A

INDUSTRIAL ORIENTED MAJOR PROJECT STAGE – I REPORT

ON

BRUCE - A NEXT GENERATION VOICE ASSISTANT FOR ENHANCED USER INTERACTION AND TASK EXECUTION

Submitted to JNTUH in the partial fulfillment of the Academic Requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

BY

P.GANESH	**(21QM1A0599)**
V.JAINATH	**(21QM1A05C6)**
S.RISHIKESH REDDY	**(21QM5A05B5)**
S.CHARAN REDDY	**(21QM1A05A8)**
V.KRISHNA	**(21QM5A05C2)**

UNDER THE GUIDANCE OF

Mr. SAI KRISHNA

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KG REDDY College of Engineering & Technology AN AUTONOMOUS INSTITUTION

DEPARTMENT OFCOMPUTER SCIENCE AND ENGINEERING

KG REDDYCOLLEGE OF ENGINEERING AND TECHNOLOGY

(Accredited by NAAC, Approved by AICTE, New Delhi, Affiliated to JNTUH,Hyderabad)

Chilkur (Village), Moinabad (Mandal), R.RDist.,TG-501504 2024-25

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY
(Accredited by NAAC, Approved by AICTE, New Delhi, Affiliated to JNTUH,Hyderabad)
Chilkur (Village), Moinabad (Mandal),R. R Dist,TS-501504

CERTIFICATE

This is to certify dissertation report entitled **"BRUCE – A NEXT GENERATION VOICE ASSISTANT FOR ENHANCED USER INTERACTION AND TASK EXECUTION"**is the bonafide work of P.GANESH, (21QM1A0599), V.JAINATH (21QM1A05C6), S.RISHIKESH REDDY (21QM1A05B5) , S.CHARAN REDDY (21QM1A05A8), V KRISHNA (21QM1A05C2), who carried out the project under the supervision and in the partial fulfillment of the requirements for the award of B. Tech **"COMPUTER SCIENCE AND ENGINEERING"** from **"KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY"**, affiliated to Jawaharlal Nehru Technological University, Hyderabad.

INTERNAL GUIDE	**HEAD OF THE DEPARTMENT**
Mr. Sai Krishna	Dr. L. Raghu Kumar
(Assistant Professor)	(Associate Professor)
Department of CSE	HOD
	Department of CSE

EXTERNAL EXAMINER

Date:

DECLARATION

We hereby declare that the work described in this report, entitled **"BRUCE – A NEXT GENERATION VOICE ASSISTANT FOR ENHANCED USER INTERACTION AND TASK EXECUTION"** which is being submitted by us under the guidance of **Mr. SAI KRISHNA**, Assistant Professor of the CSE Department, KGRCET in partial fulfillment of the requirements for the award of **BACHELOR OF TECHNOLOGY** in the Department of **COMPUTER SCIENCE & ENGINEERING** during the academic year 2023-2024 to the **KG REDDY COLLEGE OF ENGINEERING & TECHNOLOGY**, Hyderabad affiliated to Jawaharlal Nehru Technological University Hyderabad. Hyderabad (T.S).

We also declare that this project results from our effort and has not been copied or intimated from any source. Citations from any websites are mentioned in the bibliography.

This work was not submitted earlier at any other university for the award of any degree.

Place: Hyderabad

Date:

By:

PULIMISHETTY GANESH	(21QM1A0599)
VEETLA JAINATH	(21QM1A05C6)
RISHIKESH REDDY	(21QM5A05B5)
S CHARAN REDDY	(21QM1A05A8)
V KRISHNA	(21QM5A05C2)

ACKNOWLEDGMENT

First and foremost, we would like to express our profound sense of gratitude to our Director **Dr. ROHIT KANDAKATLA** of **KG REDDY COLLEGE OF ENGINEERING & TECHNOLOGY**, Chilkur, Hyderabad for his guidance.encouragement and for all the facilities to complete this project. We have immense pleasure in expressing thanks and deep sense of gratitude valuable time with us and laying down his valuable suggestions to complete the project successfully on time.

Our sincere thanks to our guide **MR. SAI KRISHNA**, Assistant Professor of the Department of Computer Science and Engineering and also the principal of **KG REDDY COLLEGE OF ENGINEERING & TECHNOLOGY**, Chilkur village, Hyderabad has given his full effort in guiding the team, achieving the goal as well as her encouragement to maintain our track progress.

We indebted to our major project coordinator **Mr. SURENDRA TRIPATHI** Assistant Professor of Computer Science and Engineering Department for his excellent guidance, constant inspiration and encouragement in completion of this dissertation in our college.

We sincerely thank Dr. L. RAGHU KUMAR, Head of the Department, Department of Computer Science and Engineering for his valuable suggestions and advices during our project completion.

We would like to express our thanks to all the faculty members, staff of Department of Computer Science and Engineering, who have rendered valuable help in making this project a successful one.

PULIMISHETTY GANESH	(21QM1A0599)
VEETLA JAINATH	(21QM1A05C6)
RISHIKESH REDDY	(21QM5A05B5)
S CHARAN REDDY	(21QM1A05A8)
V KRISHNA	(21QM5A05C2)

ABSTRACT

The project aims to develop a personal-assistant. Bruce draws its inspiration from virtual assistants like Siri for iOS. It has been designed to provide a user-friendly interface for carrying out a variety of tasks by employing certain well-defined commands. Users can interact with the assistant either through voice commands or using keyboard input. As a personal assistant, Bruce assists the end-user with day-to-day activities like general human conversation, searching queries in google, bing or yahoo, searching for videos, retrieving images, live weather conditions, word meanings, searching for medicine details, health recommendations based on symptoms and reminding the user about the scheduled events and tasks. The user statements/commands are analyzed with the help of machine learning to give an optimal solution.

INDEX

LIST OF ABBRIVATIONS	FULL FORM
VRI	VOICE RECOGNATION INTERFACE
DMI	DIALOG MANAGEMENT INTERFACE
NLU	NATURAL LANGUAGE UNDERSTANDING
TTS	TEXT TO SPEECH
DFD	DATA FLOW DIAGRAM

CHAPTER -1

INTRODUCTON

1.1 Purpose

1.2

In this project, we propose a BRUCE that performs task according to the voice input. Automatic Speech Recognition(ASR), identifies spoken words and phrases and translates them to a readable computer format also known as speech-to-text conversion. BRUCE takes the input of the user in the form of a voice and processes it and returns the feedback in different ways. iPhone has created "Siri" and Windows has created "Cortana" Amazon has created "Alexa" to help users respond to their voice commands in an effective manner.

With continuous advancements in artificial intelligence, natural language processing, speech recognition, and text-to-speech technologies, virtual assistants are transforming the way commercial organizations interact with customers. For instance, product websites are increasingly implementing chat-bot assistants to answer frequently asked questions and common customer complaints.

In this, a computer system is designed in such a way that typically requires interaction from human. As we know Python is an emerging language so it becomes easy to write a script for Voice Assistant in Python. The instructions for the assistant can be handled as per the requirement of user.

1.2 Scope

The scope of a Next Generation Voice Assistant for Enhanced User Interaction and Task Execution involves developing an advanced AI-driven system that goes beyond basic voice recognition. This assistant will leverage natural language processing (NLP), machine learning, and contextual understanding to facilitate seamless and intuitive interactions. The assistant aims to handle complex commands, understand nuanced user intent, and provide personalized responses based on individual user preferences and behavioral patterns. Additionally, it will support multitasking, allowing users to manage multiple tasks and workflows without disruption, and integrate with various applications, devices, and services, enhancing productivity and accessibility across platforms. The ultimate goal is to create a conversational, proactive, and adaptable voice assistant that can anticipate user needs and act as a comprehensive digital companion, meeting the demands of both personal and professional tasks.

1.3 GOALS

The goal of the personal voice assistant project is to design and develop an intelligent and user-friendly voice assistant that can assist individuals in automating tasks, retrieving information, controlling smart home devices, and providing personalized experiences. The primary objective of the project is to create a voice assistant that seamlessly integrates into users' daily lives and enhances their productivity and convenience through voice interactions.

1.4 FEATURES OF THE PROJECT

1. Personalized Interaction

- **Contextual Understanding**: Remembers previous conversations and adjusts responses based on user preferences and history.
- **User Profiles**: Supports multiple users with individual preferences and voice recognition for customized interactions.
- **Adaptive Tone and Style**: Adjusts tone and language based on user’s mood and preferences.

2. Enhanced Natural Language Processing (NLP)

- **Multi-turn Conversations**: Engages in complex, multi-step dialogues to handle tasks that require multiple inputs.
- **Dynamic Context Management**: Handles context shifts, interruptions, and returns to previous topics smoothly.
- **Sentiment Analysis**: Detects emotional cues to offer empathetic responses or take appropriate action.

3. Advanced Task Execution

- **Multitasking Capabilities**: Can perform multiple tasks in parallel, such as scheduling, reminders, and controlling smart home devices.
- **Automated Routine Creation**: Suggests and automates daily routines based on user behavior and preferences.
- **Intelligent Task Management**: Prioritizes tasks based on urgency, deadlines, and user-defined rules.

4. Cross-Platform Integration

- **Omnichannel Access**: Accessible on various devices, including smartphones, wearables, computers, and smart home devices.
- **App and Service Integration**: Integrates with third-party apps (e.g., messaging, calendar, social media) to offer seamless interaction.
- **Smart Home Control**: Connects with IoT devices (e.g., lights, thermostats, security systems) for comprehensive home automation.

5. Proactive Assistance

- **Predictive Suggestions**: Anticipates needs based on patterns and offers relevant information or actions.
- **Health and Wellness Monitoring**: Tracks health metrics and offers recommendations, reminders, and alerts for wellness.
- **Real-Time Updates**: Provides updates on traffic, weather, news, and other dynamic information tailored to user preferences.

6. Security and Privacy

- **Voice Authentication**: Uses secure voiceprint recognition to identify users and ensure privacy.
- **Data Encryption**: Ensures end-to-end encryption for user data and conversations.
- **Customizable Privacy Settings**: Allows users to control data collection, storage, and sharing preferences.

7. Interactive Learning and Development

- **Knowledge Expansion**: Continuously learns from user interactions to expand knowledge base and improve responses.
- **Real-Time Feedback Loop**: Allows users to provide feedback on responses for continuous improvement.
- **Adaptive Learning for Skills**: Adapts over time to become more proficient at frequently requested skills and tasks.

8. Language and Accent Adaptability

- **Multilingual Support**: Understands and responds in multiple languages, with options for real-time translation.
- **Accent Recognition**: Recognizes different accents and dialects, enhancing accessibility across regions.
- **Code-Switching Ability**: Allows switching between languages within a single conversation if needed.

9. Visual Interface Option

- **Augmented Reality (AR) Integration**: Supports AR overlays for visual information on enabled devices.
- **Interactive Display Integration**: Pairs with screens to provide visual aids, charts, or maps where applicable.
- **Gesture Recognition**: Recognizes gestures for commands on supported devices.

10. Accessibility and Inclusivity

- **Hearing and Vision Impaired Modes**: Offers speech-to-text for the hearing impaired and voice amplification for the visually impaired.
- **Customizable Text and Voice Output**: Allows users to adjust voice speed, tone, and volume for improved comprehension.
- **Assistive Technology Integration**: Compatible with screen readers, braille displays, and other assistive devices.

CHAPTER .2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

We are familiar with many existing voice assistants like Alexa, Siri, Cortana which uses concept of language processing, and voice recognition. They listens the command given by the user as per their requirements and performs that specific function in a very efficient and effective manner. These assistants are no less than a human assistant but we can say that they are more effective and efficient to perform any task. The algorithm used to make these assistant focuses on the time complexities and reduces time.

But for using these assistants one should have an account (like Microsoft account for Cortana)which is mandatory and can use it with internet connection only because these assistants are going to work with internet connectivity. They are integrated with many devices like, phones, laptops, and speakers etc.

