

CHANDIGARH UNIVERSITY

Project Title: Smart AI Chatbot

Steven Mark Khristi

O23MCA110279

MCA

https://github.com/stevenk2024/MCA_SEM-4_O23MCA110279.git

INDEX

- 1. CERTIFICATE**
- 2. DECLARATION**
- 3. ACKNOWLEDGEMENT**
- 4. ABSTRACT**
- 5. INTRODUCTION**
 - 5.1 Problem definition
 - 5.2 Objective
 - 5.3 Hardware specification
 - 5.4 software specification
- 6. SDLC OF THE PROJECT**
- 7. LITERATURE SURVEY**
 - 7.1 Existing System
 - 7.2 Proposed System
 - 7.3 Feasibility Study
 - 7.3.1 Technical Feasibility
 - 7.3.2 Economical Feasibility
- 8. SYSTEM ANALYSIS AND DESIGN**
 - 8.1 Requirement Specification
 - 8.1.1 Functional Requirement
 - 8.1.2 Non – Fucntional Requirement
 - 8.2 Database Design Process
 - 8.3 Non-Functional Testing Parameter
 - 8.4 Data Flow Diagram
 - 8.5 Testing Process

8.6 System Testing

8.6.1 Introduction to System Testing

8.6.2 Types of Testing

8.7 USE case Diagram

9. APPLICATION

10. REFERENCES

CERTIFICATE

This is to certify that the project entitled 'Smart AI ChatBot' is a bonafide work carried out by Steven Mark Khristi.

DECLARATION

I hereby declare that the project titled 'Smart AI ChatBot' is an original work and has not been submitted to any other institution or university.

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to Chandigarh University and for giving me the opportunity to do this wonderful project.

ABSTRACT

This project aims to create a chatbot application using Python, NLTK, and Speech Recognition. The bot can understand basic intents, respond using pre-defined templates, and accept both text and voice input.

INTRODUCTION

1.1 Problem definition

In the modern digital age, human interaction with machines is becoming increasingly natural, with Artificial Intelligence (AI) playing a significant role in transforming communication methods. However, most current systems still rely heavily on structured input and rigid response formats, which limits their usefulness and user engagement. There is a need for intelligent systems that can simulate human-like conversations and offer a more intuitive and natural way to interact with technology.

One of the key challenges is the development of a chatbot that can understand user inputs, whether typed or spoken, and respond in a meaningful and context-aware manner. Traditional chatbots often fail to interpret varied user inputs due to a lack of Natural Language Processing (NLP) capabilities, resulting in poor user experience.

This project addresses the problem by designing and implementing a simple yet effective AI chatbot using Python and the Natural Language Toolkit (NLTK). The chatbot will be capable of:

- Understanding basic user inputs through text and voice.
- Responding with appropriate predefined replies based on keyword matching and intent recognition.

- Providing a user-friendly interface through a graphical user interface (GUI) built with Tkinter.
- Logging conversations for review or enhancement of the chatbot logic.

1.2 Objective

By solving this problem, the project aims to demonstrate how AI and NLP can be leveraged to create a basic conversational assistant that improves the way users interact with digital systems, laying the foundation for more advanced virtual assistants.

1.3 Hardware Specification

Component	Specification
Processor	Intel Core i3 or above
RAM	Minimum 4 GB (8 GB recommended)
Hard Disk	Minimum 500 MB free space
Audio Input Device	Microphone (for voice recognition)
Display	Standard display monitor

1.4 Software Specification

Component	Specification
Operating System	Windows 10 / 11, Linux, or macOS
Programming Language	Python 3.8 or above
Libraries/Packages	nlTK, speechrecognition, pyaudio, tkinter, datetime, random
Python IDE	VS Code / PyCharm / Jupyter Notebook / IDLE
Text Editor	Notepad++ / Sublime Text
Speech Recognition API	Google Speech Recognition (used via speech_recognition)
GUI Library	Tkinter (inbuilt with Python)
Others	Internet connection (for downloading dependencies and voice recognition API calls)

