: Deploying Bella on cloud platforms to ensure scalability and handle a larger number of simultaneous interactions. This could involve using services like AWS, Google Cloud, or Azure.
* **API Integration**: Developing APIs to integrate Bella with other applications and services, such as messaging platforms, customer support systems, and IoT devices.

**Security and Privacy:**

* **Data Encryption**: Ensuring all user data is encrypted and securely stored to protect privacy.
* **Compliance with Regulations**: Adhering to data protection regulations such as GDPR or CCPA to ensure ethical handling of user data.

By addressing these areas in future work, Bella can evolve into a more advanced, user-friendly, and versatile chatbot, capable of providing richer interactions and meeting a wider range of user needs.

### **REFERENCES**

* Jurafsky, D., & Martin, J. H. (2020). Speech and Language Processing (3rd ed.). Pearson.
* Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press.
* Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
* Chollet, F. (2018). Deep Learning with Python. Manning Publications.
* Adamopoulou, E., & Moussiades, L. (2020). An overview of chatbot technology. Artificial Intelligence Applications and Innovations, 373-383.
* Singh, J., & Jain, N. (2020). Survey and classification of chatbot implementation techniques. Journal of Information and Optimization Sciences, 41(2), 507-530.
* Brownlee, J. (2017). A Gentle Introduction to the Bag-of-Words Model. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/gentle-introduction-bag-words-model/
* Goldberg, Y. (2016). A Primer on Neural Network Models for Natural Language Processing. arXiv preprint arXiv:1510.00726.
* Patel, S. (2018). Text to Speech (TTS) API in Python with gTTS. Medium. Retrieved from https://medium.com/analytics-vidhya/text-to-speech-tts-api-in-python-with-gtts-92f157fc0a60
* Pant, A. (2019). Building a Speech Recognition System Using Python. Towards Data Science. Retrieved from https://towardsdatascience.com/building-a-speech-recognition-system-using-python-52d876b2bdd8
* Raj, R. (2020). How to Build Your Own AI Chatbot with Python. DataCamp. Retrieved from https://www.datacamp.com/community/tutorials/building-a-chatbot-python
* Yegulalp, S. (2018). How to train your own chatbot with Python. InfoWorld. Retrieved from https://www.infoworld.com/article/3272921/how-to-train-your-own-chatbot-with-python.html

#### 

* Python Software Foundation. Python Language Reference, version 3.8. Available at <http://www.python.org>
* Numpy. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362.
* Transformers: Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. arXiv preprint arXiv:1910.03771. Available at <https://github.com/huggingface/transformers>
* Scikit-learn: Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Available at <https://scikit-learn.org/>
* Tkinter. Tkinter Documentation. Available at <https://docs.python.org/3/library/tkinter.html>
* Pillow (PIL): Clark, A. (2015). Pillow (PIL Fork) Documentation. Available at <https://pillow.readthedocs.io/en/stable/>
* OpenCV: Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools. Available at <https://opencv.org/>
* Pygame: Shinners, P. (2001). Pygame Documentation. Available at https://www.pygame.org/docs/
* SpeechRecognition: Anthony Zhang. SpeechRecognition Documentation. Available at <https://github.com/Uberi/speech_recognition>
* gTTS (Google Text-to-Speech): Karpathy, A. (2016). gTTS Documentation. Available at <https://pypi.org/project/gTTS/>
* SQLite: Hipp, W. (2000). SQLite Documentation. Available at https://www.sqlite.org/docs.html