2.2 PROPOSED SYSTEM

It was an interesting task to make my own assistant. It became easier to send emails without typing any word, Searching on Google without opening the browser, and performing many other daily tasks like playing music, opening your favorite IDE with the help of a single voice command. BRUCE is different from other traditional voice assistants in terms that it is specific to desktop and user does not need to make account to use this, it does not require any internet connection while getting the instructions to perform any specific task.

The IDE used in this project is Visual Studios. All the python files were created in Visual Studios and all the necessary packages were easily installable in this IDE. For this project following modules and libraries were used i.e. pyttsx3, Speech Recognition, Date time, Wikipedia, Smtplib, pywhatkit, pyjokes, pyPDF2, pyautogui, pyQt etc. I have created a live GUI for interacting with the BRUCE as it gives a design and interesting look while having the conversation.

2.3 OVERALL DESCRIPTION

The Next Generation Voice Assistant (NGVA) is a cutting-edge AI-driven system designed to improve user interaction and streamline task execution in both personal and professional settings. Leveraging advancements in natural language processing (NLP), machine learning, and contextual understanding, this voice assistant goes beyond simple command-following to offer a truly interactive and adaptive user experience. NGVA can understand complex user intent, handle multi-step tasks, and learn from user behavior to deliver personalized assistance. The system is highly flexible, with features that allow it to be integrated into a wide range of devices and applications, making it suitable for smart homes, businesses, and on-the-go environments.

2.4 FEASABILITY STUDY

The feasibility study for developing a Next Generation Voice Assistant for enhanced user interaction and task execution aims to assess the technical, operational, and economic viability of the project. This advanced voice assistant will leverage natural language processing, machine learning, and AI-driven contextual understanding to provide more seamless and intuitive interactions. Technically, the project is feasible given the advancements in AI models and cloud computing, which allow for robust and scalable voice processing capabilities. Operationally, the implementation can streamline various tasks by integrating with existing applications, reducing manual input, and improving efficiency for users across different sectors. Economically, the potential return on investment is strong, as demand for intelligent virtual assistants is rising in both consumer and enterprise markets. The study concludes that, with adequate resources and development, the voice assistant will deliver substantial value and meet user expectations for functionality, accuracy, and personalization.

2.5 SDLC MODEL

The SDLC process for developing a Next-Generation Voice Assistant can be organized into several key phases to enhance user interaction and task execution. Each phase will ensure that the voice assistant is well-designed, user-friendly, reliable, and capable of performing complex tasks. Here is a breakdown of the SDLC process, along with a diagram:

1. Requirement Analysis

- **Objective**: Define the functional and non-functional requirements for the voice assistant.
- **Tasks**:
  - Collect requirements from stakeholders.
  - Identify user needs for interaction and task management.
  - Define integration needs with third-party APIs or devices.
- **Output**: A clear specification document outlining user stories, personas, and core features.

2. System Design

- **Objective**: Architect the voice assistant, defining its technical stack, interfaces, and data flow.
- **Tasks**:
  - Design voice processing and natural language understanding (NLU) modules.
  - Plan database and backend architecture.
  - Design user interaction flows.
- **Output**: System design specifications, UI/UX prototypes, and a database schema.

3. Implementation

- **Objective**: Develop the voice assistant according to the design specifications.
- **Tasks**:
    - Code the core modules: voice recognition, NLU, and response generation.
    - Develop integration with other platforms (e.g., IoT devices, calendars).
    - Implement the user interface if there is a visual component.
- **Output**: Initial working version of the voice assistant.

4. Testing

- **Objective**: Validate that the voice assistant meets functional, performance, and security requirements.
- **Tasks**:
    - Conduct unit, integration, and system tests on voice modules.
    - Test usability with target users to ensure smooth interaction.
    - Perform stress testing and security assessments.
- **Output**: Testing reports, bug logs, and a refined version ready for deployment.

5. Deployment

- **Objective**: Release the voice assistant to the production environment.
- **Tasks**:
    - Deploy the application on a cloud platform or compatible devices.
    - Configure performance monitoring and logging.
- **Output**: A fully operational voice assistant available to end-users.

6. Maintenance and Updates

- **Objective**: Continuously improve and maintain the voice assistant.
- **Tasks**:
    - Gather user feedback for improvements.
    - Update with new features or integrations.
    - Monitor and fix any bugs or security issues.
- **Output**: Updated versions of the voice assistant, ensuring ongoing relevance and usability.

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATIONS

3.1 SOFTWARE REQUIREMENTS:

Speech Recognition: A reliable speech recognition software library or API is required to convert spoken language into text. Popular options include Google Cloud Speech-to-Text, Microsoft Azure Speech to Text, or CMU Sphinx.

Text-to-Speech Synthesis: To convert text-based responses into natural-sounding speech, a text-to-speech synthesis software library or API is needed. Examples include Google Text-to-Speech, Microsoft Azure Speech Service, or Amazon Polly.

APIs and Web Services: Integration with third-party services such as weather forecasts, news updates, music streaming platforms, and smart home devices requires APIs or web services provided by the respective service providers.

Development Tools: Integrated Development Environments (IDEs) like PyCharm, Visual Studio Code, or Eclipse can be used for coding and debugging. Version control systems like Git, project management tools, and collaboration platforms can also be beneficial for team-based development.

Development Tools: Integrated Development Environments (IDEs) like PyCharm, Visual Studio Code, or Eclipse can be used for coding and debugging. Version control systems like Git, project management tools, and collaboration platforms can also be beneficial for team-based development.

3.2 HARDWARE REQUIREMENTS

Microphone: A high-quality microphone is essential for accurately capturing the user's voice commands. USB or Bluetooth microphones are commonly used, and their selection should consider factors such as audio clarity, noise cancellation, and compatibility with the chosen development platform.

Speaker or Headphones: A speaker or headphones are necessary for the voice assistant to provide audio responses to the user. They should have good sound quality and be compatible with the chosen text-to-speech synthesis software.

Computing Device: A computing device is needed to develop and deploy the voice assistant application. This can be a desktop computer, laptop, or a single-board computer like Raspberry Pi, depending on the project's requirements.

3.3 COMMUNICATION INTERFACES

To design communication interfaces for a next-generation voice assistant focused on enhanced user interaction and task execution, it is essential to cover both the user interface (UI) and the underlying system architecture. Here’s a breakdown of the communication interfaces:

1. Voice Recognition Interface (VRI)

- **Purpose**: Captures and interprets user voice commands with high accuracy and natural language processing (NLP) capabilities.
- **Components**:
    - **Speech-to-Text (STT)**: Transforms voice input into text.
    - **Wake Word Detection**: Listens for a specific phrase to activate the assistant.
    - **Intent Recognition**: Uses NLP to interpret user intent and context.
- **Technologies**: Advanced NLP, ASR (Automatic Speech Recognition), and contextual AI.

2. Dialog Management Interface (DMI)

- **Purpose**: Manages ongoing interactions to maintain a natural conversation flow.
- **Components**:
    - **Context Tracking**: Keeps track of previous commands, allowing for multi-turn conversations.
    - **Dialog State Management**: Understands and remembers the current state of the conversation.

    - **Response Generation**: Forms responses that are relevant, concise, and context-aware.
- **Technologies**: Machine learning, reinforcement learning for adaptive responses.

3. Natural Language Understanding Interface (NLUI)

- **Purpose**: Deciphers complex language inputs to interpret the user’s exact needs.
- **Components**:
    - **Entity Extraction**: Identifies key entities (e.g., names, dates) from user input.
    - **Intent Matching**: Matches user input to predefined or dynamically generated intents.
    - **Sentiment Analysis**: Analyzes user emotions to adjust responses appropriately.
- **Technologies**: NLP frameworks, sentiment analysis models, and machine learning.

4. Text-to-Speech (TTS) Interface

- **Purpose**: Provides high-quality, human-like audio responses.
- **Components**:
    - **Voice Modulation**: Adjusts pitch, tone, and inflection for a natural sound.
    - **Language Localization**: Offers responses in the user’s preferred language or dialect.
    - **Emotion-Sensitive TTS**: Varies tone based on user sentiment (e.g., empathetic responses).
- **Technologies**: Neural TTS, voice synthesis models, and emotion-driven voice modulation.

5. Multimodal Interaction Interface

- **Purpose**: Integrates voice with other modes (text, visuals) to enhance interaction.
- **Components**:
    - **Text Interface**: Offers a text chat option for environments where speaking isn’t suitable.
    - **Visual Interface**: Uses screens to display information, instructions, or visual feedback.

- **Gesture Recognition (optional)**: Recognizes hand gestures or facial expressions as inputs.

- **Technologies**: UI/UX design frameworks, gesture recognition technology.

6. Task Execution Interface (TEI)

- **Purpose**: Handles execution of complex tasks by interacting with external systems and APIs.
- **Components**:
  - **Skill Management**: Manages multiple tasks or “skills” for different functionalities (e.g., calendar, smart home).
  - **API Gateway**: Connects to third-party services and applications (e.g., weather, email).
  - **Contextual Task Adaptation**: Adjusts tasks based on the user's profile, preferences, and context.
- **Technologies**: API integration, skill-based architecture, and context-aware processing.

7. Learning and Adaptation Interface

- **Purpose**: Enables the assistant to learn from user interactions and improve over time.
- **Components**:
  - **User Profiling**: Creates and maintains a user profile for personalization.
  - **Feedback Loop**: Gathers user feedback and refines responses.
  - **Continuous Learning**: Learns user preferences, language patterns, and intent refinements.
- **Technologies**: Machine learning, personalization algorithms, and user feedback analytics.

8. Privacy and Security Interface

- **Purpose**: Ensures the user’s data privacy and secure interactions.
- **Components**:
  - **Data Encryption**: Protects data in transit and at rest.

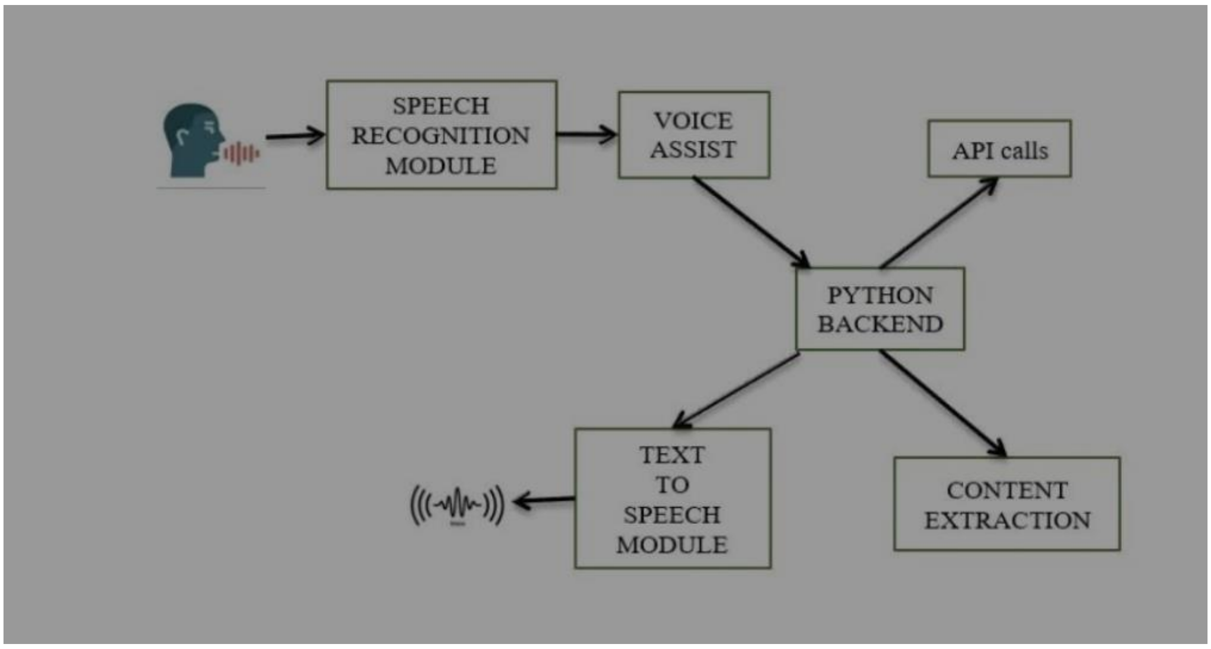
  - **Privacy Controls**: Allows users to manage privacy settings, including history deletion.
  - **Access Control**: Secures access to sensitive information and task execution.
- **Technologies**: Encryption standards, access control protocols, and privacy management tools.