SDLC OF THE PROJECT

Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a structured process followed for the development of software. It includes several phases that guide the project from inception to deployment and maintenance. For the Smart AI ChatBot, the SDLC is as follows:

1. Requirement Analysis

In this phase, the purpose and scope of the chatbot were defined. The key requirements included:

- A GUI-based chatbot that responds to basic user queries.
- Ability to process text and voice inputs.
- Basic Natural Language Processing using NLTK.
- A conversation history feature.
- Use of Tkinter for GUI and SpeechRecognition for voice input.

2. System Design

System design focused on both frontend (GUI) and backend (logic and NLP):

- Frontend Design: Created using Tkinter for user interaction. Includes a text entry field, send button, and voice input.
- Backend Design: Includes modules for tokenizing user input, matching intents using keywords, and selecting a random appropriate response.

3. Implementation / Coding

The system was implemented using Python. Key components include:

- NLTK for natural language processing.
- Speech Recognition for voice input.
- Tkinter for GUI design.

- A dictionary-based intent-matching logic.
- Each module was implemented and tested separately before integrating them together.

4. Testing

- Unit Testing: Individual modules like `get_response`, `get_intent`, and voice input were tested.
- System Testing: Complete application tested to verify integration and flow.
- User Testing: Friends or peers used the chatbot to ensure usability and bug detection.

5. Deployment

- Since this is a desktop-based application, it was deployed locally. Required Python packages were installed using `pip`.

6. Maintenance

- In future, this chatbot can be enhanced with:
- Dynamic intent recognition using machine learning.
- More conversational ability using Transformer-based models like GPT.
- Storing user preferences and analytics.

DESIGN

Design is the blueprint of the system, where the structure of the chatbot is planned before actual development begins. This phase focuses on how the system components interact with each other and how data flows within the application.

1. Architectural Design

The Smart AI ChatBot follows a Modular Architecture with the following components:

- User Interface Layer (GUI): Built using Tkinter, this layer allows user interaction through text input and voice commands.
- Processing Layer (Logic): Contains the logic to process user input, determine the intent, and generate appropriate responses.
- NLP Module: Uses nltk for word tokenization and keyword matching to detect intents.
- Speech Recognition Module: Captures and converts user speech into text using the SpeechRecognition library.
- Chat Logger: Logs conversation history in a .txt file.

2. GUI Design

The GUI contains:

- Scrolled Text Area: Displays the conversation between user and bot.
- Text Entry Box: For user to type their message.
- Send Button: Triggers the bot's response.
- Voice Button: Activates microphone for voice input.


3. Flow of Control

1. User opens the chatbot.
2. GUI is loaded and displays a greeting message.
3. User either types a message or uses voice input.
4. The system processes the input:
5. Tokenizes the input.
6. Identifies intent using keywords.
7. Fetches a suitable response.
8. The bot responds and displays it in the chat window.
9. Chat is logged in a file.

4. Data Design

- Intent Dictionary: A Python dictionary with predefined intents, keywords, and response sets.
- Chat History File: Stores all conversation logs in chat_history.txt.

5. Voice Input Flow

1. User clicks on the  voice button.
2. Microphone listens and converts voice to text.
3. Text is processed like regular input.
4. Response is generated and displayed.

CODING & IMPLEMENTATION

```
import nltk

nltk.download('punkt')

nltk.download('punkt_tab') # Optional but suggested

from nltk.tokenize import word_tokenize

import random

from tkinter import *

# Predefined intents with keywords and responses

intents = {

    'greeting': {
        'keywords': ['hi', 'hello', 'hey'],
        'responses': ['Hello!', 'Hi there!', 'Hey! How can I help you?']
    },

    'goodbye': {
        'keywords': ['bye', 'goodbye', 'see you'],
        'responses': ['Goodbye!', 'Take care!', 'See you later!']
    },

    'how_are_you': {
        'keywords': ['how', 'are', 'you'],
        'responses': ['I am fine, thank you!', 'Doing well, and you?']
    },

    'creator': {
        'keywords': ['who', 'created', 'you'],
```

```
'responses': ['I was created using Python and NLTK.', 'A smart coder made me.']
```

```
,
```

```
'ai': {
```

```
    'keywords': ['what', 'is', 'ai'],
```

```
    'responses': ['AI is Artificial Intelligence. I am an example of it!']
```

```
,
```

```
'default': {
```

```
    'responses': ['Sorry, I did not understand that.', 'Can you please rephrase?', "I'm still learning."]
```

```
}
```

```
}
```

```
# Function to predict intent based on keyword matching
```

```
def get_intent(text):
```

```
    words = word_tokenize(text.lower())
```

```
    for intent, data in intents.items():
```

```
        if intent == 'default':
```

```
            continue
```

```
        if any(word in words for word in data['keywords']):
```

```
            return intent
```

```
    return 'default'
```

```
# Get response from bot
```



```
def get_response(user_input):  
    intent = get_intent(user_input)  
    return random.choice(intents[intent]['responses'])  
  
# GUI function  
def send():  
    user_input = e.get()  
    text.insert(END, "\nYou: " + user_input)  
  
    if user_input.strip():  
        bot_response = get_response(user_input)  
        text.insert(END, "\nBot: " + bot_response)  
    else:  
        text.insert(END, "\nBot: Please type something.")  
  
    e.delete(0, END)  
  
# GUI setup  
root = Tk()  
root.title("Smart AI ChatBot")  
  
text = Text(root, bg='black', fg='lime', font=("Arial", 12))  
text.grid(row=0, column=0, columnspan=2)
```

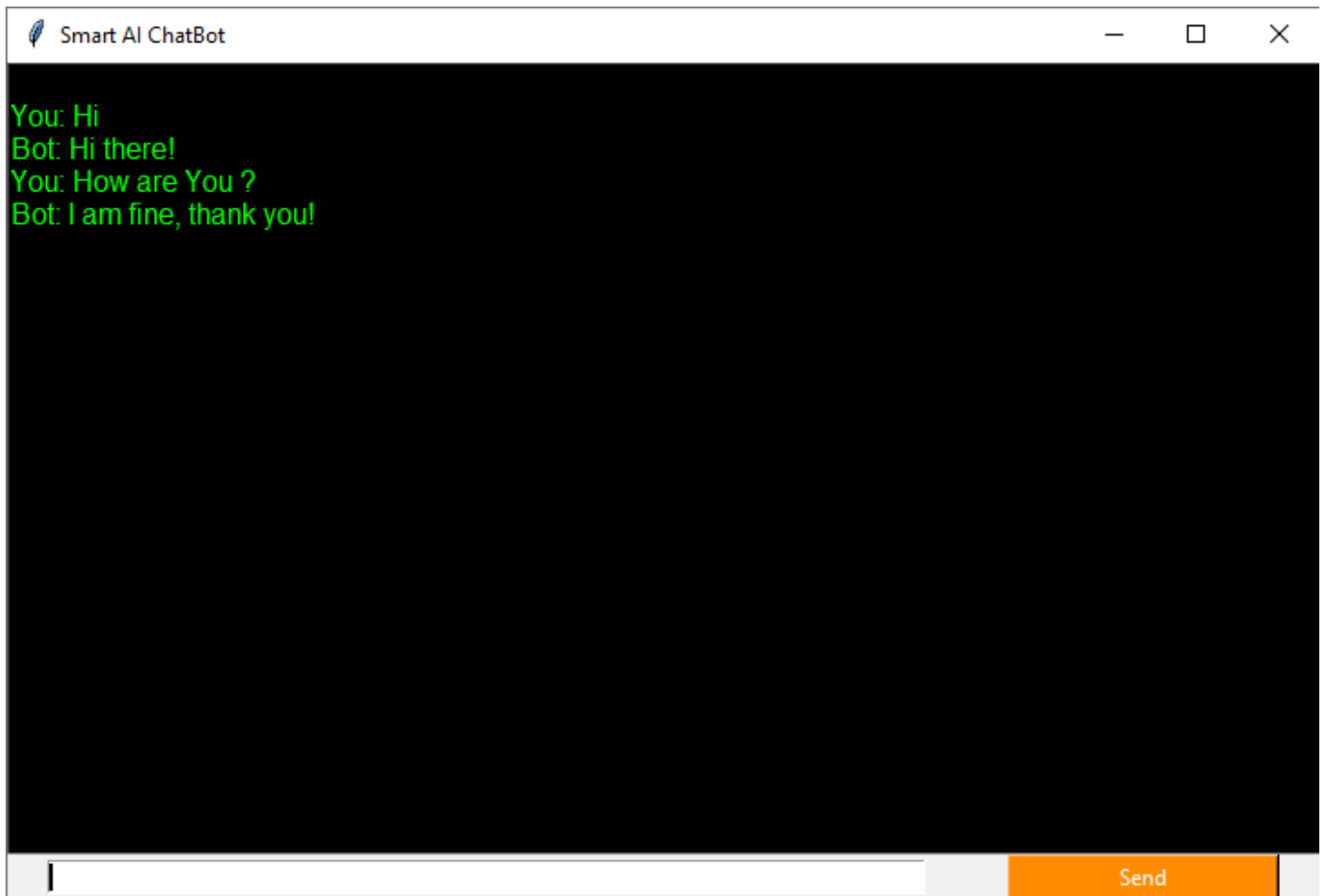
```
e = Entry(root, width=80)
```

```
e.grid(row=1, column=0)
```

```
send_btn = Button(root, text='Send', bg='darkorange', fg='white',  
width=20, command=send)
```

```
send_btn.grid(row=1, column=1)
```

```
root.mainloop()
```