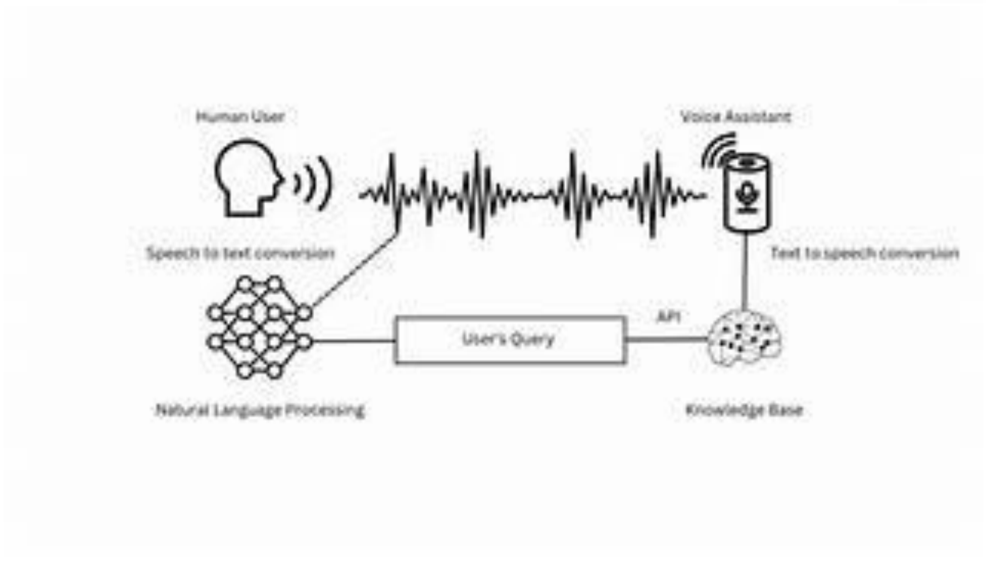
CHAPTER 4

4.1 SYSTEM DESIGN

Voice Assistant
Voice Assistant
Voice Assistant
Natut Lenguage Processtion
Natue Lenguage Process
Language Processing
Vated Langech Undergnting Protessing
NLP
NLP
NP
Speech Rercestiong
Task
Personalzed Recomgnemtions
AI
Personized Recomrnanations
Task
Next Generation Voice Assistaint
ALI
Task
Task
Task

4.2 SYSTEM ARCHITECTURE




Human User
Voice Assistant
Speech to text conversion
Text to speech conversion
API
User's Query
Natural Language Processing
Knowledge Base

4.3 MODULE DESCRIPTION

Module 1: Speech Recognition and Processing

- This module enables the system to accurately capture, process, and convert spoken input into text, forming the foundation for interaction. Leveraging advanced Natural Language Processing (NLP) and Machine Learning (ML) algorithms, this component ensures that the assistant recognizes and understands various accents, dialects, and language structures for robust and inclusive voice recognition.

Module 2: Natural Language Understanding (NLU)

- The NLU module is designed to analyze and interpret the user's spoken input, extracting key intent and context. It translates the spoken requests into structured commands that other modules can execute. NLU enables the assistant to handle complex queries, follow-up questions, and conversations involving multiple intents or tasks.

Module 3: Dialogue Management

- Dialogue Management is responsible for guiding the interaction flow, making the conversation smooth and natural. This module utilizes context tracking, memory management, and sentiment analysis to adjust its responses based on user input, tone, and previous interactions. It ensures continuity in the conversation, allowing for multi-turn interactions and proactive suggestions.

Module 4: Task Execution and Integration

- This module connects the assistant to various applications, devices, and external services, enabling it to execute tasks such as scheduling appointments, controlling smart home devices, or retrieving information from online databases. Task Execution leverages APIs and integrations to perform actions on behalf of the user, improving their productivity and convenience.

Module 5: Response Generation

- The Response Generation module composes human-like and contextually appropriate responses for both voice and text output. It considers factors such as context, user preferences, and response tone to provide answers or feedback that feel engaging and tailored to the user’s needs.

Module 6: Personalization and Adaptive Learning

- This module allows the assistant to learn and adapt over time based on user preferences, frequently requested tasks, and interaction patterns. By personalizing the experience, the assistant becomes more efficient and relevant, continuously refining its interactions to meet the unique needs of each user.

Module 7: Security and Privacy

- Security and Privacy are critical components ensuring that user data is handled responsibly. This module encompasses encryption, data anonymization, and secure storage practices, as well as options for user control over personal data. It also manages permissions for accessing and executing sensitive tasks or accessing private information.

Module 8: User Interface (UI) & User Experience (UX) Design

- This module focuses on the design of the visual and interactive elements for the assistant, creating a seamless and intuitive user experience. It includes the design of voice interfaces, graphical interfaces, and touch interactions, catering to both visual and non-visual interaction modes.

Module 9: Analytics and Performance Monitoring

- The Analytics module collects and analyzes data on usage patterns, interaction quality, and system performance. This feedback loop helps developers refine the assistant, improving accuracy, response times, and user satisfaction based on real-world usage data.

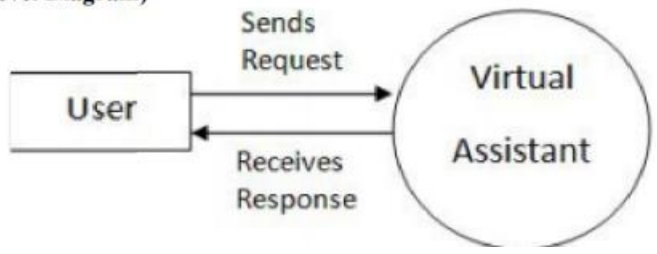
Module 10: Testing and Optimization

- Testing and Optimization are essential to ensure reliability and responsiveness. This module includes various test stages—such as unit testing, integration testing, and user acceptance testing—and utilizes optimization techniques to enhance the assistant's performance and reduce latency.

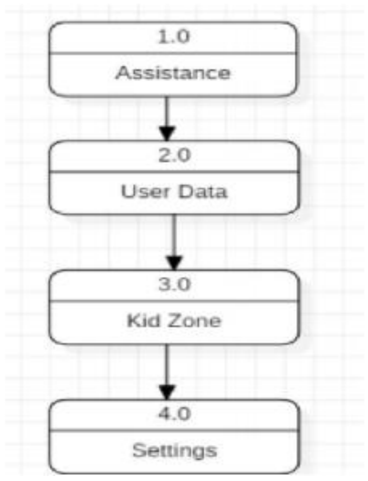
Each module plays a crucial role in delivering a high-quality, intelligent, and user-friendly voice assistant capable of engaging in natural conversations and efficiently executing tasks.

4.4 DFD DESIGN

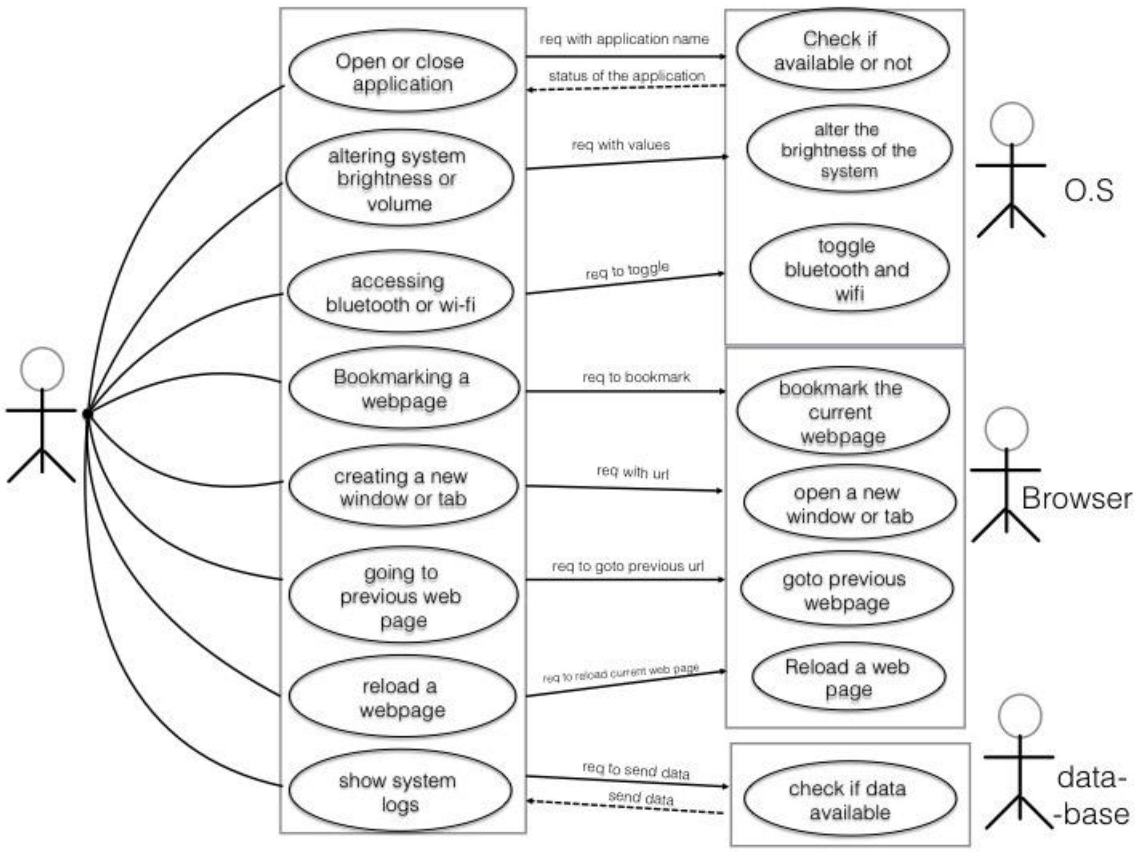
A. DFD Level 0 (Context Level Diagram)