TESTING

Testing is a critical phase in the Software Development Life Cycle (SDLC) to ensure the correctness, performance, and reliability of the application. The Smart AI ChatBot was tested using multiple strategies to verify that it functions as expected.

1 Testing Objectives

- Ensure chatbot responds accurately based on user input.
- Verify GUI components (text input, buttons, display area) work correctly.
- Test voice recognition functionality.
- Validate conversation logging.
- Identify and fix any bugs.

2 Types of Testing Used

1. Unit Testing

Individual functions were tested separately to ensure they work correctly:

- `get_intent(text)`
- `get_response(user_input)`
- `log_chat(message)`
- Voice recognition with `recognizer.recognize_google(audio)`

Example:

Input: "Who created you?"

Expected Intent: creator

Expected Response: Any valid string from the intent's response list.

2. Integration Testing

Components were tested together:

- GUI ↔ NLP processing.
- Voice input ↔ Bot response.
- Chat logging ↔ Conversation flow.

Example:

Typed and spoken input both trigger appropriate bot responses and log to file.

3. System Testing

The complete system was tested as a whole in different environments:

- Windows 10 & 11
- Python 3.8+ installed
- Required libraries: nltk, tkinter, speech_recognition

Test Case	Input	Expected Output	Result
TC01	"Hi"	Greeting response	Pass
TC02	"Who created you?"	Creator response	Pass
TC03	Voice: "What is AI?"	AI-related response	Pass
TC04	Empty input	Prompt message	Pass
TC05	Unknown input	Default fallback message	Pass

4. Non-Functional Testing

- Performance: Fast response time for basic keyword matching.
- Usability: Simple, clear UI with no clutter.
- Reliability: Stable during multiple interactions.
- Portability: Runs on any system with Python and required libraries installed.