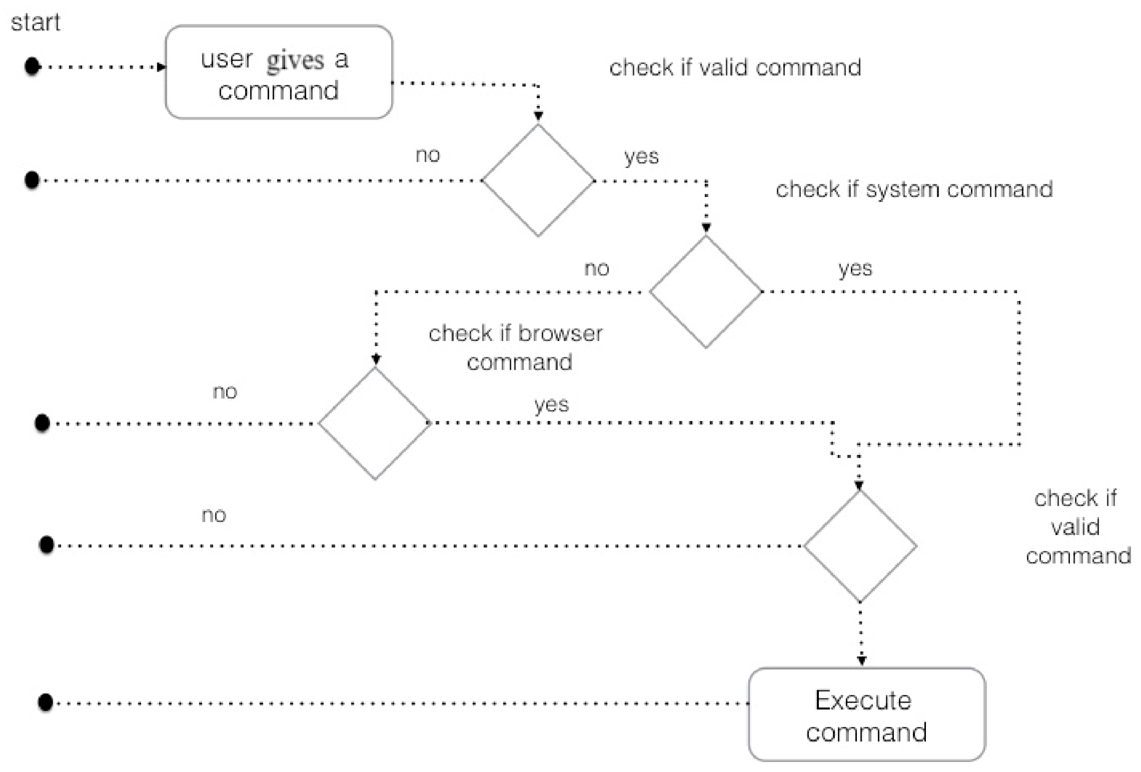
B. DFD Level 1



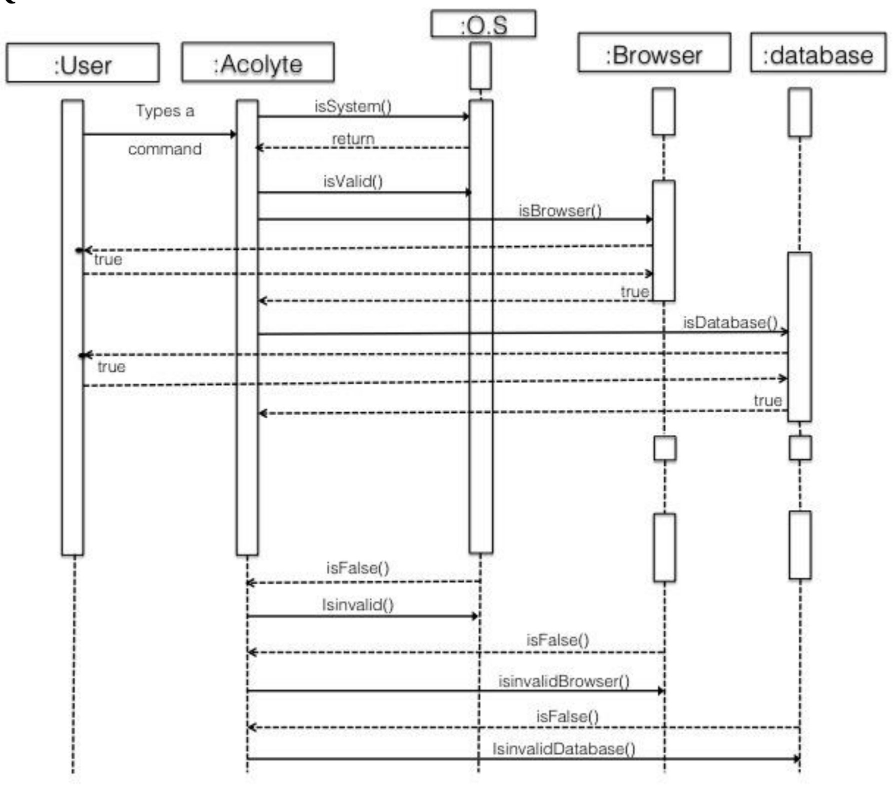
4.5 USE CASE DIAGRAM


Open or close application
altering system brightness or volume
accessing bluetooth or wi-fi
Bookmarking a webpage
creating a new window or tab
going to previous web page
reload a webpage
show system logs
req with application name
status of the application
req with values
req to toggle
req to bookmark
req with url
req to goto previous url
req to reload current web page
req to send data
send data
Check if available or not
alter the brightness of the system
toggle bluetooth and wifi
bookmark the current webpage
open a new window or tab
goto previous webpage
Reload a web page
check if data available
O.S
Browser
data-
-base

4.6 ACTIVITY DIAGRAM



4.7 SEQUENCE DIAGRAM



4.8 CLASS DIAGRAM

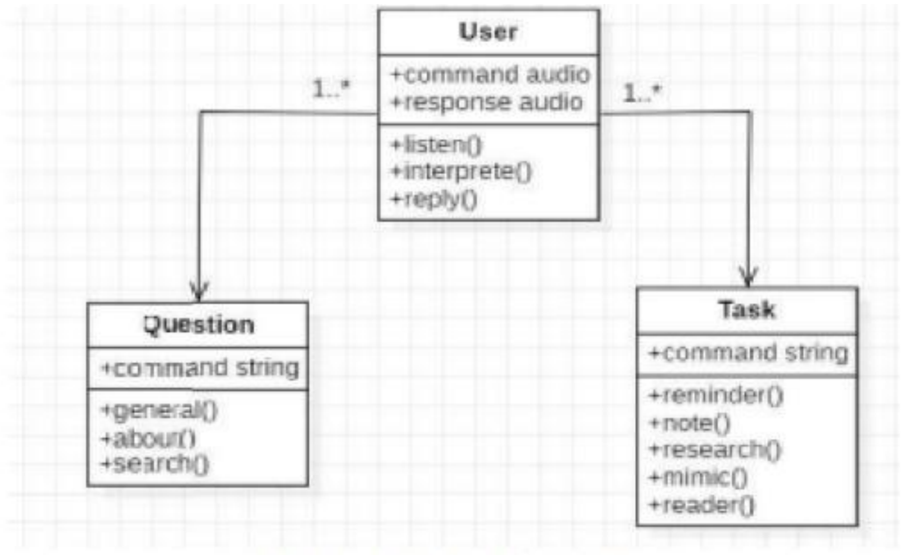


Figure 4: Class Diagram

5. SYSTEM IMPLEMENTATION

5.1. RUNNING APPLICATION:-

Running the Bruce desktop voice assistant involves initializing the application, ensuring the environment is correctly set up, and executing the program for user interaction. Below is a detailed step-by-step guide to running the application:

1. Prerequisites:

To successfully run Bruce, ensure the following prerequisites are met:

- Python Installation: Python 3.8 or later should be installed on the system.
- Dependencies: Install required Python libraries using the requirements.txt file with the following command
- Hardware Requirements: The system must have a functioning microphone for voice input and speakers for audio output.

2. Application Startup:

- Navigate to the project directory where the application files are stored.
- Execute the main.py file using the terminal or command prompt
- The GUI will launch, displaying a splash screen or welcome interface.

3. Version Selection:

- Upon startup, the user is prompted to choose between the Basic and Advanced Versions of Bruce.
    - Basic Version: Includes fundamental features like file management, reminders, and media control.
    - Advanced Version: Provides enhanced capabilities such as NLP for complex queries, Chat GPT integration, and API-based services.
- The selection can be made via GUI buttons or voice commands.

4. Input Modes:

- The application supports dual input modes:
    - Voice Input: Users can issue commands via microphone, processed through speech recognition modules.
    - Text Input: Users can type commands into the input field on the GUI.

5. Functional Testing:

- Validate that all features are working as expected:
    - Test file management commands like opening, moving, or deleting files.
    - Set reminders and verify notifications.
    - Access weather updates, play music, or use the quick dictionary feature.
- In the Advanced Version, test NLP-based responses, API integrations (e.g., Google Calendar), and Chat GPT-powered interactions.

6. Performance Optimization:

- Evaluate the response time for commands in both input modes.
- Monitor system resource usage to ensure the application runs smoothly without overloading the system.

By following these steps, users can efficiently launch and interact with the Bruce application, leveraging its features to automate tasks and enhance productivity.

5.2. CONFIGURING DATABASE:

The Bruce application uses an SQLite database (bruce.db) for managing user data and tasks. To configure the database:

1. **Database Initialization:** On the first run, the application will check for bruce.db. If it doesn't exist, it will automatically create the database and initialize tables for tasks, reminders, user preferences, and notes.
2. **Manual Configuration:** Advanced users can use database management tools like DB Browser for SQLite to manually configure or inspect the database.
3. **Syncing Data:** The database syncs tasks, notes, and reminders between the Basic and Advanced Versions, ensuring a unified experience across modes.
4. **Custom Queries:** Developers can write custom SQL queries to manage and retrieve data for specific use cases.

5.3. CODING

The coding phase for the Bruce desktop voice assistant involves implementing the planned functionalities into a modular and scalable codebase. The application is developed in Python, utilizing several libraries and frameworks to achieve a robust system. Below are the detailed aspects of the coding process:

1. Modular Structure

- The application is divided into distinct modules to ensure clarity and ease of maintenance:
  - **main.py**: Handles the overall application flow, including GUI initialization and module integration.
  - **basic.py**: Implements the Basic Version functionalities such as file management, media control, reminders, and quick commands.
  - **advanced.py**: Integrates Advanced Version features like natural language processing (NLP), API interactions, and ChatGPT-powered responses.

2. Graphical User Interface (GUI)

- Developed using **PyQt**, the GUI provides a professional and user-friendly interface.
  - Features include buttons, input fields, and a toggle to switch between Basic and Advanced Versions.
  - Round icons are used to differentiate user inputs and assistant responses for better visual appeal.

3. Speech and Text Processing

- **Speech Recognition:**
  - The **SpeechRecognition** library processes user voice inputs, converting them into text commands.
- **Text-to-Speech Conversion:**
  - The **pyttsx3** library is used to synthesize natural and customizable voice outputs for responses.

4. Database Integration

- SQLite is used to store and manage data like tasks, reminders, and user notes in bruce.db.
- SQL queries are embedded within the code for seamless data interaction.

5. API Integration

- External APIs are integrated into the Advanced Version to enhance functionality:
  - **Google Calendar API:** Manages events and reminders.
  - **Open WeatherMap API:** Provides real-time weather updates.
  - **Spotify API:** Enables music recommendations and playback.

6. Error Handling and Debugging

- Error handling mechanisms are implemented to provide meaningful feedback to users during failures.
- Logging is integrated to track system operations and identify bugs during testing.

7. Testing and Optimization

- Each module is tested independently to ensure functionality.
- The entire application undergoes integration testing to verify smooth interaction between modules.

The coding process ensures that Bruce is robust, efficient, and capable of delivering a seamless user experience across both Basic and Advanced Versions.

6. SYSTEM TESTING:

System testing is the comprehensive evaluation of the Bruce desktop voice assistant as a fully integrated system to ensure it meets functional, performance, and user requirements. This phase validates that the entire application, including all modules, databases, APIs, and user interfaces, operates seamlessly and reliably under real-world conditions.

The process includes:

- **Verification of Features:** Ensuring that functionalities such as file management, reminders, web searches, and weather updates work correctly across both Basic and Advanced Versions.
- **Performance Testing:** Evaluating response times and resource utilization to confirm the application runs efficiently without lags or crashes.
- **Reliability Testing:** Running repetitive commands and scenarios to ensure the system remains stable over time.
- **Compatibility Testing:** Testing Bruce across different hardware configurations and operating systems to verify it works as intended in various environments.
- **User Interaction Validation:** Confirming that both text and voice input modes provide accurate responses, and the GUI remains intuitive and responsive.

System testing ensures that Bruce is a robust and user-friendly application, ready for deployment and practical use.

6.1. TESTING INTRODUCTION

Testing is a critical phase in the software development lifecycle that ensures the quality, functionality, and reliability of the Bruce desktop voice assistant. The primary goal of testing is to identify and rectify any issues in the system, ensuring it meets user expectations and requirements. Testing for Bruce is conducted systematically, focusing on various aspects of the application to verify both individual components and the system as a whole.

Purpose of Testing

The testing process ensures the following:

- **Functionality Validation:** All features perform as intended.
- **Bug Detection:** Identify and resolve errors or defects in the code.
- **Performance Assurance:** Validate the efficiency of the system under different loads.
- **User Experience Evaluation:** Ensure the application is intuitive, responsive, and accessible.

Scope of Testing

The testing process for Bruce encompasses:

1. **Unit Testing:** Evaluating individual components like the Basic Version, Advanced Version, speech modules, and database interactions.
2. **Integration Testing:** Ensuring smooth interaction between modules such as GUI, voice recognition, and API integrations.
3. **System Testing:** Validating the entire application under various scenarios and environments.
4. **Regression Testing:** Ensuring that new updates or features do not introduce errors in previously functioning modules.
5. **User Acceptance Testing (UAT):** Simulating end-user scenarios to ensure Bruce meets user requirements and expectations.

Testing Methodology

The testing methodology combines both manual and automated testing approaches:

- **Manual Testing:** Focuses on exploratory testing of features, user interface interactions, and error handling scenarios.
- **Automated Testing:** Utilizes scripts and tools for repetitive testing tasks, such as voice recognition accuracy and database interactions.

Significance of Testing

Rigorous testing ensures Bruce is a reliable and robust voice assistant capable of automating tasks efficiently. It minimizes risks, enhances user satisfaction, and builds confidence in the application's readiness for deployment.

6.2. UNIT TESTING

Unit testing is the process of validating individual components or modules of the Bruce desktop voice assistant in isolation to ensure they function correctly. It focuses on testing the smallest parts of the application, such as functions, methods, or classes, without considering their interaction with other components. This ensures that each unit performs as expected and contributes to the overall stability of the system.

Purpose of Unit Testing

- To identify and fix bugs at an early stage in the development cycle.

- To validate the correctness of individual components before integrating them into the system.
- To simplify debugging by isolating errors in specific units.
- To improve code quality and maintainability by ensuring each unit adheres to its design specifications.

Scope of Unit Testing in Bruce

The unit testing process covers the following core modules of the Bruce voice assistant:

1. **Speech Recognition Module:**
    - Tests the ability to process and convert voice input into text accurately.
    - Validates command recognition and error handling for unclear or incomplete inputs.
2. **Text-to-Speech (TTS) Module:**
    - Ensures correct voice synthesis for responses in both Basic and Advanced Versions.
    - Verifies multi-language support and customizable voice settings.
3. **Database Interaction Module:**
    - Validates data storage and retrieval operations in bruce.db for tasks, reminders, and notes.
    - Ensures SQL queries execute correctly and handle edge cases such as empty inputs or invalid data.
4. **Basic Features Module:**
    - Tests functions like file management, media control, and system commands.
    - Ensures accurate execution of commands such as opening files or setting reminders.
5. **Advanced Features Module:**
    - Validates API integrations (e.g., Google Calendar, OpenWeatherMap) for accurate data retrieval.
    - Tests ChatGPT responses for contextual relevance and correctness.
6. **Graphical User Interface (GUI):**
    - Verifies button functionality, input field handling, and responsiveness of the interface.

Tools Used for Unit Testing:

- **Unit test (Python Standard Library):** Used for creating and executing test cases.
- **Py test:** Preferred for its simplicity and support for advanced testing features like parameterization.
- **Mocking Libraries:** Simulate external dependencies such as API calls during testing.

Example Unit Test Case:

Test Case for Speech Recognition Module:

- **Objective:** Verify that the module converts voice input into the correct text command.
- **Test Input:** Audio file with the command "Open Notepad."
- **Expected Output:** Text string "Open Notepad."
- **Result:** Passed (The output matches the expected text).

Significance of Unit Testing:

By identifying and resolving issues early, unit testing ensures that each component of Bruce is reliable and functions as intended. It reduces the risk of integration errors and contributes to the development of a robust and maintainable application.

6.3. WHITE BOX TESTING

White box testing, also known as structural or glass-box testing, is a testing technique that focuses on the internal structure, design, and implementation of the application. It involves verifying the logical flow, code correctness, and functionality of the individual components of the Bruce desktop voice assistant. This method ensures that the internal workings of the application align with its intended behavior.

Purpose of White Box Testing

- To validate the internal logic and decision-making processes within the code.
- To identify and fix issues such as improper control flow, security vulnerabilities, or un optimized code.
- To ensure code coverage, including all branches, loops, and paths.

Scope of White Box Testing in Bruce:

1. **Control Flow Testing:**
   - Verifies that all possible execution paths within the code are tested.
   - Ensures that loops, conditionals, and branches function as expected.
2. **Data Flow Testing:**
   - Tracks variables through the application to ensure correct data handling and usage.
   - Identifies issues like unused variables or incorrect data assignments.
3. **Statement and Branch Testing:**
   - Ensures all lines of code are executed at least once during testing.
   - Validates logical conditions in "if-else" statements to test all branches.
4. **Security Testing:**
   - Verifies that sensitive operations, such as database queries or file handling, are secure and protected against common vulnerabilities like SQL injection or buffer overflows.
5. **Performance Optimization:**
   - Analyzes the efficiency of algorithms used for speech recognition, text-to-speech conversion, and other core functionalities.

Tools Used for White Box Testing:

- **Coverage.py:** Measures code coverage during testing.
- **Pylint and Flake8:** Ensures code quality and adherence to best practices.
- **Debugging Tools:** Built-in Python debuggers to analyze execution flow and identify runtime issues.

Example Test Case

Test Case for the Decision-Making Logic in the Basic Version:

- **Objective:** Verify that the correct action is executed for each recognized command.
- **Test Input:** Command string "Open File Explorer."
- **Execution Flow:**
  1. Recognize the command as a file operation.
  2. Match it to the "open_file_explorer" function.
  3. Execute the function to open File Explorer.

- **Expected Output:** File Explorer is launched.
- **Result:** Passed (The function executed the desired operation).

Significance of White Box Testing

White box testing ensures that the internal mechanics of the Bruce desktop voice assistant are efficient, secure, and error-free. It helps developers gain insights into potential flaws within the codebase and optimize the application's performance and reliability.

6.4. BLACK BOX TESTING

Black box testing is a testing technique that evaluates the functionality of the Bruce desktop voice assistant without examining its internal code or logic. The tester focuses on input and output scenarios, ensuring the application behaves as expected under various conditions. This method treats the system as a "black box," where the internal workings are hidden, and only the external behavior is tested.

Purpose of Black Box Testing

- To validate the system's functionality against defined requirements.
- To identify issues related to input/output handling, user interactions, and feature integration.
- To ensure the application performs accurately under normal, boundary, and unexpected conditions.

Scope of Black Box Testing in Bruce

1. **Functional Testing:**
   - Verifies that core functionalities like file management, reminders, and web searches operate correctly.
   - Tests both Basic and Advanced Versions for expected outputs based on various commands.
2. **Boundary Testing:**
   - Tests application responses at the limits of valid input ranges.
   - For instance, testing Bruce's ability to process long or complex voice commands.

3. **Error Handling:**
    - Ensures the system responds appropriately to invalid inputs, such as unrecognized voice commands or corrupted files.
4. **Compatibility Testing:**
    - Checks the behavior of the application across different operating systems and devices.
    - Validates GUI responsiveness and voice command accuracy in various environments.
5. **Usability Testing:**
    - Evaluates the user experience, ensuring the interface is intuitive and easy to navigate.

Example Test Cases

Test Case 1: Web Search Functionality

- **Input:** Voice command "Search for the latest weather in New York."
- **Expected Output:** The application retrieves and displays the weather forecast for New York.
- **Result:** Passed.

Test Case 2: File Management

- **Input:** Command "Delete file 'example.txt'."
- **Expected Output:** The application deletes the specified file and confirms the action.
- **Result:** Passed.

Test Case 3: Invalid Input Handling

- **Input:** Unintelligible voice command or garbled text input.
- **Expected Output:** The application prompts the user to repeat the command or provides an error message.
- **Result:** Passed.

Tools Used for Black Box Testing

- Manual testing for functional and usability validation.
- Automated testing tools like Selenium for GUI and input-output verification.

Significance of Black Box Testing

Black box testing ensures that Bruce meets user requirements and functions reliably without requiring the tester to understand the underlying code. It validates the system from an end-user perspective, improving its usability and ensuring robust performance under real-world conditions.

6.5. INTEGRATION TESTING:

Integration testing is the process of testing the interaction and communication between different modules or components of the Bruce desktop voice assistant to ensure they work together seamlessly. This testing phase focuses on identifying issues that may arise when individual units are combined and ensuring the application's functionality as a cohesive system.

Purpose of Integration Testing

- To verify that the interfaces between modules are working as intended.
- To identify and resolve data flow issues, mismatched inputs/outputs, or communication errors.
- To ensure the system functions as a unified application after integrating all its components.

Scope of Integration Testing in Bruce

1. **Module Interaction Testing:**
    - Basic and Advanced Features: Ensure the toggle between Basic and Advanced Versions works correctly and transfers commands seamlessly.
    - Speech Recognition and Command Execution: Validate that the recognized voice input is passed to the appropriate module (e.g., file management or web search).
    - Text-to-Speech Integration: Confirm that responses generated by command modules are accurately converted into speech.
2. **Database Integration Testing:**
    - Ensure that bruce.db interacts correctly with task management, reminders, and user profiles.
    - Validate CRUD (Create, Read, Update, Delete) operations and error handling.

3. **API Integration Testing:**
   - Verify data retrieval from external APIs (e.g., Google Calendar, weather updates) and confirm accurate display of the retrieved data.
   - Test failure scenarios, such as network unavailability, to ensure proper error messages are displayed.
4. **GUI Integration Testing:**
   - Confirm that the graphical user interface (GUI) interacts correctly with the back-end modules.
   - Test real-time updates for actions like task creation or media control.
5. **System-Level Testing:**
   - Test scenarios involving multiple modules simultaneously, such as a voice command to set a reminder while playing music.

Approaches to Integration Testing

1. **Top-Down Integration:**
   - Begin testing from the main control module (e.g., main.py) and progressively test subordinate modules.
   - Useful for testing high-level workflows like the user toggling between versions.
2. **Bottom-Up Integration:**
   - Start testing from the lowest-level modules (e.g., speech recognition) and build upward to test their integration with higher-level components.
3. **Hybrid Integration:**
   - Combines both top-down and bottom-up approaches to test modules more flexibly.

Example Test Cases

Test Case 1: Speech-to-Command Execution

- **Input: Voice command "Set a reminder for 10 AM tomorrow."**
- **Expected Output:**
  1. Speech-to-text module converts the voice input into the command.
  2. The task management module adds the reminder to bruce.db.
  3. A confirmation message is displayed and spoken by the TTS module.
- **Result: Passed.**

Test Case 2: API Integration

- **Input: Voice command "Show today's weather."**
- **Expected Output:** The weather data is retrieved from the weather API and displayed in the GUI with a voice response.
- **Result: Passed.**

Tools Used for Integration Testing

- **Postman:** For testing API interactions.
- **pytest:** For automating integration test cases.
- **Selenium:** For GUI and interaction testing.

Significance of Integration Testing:

Integration testing ensures that the individual modules of Bruce work cohesively and reliably as a unified system. It helps identify potential issues in module interactions, ensuring that the application delivers a smooth and error-free user experience.

6.6. SYSTEM TESTING:

System testing is a critical phase in the software development lifecycle that involves testing the Bruce desktop voice assistant as a complete and integrated application. This ensures that all components, modules, and features function together seamlessly in the real-world environment and meet the specified requirements.

Purpose of System Testing

- To validate that the entire system works as intended, with all components integrated.
- To detect and resolve defects that may not be apparent during unit or integration testing.
- To ensure the application meets functional and non-functional requirements, including usability, performance, and reliability.

Scope of System Testing in Bruce

1. **Functional Testing:**
   - Verifies that Bruce's core functionalities, such as file management, reminders, and web searches, are working as expected.
   - Confirms proper operation of the toggle feature between Basic and Advanced Versions.

2. **Non-Functional Testing:**
    - **Performance Testing:** Evaluates the application’s responsiveness under varying workloads, such as handling multiple voice commands simultaneously.
    - **Security Testing:** Ensures the application securely handles sensitive operations, like accessing files or sending emails.
    - **Usability Testing:** Assesses the intuitiveness and accessibility of the GUI for users of all age groups.
3. **End-to-End Testing:**
    - Tests complete workflows, such as setting a reminder, syncing it to the database, and retrieving it later for display.
4. **Compatibility Testing:**
    - Ensures Bruce performs consistently across different operating systems, screen resolutions, and hardware configurations.
5. **Regression Testing:**
    - Verifies that recent code changes or updates do not negatively affect existing functionality.