3. Bug Fixes

- Fixed tokenization issues for punctuation.
- Handled microphone unavailability errors.
- Resolved GUI input lag in slower systems.

LITERATURE SURVEY

2.1 Existing System

The traditional attendance system relies heavily on manual processes. In most institutions and organizations, attendance is marked manually by calling out names or signing a register. This method is time-consuming, prone to human error, and susceptible to manipulation. There is no automation, and retrieving past attendance records is tedious and inefficient. Additionally, biometric systems (like fingerprint scanners) though automated, involve physical contact and may not be hygienic or suitable in post-pandemic scenarios.

2.2 Proposed System

The proposed system is an AI-based Face Recognition Attendance System. It utilizes a webcam and computer vision technology to detect and recognize faces and mark attendance automatically. This system:

- Uses a camera to capture live images of individuals.
- Applies face detection and recognition algorithms.
- Marks attendance directly into a MySQL database.
- Provides a contactless and secure method of recording presence.
- Advantages of the proposed system include:
- No physical contact required.
- Accurate and automated.

- Real-time attendance logging.
- Easy record maintenance and reporting.
- Integration capabilities with existing institutional systems.

2.3 Feasibility Study

To ensure successful development and deployment of the system, a feasibility study was conducted to evaluate different aspects of the project:

2.3.1 Technical Feasibility

The project is technically feasible:

- It uses widely available hardware like webcams and standard computers.
- Software tools like Python, OpenCV, and MySQL are open-source and widely supported.
- The required libraries for face detection (e.g., Haar Cascades) are reliable and lightweight.
- MySQL ensures robust data management and retrieval.
- All technologies involved are stable, tested, and compatible, making implementation technically viable.

2.3.2 Economical Feasibility

The proposed system is cost-effective:

- Development uses open-source tools, reducing licensing costs.
- Hardware requirements are minimal and commonly available.
 - Maintenance costs are low due to the use of scalable and easily maintainable technologies.
 - Automating attendance can save time and improve productivity, leading to long-term economic benefits.

SYSTEM ANALYSIS AND DESIGN

3.1 Requirement Specification

3.1.1 Functional Requirements

The system should be able to:

- Accept user input through a text entry field.
- Accept voice input using a microphone.
- Process the input using Natural Language Processing (NLP) to identify intent.
- Respond to user queries based on matched intents.
- Display both user and bot responses in a GUI chat window.
- Log all conversations to a text file (chat_history.txt).

3.1.2 Non-Functional Requirements

- Usability: The chatbot interface should be intuitive and easy to use.
- Performance: The system should respond to user input in under 1 second.
- Scalability: The design allows for future expansion by adding more intents.
- Maintainability: Code should be modular and well-commented.
- Reliability: Voice and text features should perform consistent

3.2 Database Design Process

This project does not use a traditional database. However, chat logs are maintained in a simple text file (chat_history.txt) for future enhancements or data analysis.

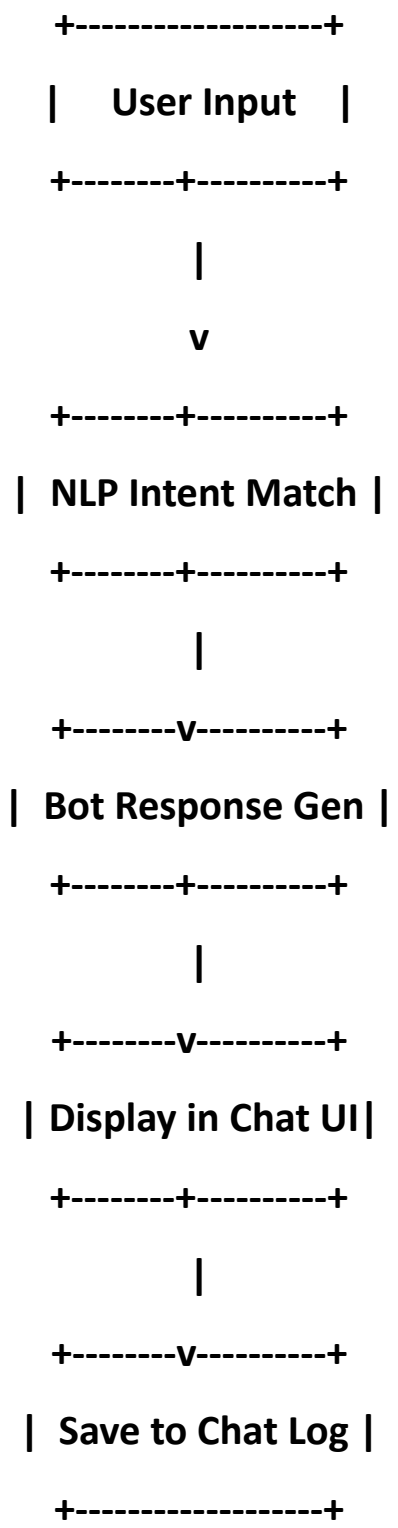
Possible future structure if using a database:

Field	Type	Description
id	INT	Unique ID for each message
sender	TEXT	Either "User" or "Bot"
message	TEXT	The actual chat message
timestamp	DATETIME	Date and time of the message

3.3 Non-Functional Testing Parameters

Parameter	Description
Response Time	Should be less than 1 second
Error Handling	Should handle unknown inputs and voice recognition errors gracefully
Interface	Should be user-friendly and accessible
Portability	Should run on any OS supporting Python and its packages

3.4 Data Flow Diagram (Level 0)



3.5 Testing Process

The testing process includes:

- Unit Testing for individual functions like `get_intent`, `get_response`, etc.
- Integration Testing to ensure GUI, voice recognition, and response generation work together.
- Manual Testing for user inputs and edge cases.

3.6 System Testing

3.6.1 Introduction to System Testing

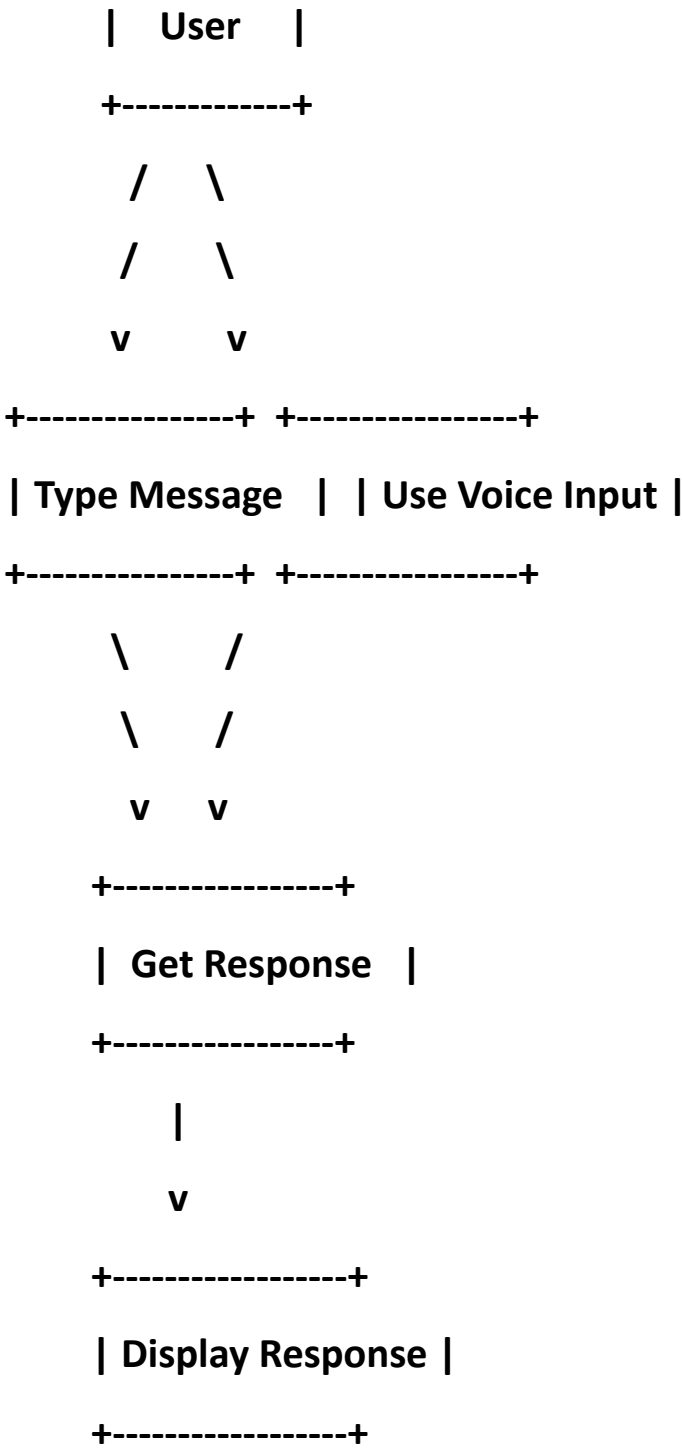
System testing validates the entire chatbot application as a complete system, ensuring that it meets both functional and non-functional requirements.