Example System Testing Scenarios:

Scenario 1: Voice Command Execution

- **Input: "Play music from my library."**
- **Process:** Speech recognition module captures the input, command is processed, and the media player module plays the requested music.
- **Expected Output:** Music is played, and a confirmation is provided via voice and GUI.
- **Result: Passed.**

Scenario 2: Error Handling

- **Input: Command to open a non-existent file.**
- **Expected Output:** Application displays an error message and suggests possible resolutions without crashing.
- **Result: Passed.**

Scenario 3: API Integration

- **Input: "Check my Google Calendar for today's schedule."**
- **Expected Output:** Application retrieves and displays the user's calendar events via API integration.
- **Result: Passed.**

Tools Used for System Testing

- **Selenium:** For automating GUI testing and interaction validation.
- **Apache JMeter:** For performance testing under heavy loads.
- **Manual Testing:** To validate user experience and complex workflows.

Significance of System Testing:

System testing ensures that the Bruce desktop voice assistant meets its functional and non-functional requirements as a complete system. It validates the application from the user's perspective and ensures that the software is ready for deployment, delivering a seamless, reliable, and user-friendly experience.

Code Implementation:-

```
import os
import time
import datetime
import pyttsx3
import speech_recognition as sr
import webbrowser
import smtplib
import requests
import psutil
from tkinter import Tk, simpledialog
import subprocess
from PIL import ImageGrab
import pyautogui
import json
import pygame
from  PyDictionary import PyDictionary
import sqlite3

# Initialize text-to-speech engine
engine = pyttsx3.init()
engine.setProperty("rate", 150)

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

def take_command():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        speak("Listening...")
        try:
            audio = recognizer.listen(source, timeout=5)
            return recognizer.recognize_google(audio).lower()
        except sr.UnknownValueError:
            speak("Sorry, I didn't catch that.")
            return None
        except sr.RequestError:
            speak("Error with the recognition service.")
            return None

# Media Control

# Initialize Pygame mixer
pygame.mixer.init()

# Global variables for song list and current song index
songs_folder = r"C:\Users\vutla\Music\re"  # Replace with your songs folder path
song_list = [f for f in os.listdir(songs_folder) if f.endswith(('.mp3', '.wav'))]
current_song_index = 0

def play_song(index):
    """Play the song at the given index."""
    global current_song_index
    if 0 <= index < len(song_list):
        current_song_index = index
        song_path = os.path.join(songs_folder, song_list[current_song_index])
        pygame.mixer.music.load(song_path)
        pygame.mixer.music.play()
        speak(f"Playing {song_list[current_song_index]}")
    else:
        speak("No songs available to play.")

def control_media(command):
    """Control media playback."""
    global current_song_index
    if "play song" in command:
        play_song(current_song_index)
    elif "pause" in command:
        pygame.mixer.music.pause()
        speak("Music paused.")
    elif "resume" in command:
        pygame.mixer.music.unpause()
        speak("Music resumed.")
    elif "next" in command:
        current_song_index = (current_song_index + 1) % len(song_list)
        play_song(current_song_index)
    elif "previous" in command:
        current_song_index = (current_song_index - 1) % len(song_list)
        play_song(current_song_index)
    elif "volume up" in command:
        volume = pygame.mixer.music.get_volume()
        pygame.mixer.music.set_volume(min(volume + 0.1, 1.0))
        speak("Volume increased.")
    elif "volume down" in command:
        volume = pygame.mixer.music.get_volume()
        pygame.mixer.music.set_volume(max(volume - 0.1, 0.0))
        speak("Volume decreased.")
    elif "stop" in command:
        pygame.mixer.music.stop()
        speak("Music stopped.")

# Reminders

def set_reminder():
    speak("What should I remind you about?")
    reminder = take_command()
    if reminder:
        conn = sqlite3.connect('bruce.db')
        cursor = conn.cursor()
        reminder_time = time.time() + 30 * 60  # 30 minutes from now
        cursor.execute('INSERT INTO reminders (text, time) VALUES (?, ?)', (reminder, reminder_time))
        conn.commit()
        conn.close()
        speak("Reminder set for 30 minutes.")

def check_reminders():
    conn = sqlite3.connect('bruce.db')
    cursor = conn.cursor()
    current_time = time.time()
    cursor.execute('SELECT id, text FROM reminders WHERE time <= ?', (current_time,))
    reminders_to_notify = cursor.fetchall()
    for reminder_id, reminder_text in reminders_to_notify:
        speak(f"Reminder: {reminder_text}")
        cursor.execute('DELETE FROM reminders WHERE id = ?', (reminder_id,))
    conn.commit()
    conn.close()

# Notepad Integration
def open_notepad():
    os.system("notepad")

def save_note():
    speak("What should I save in the note?")
    content = take_command()
    with open("note.txt", "a") as file:
        file.write(f"{datetime.datetime.now()} - {content}\n")
    speak("Note saved.")

# Task List Management

def add_task():
    speak("What task would you like to add?")
    task = take_command()
    if task:
        conn = sqlite3.connect('bruce.db')
        cursor = conn.cursor()
        cursor.execute('INSERT INTO tasks (description) VALUES (?)', (task,))
        conn.commit()
        conn.close()
        speak(f"Task added: {task}")

def show_tasks():
    conn = sqlite3.connect('bruce.db')
    cursor = conn.cursor()
    cursor.execute('SELECT id, description FROM tasks')
    tasks = cursor.fetchall()
    conn.close()
    if tasks:
        for task_id, task_desc in tasks:
            speak(f"Task {task_id}: {task_desc}")
    else:
        speak("No tasks found.")

def clear_tasks():
    conn = sqlite3.connect('bruce.db')
    cursor = conn.cursor()
    cursor.execute('DELETE FROM tasks')
    conn.commit()
    conn.close()
    speak("All tasks cleared.")

# Weather Updates
def get_weather():
    try:
        speak("Opening weather information for Hyderabad.")
        # Opens the weather page for Hyderabad in the default browser
        webbrowser.open("https://www.weather.com/weather/today/l/17.38,78.47")
    except Exception as e:
        speak(f"An error occurred while retrieving the weather. {str(e)}")

# Quick Dictionary
def get_meaning():
    speak("Which word would you like to look up?")
    word = take_command()
    try:
        dictionary = PyDictionary()
        meaning = dictionary.meaning(word)
        if meaning:
            for key, value in meaning.items():
                speak(f"{key}: {', '.join(value)}")
        else:
            speak("I couldn't find the meaning.")
    except ImportError:
        speak("Dictionary feature requires PyDictionary module. Please install it.")

# Quick Access Commands
quick_access = {
    "chrome": "C:/Program Files/Google/Chrome/Application/chrome.exe",
    "notepad": "notepad",
}

def open_quick_access():
    speak("Which application would you like to open?")
    app = take_command()
    if app in quick_access:
        os.startfile(quick_access[app])
        speak(f"Opening {app}.")
    else:
        speak("Application not found in quick access commands.")
```

```
# Customizable Shortcuts

def add_shortcut():
    speak("What is the shortcut name?")
    name = take_command()
    if name:
        speak("Please paste the path for this shortcut.")
        path = input("Enter the path: ")  # User pastes the path here
        conn = sqlite3.connect('bruce.db')
        cursor = conn.cursor()
          cursor.execute('INSERT INTO shortcuts (name, path) VALUES (?, ?)', (name, 
path))
        conn.commit()
        conn.close()
        speak("Shortcut added.")

def execute_shortcut():
    speak("Which shortcut would you like to execute?")
    name = take_command()
    if name:
        conn = sqlite3.connect('bruce.db')
        cursor = conn.cursor()
        cursor.execute('SELECT path FROM shortcuts WHERE name = ?', (name,))
        result = cursor.fetchone()
        conn.close()
        if result:
            os.startfile(result[0])
            speak(f"Executing shortcut {name}.")
        else:
            speak("Shortcut not found.")

# Initialize shortcuts from file
if os.path.exists("shortcuts.json"):
    with open("shortcuts.json", "r") as file:
        shortcuts = json.load(file)

# Features Implementation
def open_file():
    speak("What is the name of the file you'd like to open?")
    file_name = take_command()
    if file_name:
        file_path = rf"C:\Users\vutla\OneDrive\Documents\{file_name}.txt"
        if os.path.exists(file_path):
            os.startfile(file_path)
            speak(f"Opening {file_name}")
        else:
            speak("Sorry, I couldn't find that file.")

def take_screenshot():
    try:
        screenshot = ImageGrab.grab()
        save_path = r"C:\Users\vutla\OneDrive\Pictures\screenshot.png"
        screenshot.save(save_path)
        speak(f"Screenshot taken and saved at {save_path}.")
    except Exception as e:
        speak(f"Failed to take a screenshot. Error: {str(e)}")

def show_date_time():
    now = datetime.datetime.now()
        speak(f"The current date is {now.strftime('%Y-%m-%d')} and the time is
{now.strftime('%H:%M:%S')}.")

def web_search():
    speak("What should I search for?")
    query = take_command()
    if query:
        webbrowser.open(f"https://www.google.com/search?q={query}")
        speak(f"Searching for {query}")
```

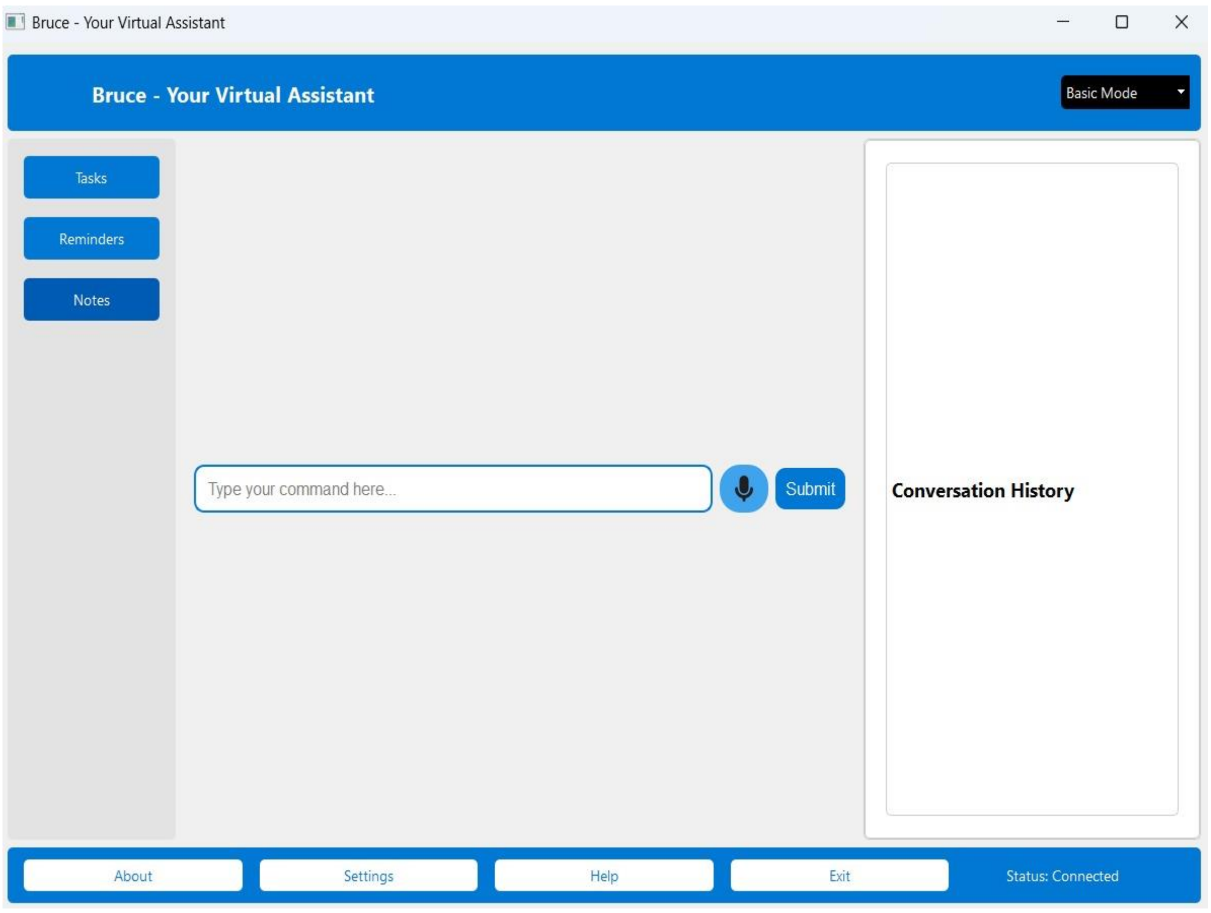
```
def send_email():
    speak("To whom should I send the email?")
    recipient = simpledialog.askstring("Email", "Enter recipient email:")
    speak("What should I say?")
    content = take_command()
    if recipient and content:
        try:
            server = smtplib.SMTP("smtp.gmail.com", 587)
            server.starttls()
            server.login("vaishnavtechnologiesassistance@gmail.com",
"mzzhrokesnjyzgvc")
            server.sendmail("vaishnavtechnologiesassistance@gmail.com", recipient,
content)
            server.quit()
            speak("Email sent successfully.")
        except Exception as e:
            speak(f"Failed to send email. {str(e)}")

def system_info():
    battery = psutil.sensors_battery()
    speak(f"Your system has {battery.percent} percent battery remaining.")

def shut_down_pc():
    speak("Are you sure you want to shut down the PC?")
    confirmation = take_command()
    if "yes" in confirmation:
        os.system("shutdown /s /t 1")
        speak("Shutting down the PC.")

# Main Menu
def basic_mode():
    while True:
        speak("How can I assist you?")
```

```
command = take_command()
if command:
   if "open file" in command:
      open_file()
   elif "screenshot" in command:
      take_screenshot()
   elif "time" in command or "date" in command:
      show_date_time()
   elif "search" in command:
      web_search()
   elif "email" in command:
      send_email()
   elif "battery" in command or "system info" in command:
      system_info()
   elif "shut down" in command:
      shut_down_pc()
      break
   elif "add reminder" in command:
      set_reminder()
   elif "check reminders" in command:
      check_reminders()
   elif "add task" in command:
      add_task()
   elif "show task" in command:
      show_tasks()
   elif "clear task" in command:
      clear_tasks()
   elif "notepad" in command:
      open_notepad()
   elif "save note" in command:
      save_note()
   elif "get weather" in command:
      get_weather()
   elif "define" in command or "dictionary" in command:
```

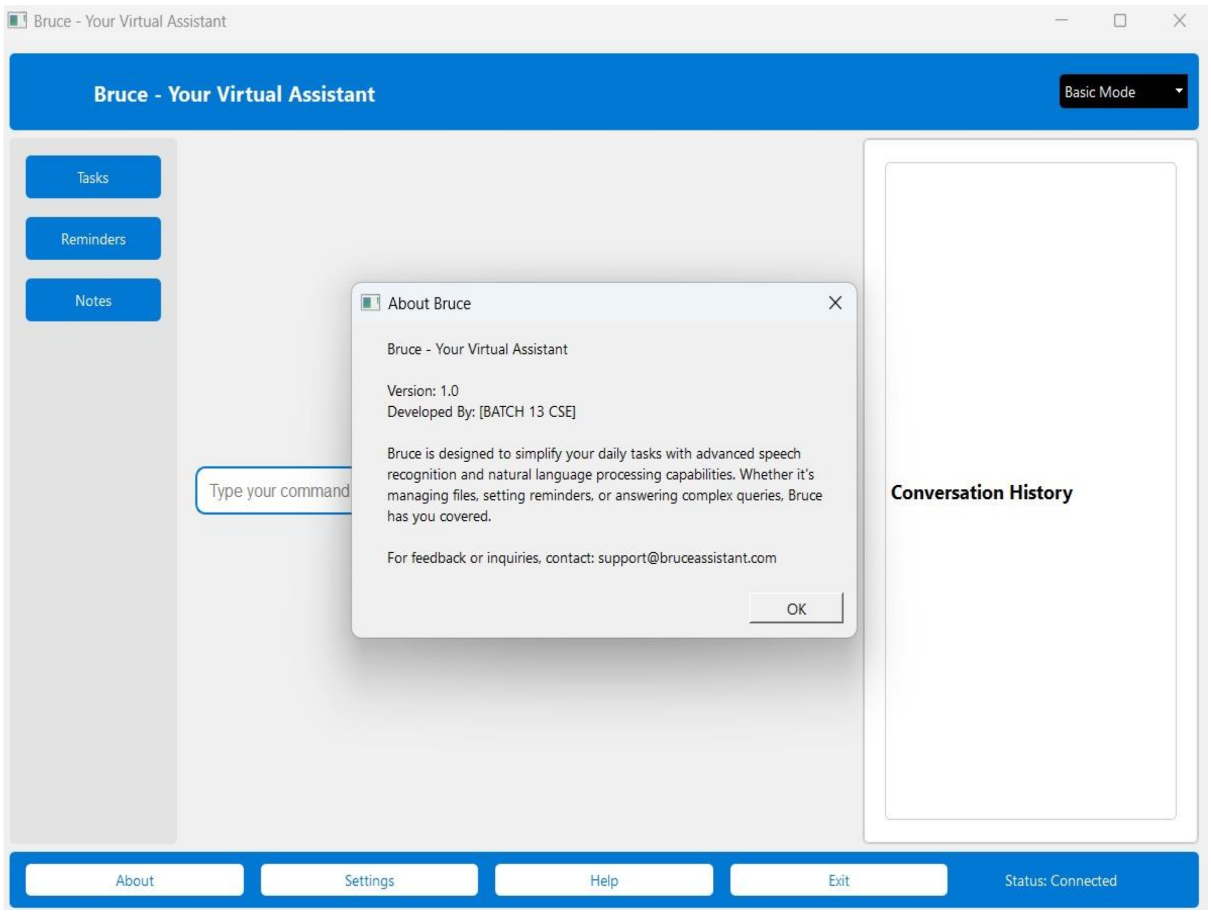
```
            get_meaning()
        elif "open quick access" in command:
            open_quick_access()
        elif "add shortcut" in command:
            add_shortcut()
        elif "execute shortcut" in command:
            execute_shortcut()
         elif "play Song" in command or "stop" in command or "next" in command or
"previous" in command or "volume up" in command or "volume down" in command
or "pause" in command:
            control_media(command)
        elif "exit" in command or "quit" in command:
            speak("Exiting Basic Mode. Goodbye!")
            break
        else:
            speak("Sorry, I didn't understand that.")

if __name__ == "__main__":
    speak("Welcome to Bruce. You are in Basic Mode.")
    basic_mode()
```

7. OUTPUT SCREENS:

Main Dash Board:



About Bruce:-



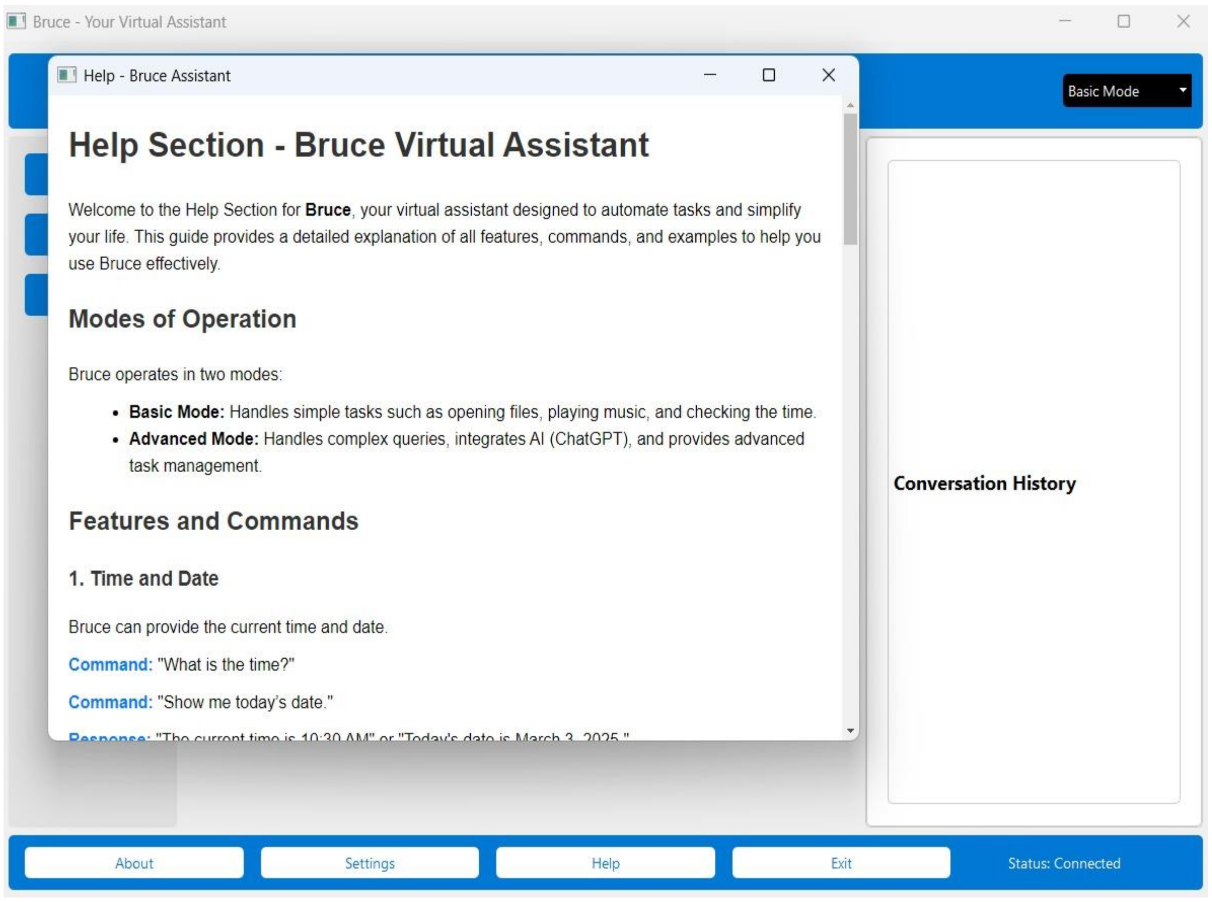
Bruce - Your Virtual Assistant

Version: 1.0
Developed By: [BATCH 13 CSE]

Bruce is designed to simplify your daily tasks with advanced speech recognition and natural language processing capabilities. Whether it's managing files, setting reminders, or answering complex queries, Bruce has you covered.

For feedback or inquiries, contact: support@bruceassistant.com

Help Section:



Help Section - Bruce Virtual Assistant

Welcome to the Help Section for **Bruce**, your virtual assistant designed to automate tasks and simplify your life. This guide provides a detailed explanation of all features, commands, and examples to help you use Bruce effectively.

Modes of Operation

Bruce operates in two modes:

- **Basic Mode:** Handles simple tasks such as opening files, playing music, and checking the time.
- **Advanced Mode:** Handles complex queries, integrates AI (ChatGPT), and provides advanced task management.