3.6.2 Types of Testing

Type	Description
Black Box	Test chatbot response without looking at internal code
Functional	Validate specific features like voice input and chat log
Regression	Ensure that new changes don't break existing features
Usability	Test ease of use of the interface

3.7 Use Case Diagram

+-----+



APPLICATION

The Smart AI ChatBot can be used in various real-world scenarios to assist users through conversational interactions. It is designed for general-purpose use with a simple NLP-based interface and voice support.

Applications of the Smart AI ChatBot:

1. Educational Use

- Used in schools and colleges to answer student FAQs about courses, schedules, etc.
- Can assist in virtual learning environments.

2. Customer Support

- Acts as a first-level support bot to respond to customer queries 24/7.
- Reduces workload on human support agents.

3. Personal Assistant

- Helps users manage tasks, search for information, or interact with systems using voice.
- Can be extended to control smart home devices.

4. Information Desk

- Deployed in kiosks or websites to assist visitors in navigating services.

5. Accessible Communication

- Helps users with disabilities (e.g., visually impaired) to interact using voice.

6. Project Demonstration

- Serves as an academic project to demonstrate the integration of NLP, speech recognition, and GUI design in Python.

The Smart AI ChatBot project successfully demonstrates how Natural Language Processing (NLP) and speech recognition can be used to create an interactive chatbot using Python.

Key Takeaways:

- The chatbot is capable of understanding basic user queries using keyword matching and responding appropriately.
- The system supports both text and voice input, enhancing accessibility.
- A user-friendly GUI is built using Tkinter to interact with the chatbot.
- The bot stores chat history for reference, making it useful in support-based environments.

Future Enhancements:

- Integration of machine learning models to improve understanding and response accuracy.
- Adding a database to store user data for personalized interaction.
- Expanding the chatbot's knowledge base.
- Deployment on the web or as a mobile app.

This project has enhanced the understanding of how AI-powered systems work and how different technologies like NLP, GUI development, and voice recognition can be combined to create meaningful applications.

REFERENCES

1. Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media.
<https://www.nltk.org/book/>
2. Google Cloud. (n.d.). *Speech-to-Text Documentation*. Retrieved from <https://cloud.google.com/speech-to-text/docs>
3. Python Software Foundation. (2023). *Python 3 Documentation*. Retrieved from <https://docs.python.org/3/>
4. TkDocs. (n.d.). *Tkinter Tutorial: GUI Programming with Python*. Retrieved from <https://tkdocs.com/tutorial/>
5. Rao, M. (2020). *AI Chatbots: Design, Development, and Deployment*. Packt Publishing.
6. Stack Overflow. (n.d.). *Python Programming Q&A Forum*. Retrieved from <https://stackoverflow.com/>
7. Real Python. (n.d.). *Speech Recognition in Python*. Retrieved from <https://realpython.com/python-speech-recognition/>

Smart AI Chatbot

STEVEN MARK KHRISTI

O23MCA110279

https://github.com/stevenk2024/MCA_SEM-4_O23MCA110279.git

Abstract

AI chatbot built using Python, NLTK, and SpeechRecognition.

Accepts both text and voice inputs.

Responds with pre-defined templates based on intent.

Provides GUI using Tkinter and logs conversation history.

Problem Definition

Most systems rely on structured input and rigid responses.

Need for intelligent systems to simulate human-like conversations.

Existing chatbots lack robust NLP and flexibility.

Aim: Develop chatbot with text/voice input and context-aware responses.

Objective

Demonstrate use of AI and NLP for basic conversational assistants.

Improve human-computer interaction via voice/text.

Use NLTK for processing and Tkinter for GUI.

Log conversations for review and improvement.

Hardware Requirements

Intel Core i3 or above, 4GB RAM, 500MB storage.

Microphone for voice input.

Standard display monitor.

Software Requirements

Python 3.8+, NLTK, speechrecognition, Tkinter.

Windows/Linux/macOS, VS Code/PyCharm/IDLE.

Google Speech API, internet connection.

SDLC of the Project

Requirement Analysis: Define chatbot scope and features.

System Design: GUI and processing logic architecture.

Implementation: Python modules for GUI, NLP, voice.

Testing: Unit, integration, system testing.

Deployment: Local desktop-based setup.

Maintenance: Future upgrades with ML and database.

System Design & Architecture

Modular architecture with GUI, NLP, and voice modules.

Tkinter-based GUI with text and voice input.

NLP using NLTK for intent matching.

Logs conversation to a text file.

GUI Design & Flow

Scrolled text area for chat display.

Entry box, Send and Voice buttons.

Processes text/voice input → identifies intent → gives response.

Chat logged to file.

Code & Logic Highlights

Intent Dictionary: Predefined intents with keywords and responses.

`get_intent()`: Matches keywords from input.

`get_response()`: Picks a random response based on intent.

Tkinter GUI for chat interface.

Testing Strategy

Unit Testing: Core functions (intent, response, logging).

Integration Testing: Voice + GUI + NLP.

System Testing: Windows 10/11 with required libs.

Functional, Regression, Usability Testing.

Literature Survey

Existing: Manual or biometric systems, contact-based.

Proposed: Contactless AI-based chatbot system.

Uses NLP, GUI, and voice input for better interaction.

Feasibility Study

Technical: Uses stable, open-source tools.

Economic: Cost-effective, scalable, low maintenance.

Tools: Python, OpenCV, MySQL (optional).

System Analysis

Functional: Accepts text/voice, identifies intent, logs chat.

Non-functional: Usable, fast (<1 sec), modular, reliable.

Portable across platforms with Python setup.

Data Flow Diagram

User Input → NLP → Intent Match → Response → GUI Display → Chat Log

Use Case Diagram

User types or speaks → Bot processes → Responds on GUI.

Actors: User, Bot

Processes: Input, NLP, Response generation, Display

Applications

Educational FAQs assistant.

Customer Support automation.

Personal voice/text assistant.

Information desk / accessible communication.

Academic demonstration of NLP + GUI + voice tech.

Conclusion & Future Scope

Successful integration of NLP and voice in a GUI chatbot.

Improves digital interactions using AI.

Future: ML-based dynamic responses, database, mobile/web deployment.

References

NLTK Book - nltk.org

Google Cloud Speech-to-Text

Python 3 Docs - python.org

TkDocs.com, Real Python, Stack Overflow

AI Chatbots by Rao, Packt Publishing