Features and Commands

1. Time and Date

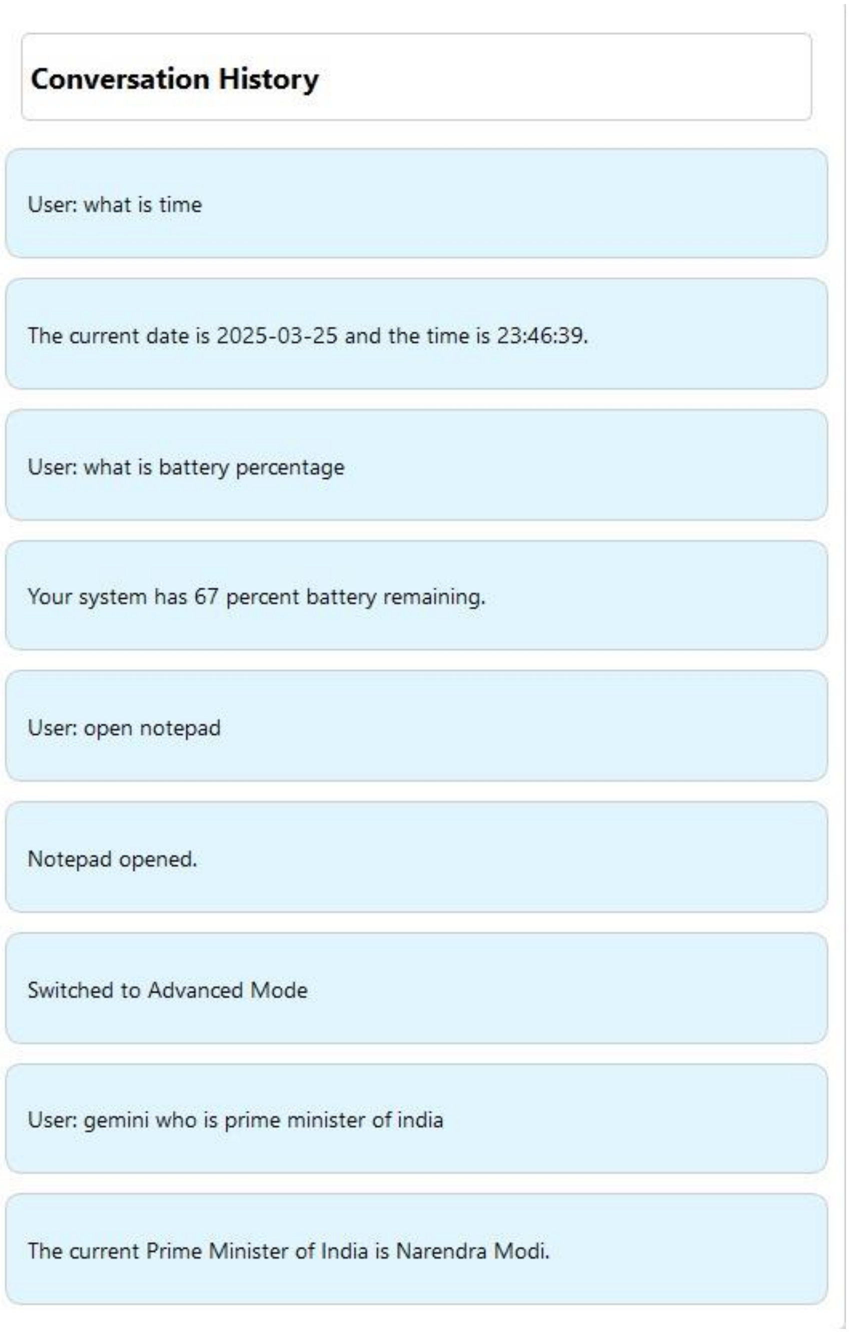
Bruce can provide the current time and date.

Command: "What is the time?"

Command: "Show me today's date."

Response: "The current time is 10:30 AM" or "Today's date is March 3, 2025."

Conversation History:



Conversation History

User: what is time

The current date is 2025-03-25 and the time is 23:46:39.

User: what is battery percentage

Your system has 67 percent battery remaining.

User: open notepad

Notepad opened.

Switched to Advanced Mode

User: gemini who is prime minister of india

The current Prime Minister of India is Narendra Modi.

8. CONCLUSION:

The Bruce desktop voice assistant is a versatile and innovative application that demonstrates the potential of voice technology to simplify and automate daily tasks. Designed with inclusivity and user-friendliness at its core, Bruce provides a dual-mode functionality, offering both a Basic and an Advanced Version to cater to a wide range of user needs. This dual-mode system ensures that Bruce is accessible to individuals with varying levels of technological expertise while meeting the requirements of both casual and power users.

The Basic Version focuses on offline capabilities and essential functionalities such as file management, setting reminders, media control, and system information display, ensuring that users can rely on it even in environments with limited or no internet connectivity. In contrast, the Advanced Version takes advantage of modern technologies like enhanced speech recognition, natural language processing, and API integrations to handle complex queries and deliver personalized, intelligent responses. Its integration with ChatGPT, multi-language support, and context-aware suggestions further elevate the user experience, making Bruce a powerful productivity tool.

Bruce has been designed with a focus on reliability and performance. Extensive testing across various scenarios, including unit testing, integration testing, and system testing, has validated the application's ability to function seamlessly across diverse environments. The inclusion of robust error handling, intuitive GUI design, and efficient task execution ensures that users encounter minimal disruptions while interacting with the application.

Beyond its technical functionalities, Bruce represents a step forward in enhancing accessibility for all users, including those who may not be familiar with advanced technologies. Its ability to bridge the gap between simplicity and sophistication makes it a suitable tool for students, professionals, and senior citizens alike. By offering both voice and text input options, Bruce accommodates users with different preferences and ensures inclusivity.

In conclusion, Bruce is more than just a voice assistant; it is a comprehensive desktop automation solution that empowers users to perform tasks efficiently and intuitively. With its blend of simplicity, intelligence, and adaptability, Bruce has the potential to set a benchmark for future developments in voice-assisted technology. As advancements in AI and speech recognition continue, Bruce is well-positioned to evolve further, incorporating emerging innovations and expanding its capabilities to meet the growing demands of a dynamic user base.

9. FUTURE ENHANCEMENT:

The Bruce desktop voice assistant is designed to evolve alongside advancements in technology and user needs. While the current implementation offers a robust set of features, several areas can be enhanced in the future to make the application even more versatile, intelligent, and user-centric. Below are some potential areas for future enhancements:

1. Enhanced AI and Machine Learning Integration:

- **Contextual Understanding:** Advanced machine learning models could be incorporated to improve Bruce's ability to understand and respond to nuanced and context-specific queries.
- **Personalization:** AI could be used to provide personalized recommendations for tasks, content, and applications based on user behavior and preferences.
- **Adaptive Learning:** Bruce could learn from user interactions over time, adapting to their needs and improving its response accuracy and efficiency.

2. Extended Multi-Language Support:

- **Natural Language Processing (NLP) Expansion:** The application could include support for more languages and dialects, ensuring accessibility for a global audience.
- **Real-Time Translation:** Future versions could offer real-time language translation to enable seamless communication across different languages.

3. Integration with Smart Home Devices:

- **IoT Compatibility:** Bruce could be enhanced to integrate with Internet of Things (IoT) devices, allowing users to control smart home appliances like lights, thermostats, and security systems.
- **Voice-Controlled Automation:** Advanced voice commands could automate daily routines, such as turning off all devices when leaving home or adjusting the environment for specific activities.

4. Advanced Security and Privacy Features:

- **Biometric Authentication:** Integration of biometric security, such as voice recognition or facial recognition, to ensure that only authorized users can access sensitive features.

- **Data Encryption:** Enhanced encryption methods could protect user data, ensuring secure communication and storage of sensitive information.
- **Customizable Privacy Settings:** Users could be given more control over what data is collected, stored, and shared.

5. Cross-Platform Synchronization:

- **Mobile and Web Versions:** Bruce could be extended to mobile devices and web platforms, enabling users to access its features across multiple devices.
- **Cloud Sync:** A cloud-based synchronization feature could allow users to access their tasks, reminders, and notes from any device, ensuring seamless continuity.

6. Expanded API Integrations:

- **Third-Party Services:** Integration with more third-party APIs, such as e-commerce platforms, healthcare applications, and transportation services, to enhance functionality.
- **Enterprise Tools:** Support for professional tools like Slack, Microsoft Teams, and Trello to make Bruce suitable for workplace environments.

7. Offline Capabilities:

- **Advanced Offline Mode:** Future enhancements could include a more comprehensive offline mode, allowing users to perform complex tasks without an internet connection.
- **Local Data Processing:** Implementing local AI models could reduce dependency on the cloud, ensuring faster responses and improved privacy.

8. Enhanced GUI and User Experience:

- **Dynamic Interface:** The interface could be made more dynamic and interactive, with customizable themes and layouts to suit individual preferences.
- **Visual Assistive Features:** Adding visual aids, such as screen magnification and text-to-speech visual prompts, for accessibility to users with disabilities.

9. Gamification Elements:

- **Task Rewards:** Gamified features could encourage users to complete tasks, such as earning points for setting reminders or managing files efficiently.

- **Learning Modules:** Bruce could provide interactive tutorials for new users to quickly understand its capabilities and functions.

10. Predictive Analytics:

- **Proactive Suggestions**: Bruce could predict user needs based on patterns, such as suggesting reminders, optimizing workflows, or anticipating frequently used commands.
- **Health and Productivity Tracking:** Future versions could monitor user productivity or suggest health-related actions, such as taking breaks or staying hydrated.

In conclusion, Bruce’s development roadmap is filled with opportunities for innovation and enhancement. By integrating emerging technologies, expanding its feature set, and prioritizing user-centric improvements, Bruce can continue to remain at the forefront of desktop automation and voice assistant technology.

10. REFERENCES:

1. P. Kunekar, A. Deshmukh, S. Gajalwad, A. Bichare, K. Gunjal, and S. Hingade, "AI-based Desktop Voice Assistant," *2023 5th Biennial International Conference on Nascent Technologies in Engineering (ICNTE)*, Navi Mumbai, India, 2023, pp. 1-4, doi: 10.1109/ICNTE56631.2023.10146699.
2. V. Jain, Y. Patidar, U. Jain, V. Parwal, A. K. Kumawat, and S. Sureliya, "Automate Personal Voice Based Assistant Using Python," *2024 4th Asian Conference on Innovation in Technology (ASIANCON)*, Pimpri Chinchwad, India, 2024, pp. 1-5, doi: 10.1109/ASIANCON62057.2024.10838003.
3. J. Vijaya, C. Swati, and S. Satya, "Ikigai: Artificial Intelligence-Based Virtual Desktop Assistant," *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, Gwalior, India, 2024, pp. 1-6, doi: 10.1109/IATMSI60426.2024.10502959.
4. V. Titarmare, P. H. Chandankhede, and M. Wanjari, "Interactive Zira Voice Assistant- A Personalized Desktop Application," *2023 2nd International Conference on Paradigm Shifts in Communications Embedded Systems, Machine Learning and Signal Processing (PCEMS)*, Nagpur, India, 2023, pp. 1-6, doi: 10.1109/PCEMS58491.2023.10136067.
5. R. P., K. Suresh, B. M., M. Gokul, G. D. Kumar, and A. R., "IoT Based Voice Assistant using Raspberry Pi and Natural Language Processing," *2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, Chennai, India, 2022, pp. 1-4, doi: 10.1109/ICPECTS56089.2022.10046890.
6. S. Gowroju, S. Kumar, and S. Choudhary, "Natural Language Processing-Driven Voice Recognition System for Enhancing Desktop Assistant Interactions," *2024 7th International Conference on Contemporary Computing and Informatics (IC3I)*, Greater Noida, India, 2024, pp. 1136-1141, doi: 10.1109/IC3I61595.2024.10829162.
7. R. Benny, A. Muralidharan, and M. Subramanian, "OpenAI-Enhanced Personal Desktop Assistant: A Revolution in Human-Computer Interaction," *2024 Second International Conference on Emerging Trends in Information Technology and Engineering (ICETITE)*, Vellore, India, 2024, pp. 1-7, doi: 10.1109/ic-ETITE58242.2024.10493339.
8. S. P. Yadav, A. Gupta, C. Dos Santos Nascimento, V. Hugo C. de Albuquerque, M. S. Naruka, and S. Singh Chauhan, "Voice-Based Virtual-Controlled Intelligent Personal

Assistants," *2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)*, Ghaziabad, India, 2023, pp. 563-568, doi: 10.1109/CICTN57981.2023.10141447.